[**Interview Questions ASP.NET Core**](http://www.aspdotnet-suresh.com/2010/05/interview-questions-in-aspnetcnetsql.html)

**Que:AddTransient, AddScoped and AddSingleton Services Differences**

* Transient objects are always different; a new instance is provided to every controller and every service.
* Scoped objects are the same within a request, but different across different requests
* Singleton objects are the same for every object and every request (regardless of whether an instance is provided in ConfigureServices)

<https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences>

**Transient**

* since they are created every time they will use **more memory** & Resources and can have the **negative** impact on performance
* use this for the **lightweight** service with little or **no state**.

**Scoped**

* better option when you want to maintain state within a request.

**Singleton**

* memory leaks in these services will build up over time.
* also memory efficient as they are created once reused everywhere.

**1. What are some characteristics of .NET Core?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

* Flexible deployment: Can be included in your app or installed side-by-side user- or machine-wide.
* Cross-platform: Runs on Windows, macOS and Linux; can be ported to other OSes. The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.
* Command-line tools: All product scenarios can be exercised at the command-line.
* Compatible: .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library.
* Open source: The .NET Core platform is open source, using MIT and Apache 2 licenses. Documentation is licensed under CC-BY. .NET Core is a .NET Foundation project.
* Supported by Microsoft: .NET Core is supported by Microsoft, per .NET Core Support

**2. What is CTS?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

The Common Type System (CTS) standardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages. CTS is designed as a singly rooted object hierarchy with System.Object as the base type from which all other types are derived. CTS supports two different kinds of types: - Value Types: Contain the values that need to be stored directly on the stack or allocated inline in a structure. They can be built-in (standard primitive types), user-defined (defined in source code) or enumerations (sets of enumerated values that are represented by labels but stored as a numeric type). - Reference Types: Store a reference to the value‘s memory address and are allocated on the heap. Reference types can be any of the pointer types, interface types or self-describing types (arrays and class types such as user-defined classes, boxed value types and delegates).

**3. What is Zero Garbage Collectors?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Zero Garbage Collectors is the simplest possible implementation that in fact does almost nothing. It only allows you to allocate objects because this is obviously required by the Execution Engine. Created objects are never automatically deleted and theoretically, no longer needed memory is never reclaimed.

**4. Explain a simple GC implementation?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

It is an excellent basis for the development of your own Garbage Collection mechanism. It provides the necessary functionality to make runtime work properly and you can build on top of that.

It may be interesting for special use cases like very short living applications or such that almost no allocate memory (you can come up with those concepts as No-alloc or Zero-alloc programming). In such a case providing GC overhead is unnecessary and it may be wise to get rid of it. It is like making huge GC.TryStartNoGCRegion overall your application.

**5. When does Garbage collection occur?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Garbage collection occurs when one of the following conditions is true.

* The system has low physical memory.
* The memory that is used by allocated objects on the managed heap surpasses an acceptable threshold. This threshold is continuously adjusted as the process runs.
* The GC.Collect method is called and in almost all cases, you do not have to call this method, because the garbage collector runs continuously. This method is primarily used for unique situations and testing.

**6. What is CoreFx?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

CoreFx is the reimplementation of the class libraries for .NET Core.

**7. What is unit testing?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Unit testing is a development process for the software that has the smallest testable parts of an application, which are called units. They are individually and independently scrutinized for any proper operation. Unit testing is can either be automated or done manually as well.

**8. What are Empty migrations?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Sometimes it's useful to add a migration without making any model changes. In this case, adding a new migration creates code files with empty classes. You can customize this migration to perform operations that don't directly relate to the EF Core model. Some things you might want to manage this way are: - Full-Text Search - Functions - Stored procedures - Triggers - Views

**9. Differences Between .net Core and .net Framework?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

The differences between the two can be summarized in these three points:

**NuGet-based**: .NET Core is distributed as a set of NuGet packages that allow app-local deployments. In contrast, the .NET-Framework is always installed in a system-wide location. This difference doesn’t matter so much for class libraries, but it matters for applications as those are expected to deploy the closure of their dependencies. But we expect this model to change how quickly class library authors can take advantage of new functionality. Since the applications can simply deploy a new version (as opposed to having to wait until a given .NET Framework version is widely adopted), there is less of a penalty for component authors to take advantage of the latest features.

**Well layered**: .NET Core was specifically designed to be layered. The goal was to create a .NET stack that can accommodate a wide variety of capabilities and system constraints without forcing customers to recompile their binaries and/or produce new assets. This means that we had to remove certain APIs because they tied lower-level components to higher-level components. In those cases, we provide alternatives, often in the form of extension methods.

**Free of problematic tech**: .NET Core doesn’t include certain technologies we decided to discontinue because we found them to be problematic, for instance, AppDomain and sandboxing. If the scenario still makes sense for .NET Core, our plan is to have replacements. For example, AssemblyLoadContext replaces AppDomains for loading and isolating assemblies.

**10. What are complex and supporting methods?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

* The System.Numerics.Complex type represents a complex number, i.e., a number with a real number part and an imaginary number part
* It supports a standard set of arithmetic, comparison, equality, explicit conversion, and implicit conversion operators, as well as mathematical, algebraic, and trigonometric methods.

**11. What is FCL?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Framework Class Libraries (FCL)- The Framework class library (FCL) is a comprehensive collection of reusable types including classes, interfaces and data types included in the .NET-Framework to provide access to system functionality.

**12. What is IGCToCLR interface?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

This interface passed as an argument to the function InitializeGarbageCollector is used to communicate with the runtime. It contains quite a lot of available methods and listing them all here is pointless.

**13. What is Modularity?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Modularity leads to performance benefits and your application can run faster, especially ASP.NET Core application.

**14. Defines the types and methods of a class library?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

* A class the library defines the types and methods that can be called from any application.
* A class the library developed using .NET Core supports the .NET Standard Library, which allows your library to be called by any .NET platform that supports that a version of the .NET Standard Library.
* When you finish your class library, you can decide whether you want to distribute it as a third-party component, or whether you want to include it as a component that is bundled with one or more applications.

**15. What are the Fundamental Libraries?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Fundamental Libraries − A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities.

**16. What are Generate SQL scripts in .Net core?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

When debugging your migrations or deploying them to a production database, it's useful to generate a SQL script. The script can then be further reviewed for accuracy and tuned to fit the needs of a production database. The script can also be used in conjunction with a deployment technology.

**17. Explain the difference between Task and Thread in .NET**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Thread represents an actual OS-level thread, with its own stack and kernel resources. Thread allows the highest degree of control; you can Abort() or Suspend() or Resume() a thread, you can observe its state, and you can set thread-level properties like the stack size, apartment state, or culture. ThreadPool is a wrapper around a pool of threads maintained by the CLR.

The Task class from the Task Parallel Library offers the best of both worlds. Like the ThreadPool, a task does not create its own OS thread. Instead, tasks are executed by a TaskScheduler; the default scheduler simply runs on the ThreadPool. Unlike the ThreadPool, Task also allows you to find out when it finishes, and (via the generic Task) to return a result.

**18. When should we use .NET Core and .NET Standard Class Library project types?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Use a .NET Standard library when you want to increase the number of apps that will be compatible with your library, and you are okay with a decrease in the .NET API surface area your library can access.

Use a .NET Core library when you want to increase the .NET API surface area your library can access, and you are okay with allowing only .NET Core apps to be compatible with your library.

**19. What are the advantages that you could speak about Web API?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Content Negotiation, routing, and model bindings are some of the top picks that one could say when it comes to the advantages of Web API.

**20. What is meant by Web API routing?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

It is a pattern matching routine that is similar to what is found in MVC architecture. Route Tables contain all the routes that have been registered.

**21. Briefly explain exception filters.**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Exceptions that are thrown but are not handled will lead to the execution of exception filters. IExceptionFilter interface is implemented by the exception filters.

**22. Explain the difference between a class and an object.**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

In short, a class is the definition of an object, and an object is instance of a class.

We can look at the class as a template of the object: it describes all the properties, methods, states and behaviors that the implementing object will have. As mentioned, an object is an instance of a class, and a class does not become an object until it is instantiated. There can be more instances of objects based on the one class, each with different properties.

**23. Explain LINQ.**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

LINQ is an acronym for Language Integrated Query, and was introduced with Visual Studio 2008. LINQ is a set of features that extends query capabilities to the .NET language syntax by adding sets of new standard query operators that allow data manipulation, regardless of the data source. Supported data sources are: .NET Framework collections, SQL Server databases, ADO.NET Datasets, XML documents, and any collection of objects that support IEnumerable or the generic IEnumerable interface, in both C# and Visual Basic. In short, LINQ bridges the gap between the world of objects and the world of data

**24. Explain the difference between the Stack and the Heap.**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

The short answer would be: in the Stack are stored value types (types inherited from System.ValueType), and in the Heap are stored reference types (types inherited from System.Object).

We can say the Stack is responsible for keeping track of what is actually executing and where each executing thread is (each thread has its own Stack). The Heap, on the other hand, is responsible for keeping track of the data, or more precise objects.

**25. What is a delegate in .NET?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

A delegate in .NET is similar to a function pointer in C or C++. Using a delegate allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked. In addition, we could use delegate to create custom event within a class.

**26. Define encapsulation.**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Encapsulation is a function that includes various methods and data within a project. This is done so that the object of the program could perform its functions smoothly and without any errors.

**27. Is there a difference between ‘debug’ and ‘trace’?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Yes. The Trace class can be used for debugging and releasing certain builds, while Debug is used solemnly for – you’ve guessed it – debugging.

This can be classified as one of the trick .NET interview questions, for it is quite easy to get forget about Trace’s additional function.

**28. What’s inheritance?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Although this isn’t necessarily one of the primary .NET interview questions, it is still often asked because of how it relates to .NET.

Inheritance happens when one smaller class takes on the features and parameters of another, bigger class. This bigger class is then seen as a “parent class” to the smaller one.

This is also one of the better .NET framework interview questions to expand upon. .NET supports only single inheritance. What this means is that the smaller, child class can only benefit (inherit) from one parent class.

**29. Define caching.**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Caching is a process when you keep your most often used files and data in a separate memory. This separate location – a cache – is where you can access all of your designated files. Caching saves developers a lot of time and increases their memory management.

**30. What is a ‘.dll’ file?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

DLL files are those which need to be hidden out of plain sight. The term directly abbreviates to “Dynamic Link Library”. These libraries are vast, containing a huge amount of files and commands. DLLs can also be shared among other programs and apps. A small piece in .NET interview questions but can make a huge difference.

**31. Can you specify access modifiers in an interface?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Assuming that the question relates to the item access modifiers – no, no you can’t. Why? Simply because the interface is always public.

**32. What are advantages of Garbage Collection?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

Garbage Collection provides the following benefits: - You don’t need to free the memory manually while developing your application. - It also allocates objects on the managed heap efficiently. - When objects are no longer used then it will reclaim those objects by clearing their memory, and keeps the memory available for future allocations. - Managed objects automatically get clean content to start with, so their constructors do not have to initialize every data field. - It also provides memory safety by making sure that an object cannot use the content of another object.

**33. What is UWP App in .Net core?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

UWP apps will be able to use libraries you have created in .net core as long as you target the .netstandard1.6 (or higher) framework moniker. UWP is only for the Windows ecosystem.

**34. What is MSBuild and how it works with .NET Core?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

MSBuild is the build platform for Microsoft and Visual Studio. In the UWP application if you open the project folder, then you will see both project.json and \*.csproj files. But if you open our previous .NET Core Console app, then you will see project.json and \*.xproj files.

**35. How does MEF work?**[**↑**](https://www.adaface.com/blog/dot-net-core-interview-questions/#questionsindex)

It allows application developers to discover and use extensions with no configuration required. MEF is an integral part of the .NET Framework 4 and is available wherever the .NET-Framework is used that improves the flexibility, maintainability, and testability of large applications

https://www.chubbydeveloper.com/top-20-asp-net-core-interview-questions/

## What functionalities are supported by ASP.NET Core?

* Built-in support for dependency injection
* Built-in support for an extensible logging framework
* Contains Kestrel – a cross-platform web server that allows web applications to execute without the need for IIS, Apache or Nginx.
* Functionality to use multiple hosts
* Supports modularity – the developer must include the module that is required by the application.
* Makes use of an app JSON file to store settings instead of a web.config
* Contains a startup class to initiate and run services (Instead of global. asax)
* Has extensive support for asynchronous programming

## What is ASP.NET Core Middleware?

ASP.NET Core Middleware is a group of small modules that are incorporated into an HTTP request pipeline. Middleware is used to implement several tasks when handling requests. Such examples include authentication, session state retrieval and persistence, logging and much more. It gives you control on the order of when the requests should be executed, unlike ASP.NET Core HTTP modules

Middleware is configured by code rather than web.config in ASP.NET. It is found inside your Startup.cs file:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

app.UseDeveloperExceptionPage();

app.UseHttpsRedirection();

app.UseExceptionHandler();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

app.UseStaticFiles();

}

The configure method as seen above is used to create a request pipeline via middlewares. The method applies the following functionality:

* Adds error pages and error handlers
* Adds HTTPS Redirection
* Adds support for static files
* Makes use of ASP.NET Identity authentication
* Adds routing and endpoint using map controllers

## What is the difference between ASP.NET and ASP.NET Core?

ASP.NET Core is a total rewrite of ASP.NET. It is an open-source cross-platform that allows you to build web applications in Windows, Mac and Linux. It is also capable of working with both .NET Core and .NET Framework.

Here are the main differences:

* Project configuration – ASP.NET Core does not make use of web.config. Instead, appsettings.json or other custom configuration files are used. A new folder named wwwroot is also added to the project structure. This is the container for all static files which are sent to the browser such as CSS, HTML, JavaScript and image files.
* Not dependent on IIS – It is not IIS dependent like ASP.NET and allows you to host on IIS, Docker, Nginx, Kestral and Apache.  
  Installation – since its cross-platform, frameworks are prepackaged and compiled via NuGet.
* Microservices – Microservices were simplified in .NET Core. Programmers can develop custom microservices and combine them to build powerful systems seamlessly.

## What is Kestral?

Kestrel is a web server build for ASP.NET core which is cross-platform

* It is based on libuv – a cross-platform asynchronous I/O library
* It is the default web server for all ASP.NET Core templates.
* It is fast and secure and can even be used without a reverse proxy server. However, it is still recommended to use with IIS, Nginx or Apache.

## What are Razor Pages in ASP.NET Core?

As of ASP.NET Core 2.1, Razor pages were introduced. It follows a page-development model like in ASP.NET web forms.  
Razor pages start with a @page directive. This means that it handles requests directly without the need to go through the controller.

Here is an example of the page and model directive in the CSHTML file and its associated page model class (CSHTML.CS)

@page

@model AboutModel

<div>

The current time is @Model.CurrentTime.ToShortTimeString()

</div>

--About.cshtml.cs---

using System;

using Microsoft.AspNetCore.Mvc.RazorPages;

public *class* AboutModelModel : PageModel

{

public DateTime CurrentTime = DateTime.UtcNow;

}

When comparing with the classic Home Controller example in ASP.NET MVC, the equivalent of razor pages would be Index (Home), About, Contact and Error pages all in the root directory.  
For example in each page, you will find the HTML page (cshtml) and its page model class page (cshtml.cs). This is in contrast with MVC approach where controllers, view models and view models all had different locations in entirely different folders.

## Can ASP.NET Core work with .NET Framework?

Yes. ASP.NET Core can work with .NET framework using the .NET standard library.

## What’s the difference between .NET Core .NET Framework and .NET Standard?

.NET Standard is a specification for implementing the Base Class Library (BCL). BCL contains classes such as exception handling, XML, collections, I/O and networking. WPF, WCF and ASP.NET do not form part of BCL and so are not included in .NET Standard library.

* NET Core is a managed framework that builds desktop and web applications in cross-platform.
* .NET Framework is a framework that builds desktop and web applications in Windows only. It is highly dependent on the architecture.
* Both “.NET Core” and “.NET Framework” include .NET Standard for BCL in their managed framework.

## What is routing in ASP.NET Core?

In ASP.NET Core, routing is the process of mapping incoming requests to the route handler. The route can have arguments inside the URL which are then used to process the call. Routing works by finding the route handler based on the URL given. There are two types of routing in ASP.NET Core.

Attribute – used in REST APIs  
Conventional – used with controllers and views in MVC architecture

endpoints.MapControllerRoute( name: "default", pattern: "{controller=Home}/{action=Index}/{id?}");

Above is an example to map both conventionally routed controllers and attribute controllers.

## How do you enable a session in ASP.NET Core?

The package Microsoft.AspNetCore.Session provides the middleware for the session. To make use of this session in an ASP.NET Core application, you need to include this package in its csproj file. The Session middleware must then be added in the Startup file in 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();

}

}

## Advantages of ASP.NET Core over ASP.NET

Here are a number of reasons why ASP.NET Core is better than ASP.NET:

1. It is cross platform (Windows/Linux/Mac/Android/iOS)
2. There exists no dependencies in the framework installation because the packages that are required are directly integrated with application
3. It handle a higher number of requests than ASP.NET
4. ASP.NET Core offers numerous deployment options.

## What is WebListener?

WebListener is a web server in ASP.NET Core that runs only on Windows host machines. It is an alternative to Kestrel and is built on HttpSys kernel-mode driver. Also, is used for direct connection to the internet without the need of an IIS as a reverse proxy server. It is not compatible with IIS.

## What is IWebHostEnvironment interface used for?

**IWebHostEnvironment** is an interface for .NET Core. It is used to get information on the web hosting environment an application is executing in. It needs to be injected as a dependency on the controller. The interface has two properties:

* **WebRootPath** – Path of the WWW folder
* **ContentRootPath** – Path of the root folder which includes all the application files.

## What are the Service lifetimes?

Service lifetimes define the conditions in which a new service instance will be instantiated. Here are the three types of service lifetimes defines by the .NET Core DI framework:

Transient – Instance is created every time they are requested.  
Scoped – Per every web request or any unit of work. We call this a scope.  
Singleton – Created only for the first request. If the singleton instance is specified at registration time, the instance will be available to all consumers of the registration type.

//Transient

services.AddTransient<ILifeServiceExample, LifeServiceExample>();

//Scoped

services.AddScoped<ILifeServiceExample, LifeServiceExample>();

//Singleton

services.AddSingleton<ILifeServiceExample, LifeServiceExample();

## What is wwwroot folder in ASP.NET Core?

By default. the wwwroot is the root folder that contains the static files such as HTML, CSS and Javascript.

The files can be stored here and accessed with a relative path to the root. Only these files inside the wwwroot can be served over HTTP Requests. All other files are filtered out and cannot be served by default.

The files can be accessed in this format:

http://domain:<port>/html/app.html

To set this up you need to add a middleware for serving static files in the Configure method of your Startup.cs.

## What is the difference between UseIIS & UseIISIntegration?

Before ASP.NET Core 2.2, ASP.NET Core was hosted outside of the IIS process. This meant that we had two processes for ASP.NET core application:

w3wp.exe – the IIS Process  
dotnet.exe – the ASP.NET Core process. This is where Kestrel web server was started.

IIS and Kestrel communicated between these two mentioned processes. In this case, you need to use UseIISIntegration.

However, in later versions, ASP.NET Core introduced in-process hosting. ASP.NET Core no longer ran separately but runs inside IIS w3wp.exe process. This removes the need for the Kestrel web server. For this case, you would need to specify UseIIS.

What is Dependency Injection and how is it implemented in ASP.NET Core?

A dependency is any object that is required from another object. Here is an example of a dependency:

public *class* DependencyClass

{

public DependencyClass() { }

public Task SayHello(string name)

{

Console.WriteLine($"Hello to you, {name}"); return Task.FromResult(0);

}

}

Now, an instance can be created from DependencyClass to get access to ‘SayHello’ method. DependencyClass now becomes a dependency of the AboutModel. See example below:

public *class* AboutModel : PageModel

{

DependencyClass \_dependencyClass = new DependencyClass();

public async Task IntroduceAsync()

{

await \_dependencyClass.SayHello("AboutModel.IntroduceAsync created this message");

}

}

This creates a problem because code dependencies (as above example) are problematic and it is recommended not to use where possible. Reasons for this are the following:

In order to replace DependencyClass with a different implementation, the class itself must be modified.  
If DependencyClass has dependencies, they must be set up by the class. With large projects containing multiple classes depending on DependencyClass, this will be cumbersome.  
It will be hard to unit test. DependencyClass cannot be mocked or stubbed in this case.

This is where Dependency injection comes in. This is how it addresses these problems in ASP.NET Core:

Interface or base class as an abstraction for the dependency implementation (IDependencyClass)

The dependency must be registered as a service inside a service container. ASP.NET Core contains a built-in service container, IServiceProvider. Services are registered in the app’s Startup configure method.

The service is then injected into the constructor in the class it’s used in. The framework takes care of instantiating the dependent and disposing of it.

The above technique is called Inversion of Control (IoC) between classes and their dependencies. For more information and samples how to do this, see here.

## What are technologies discontinued in .NET Core?

he following technologies have been discontinued in .NET Core:

* **Reflection** – Has been converted into a lightweight version. An extension method called GetTypeInfo exposes information that you normally retrieve from Type object. However, it is not as detailed as the original.
* **AppDomain** – AppDomains isolate apps from each other. They require runtime support and are costly. Creating more app domains is unsupported and there aren’t any plans to add this in the future. Microsoft recommends using separate processes or containers as an alternative.
* **Remoting** – .NET Remoting created several issues in its architectural design. Since it’s used for cross-AppDomain, which is no longer supported, it was decided to not support it either. It also cost them a lot due to runtime support.
* **Security Transparency** – Due to security reasons, Security transparency is no longer supported. It used to separate sandbox code from security-critical code in a declarative method. Microsoft recommends you use security boundaries provided by the operating system itself. Such examples include virtualization, contained or user accounts.

## What does Configure Services method do in the startup class?

ConfigureServices method takes care of registering services which are consumed across the application using Dependency Injection (DI) or Application Services.

public void ConfigureServices(IServiceCollection services)

{

services.AddEntityFramework()

.AddSqlServer()

.AddDbContext<MusicContext>()

.AddControllersWithViews().AddRazorPages();

}

## What is a Self-hosted Web Application?

SP.NET Core is fully decoupled from the web server environment hosting the application. It still supports IIS and IIS Express but by default, it uses self-hosting scenarios by using Kestrel and WebListener HTTP Servers.

Here is a sample of how to configure a web application that is self-host:

public *class* Program

{

public static void Main(string[] args)

{

var config = new ConfigurationBuilder()

.AddCommandLine(args)

.AddEnvironmentVariables(prefix: "ASPNETCORE\_").Build();

var host = new WebHostBuilder()

.UseConfiguration(config)

.UseKestrel()

.UseContentRoot(Directory.GetCurrentDirectory())

.UseStartup<Startup>().Build();

host.Run();

}

}

## What is a .NET Generic Host?

As of ASP.NET Core 3, they exported ASP.NET code to run on a .NET Generic Host instead of the previously used WebHost. .NET Generic Host is a non-web version of the WebHost that runs on ASP.NET Core. It is an abstraction layer for both Web and non-web functionality.

They decoupled HTTP pipeline from the Web Host API to enable a wider collection of host environments. For example, non-HTTP workloads that used to be in Web Host API were background tasks, messaging, dependency injection (DI), logging and more. Now they have been separated completely using the Generic host. With this abstraction, developers are able to use these mechanisms for console applications, systemmds, windows services, and web applications.

How to configure Logging in ASP.NET Core

ASP.NET Core supports a logging API that works with multiple built-in and third-party logging providers. To set up logging in ASP. NET Core you need to apply the following;

public static IHostBuilder CreateHostBuilder(string[] args)

{

Host.CreateDefaultBuilder(args).ConfigureLogging(logging =>

{

logging.ClearProviders();

logging.AddConsole();

logging.AddDebug();

logging.AddTraceSource("Information");

}).ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

In the above example, the ConfigureLogging() method takes action to delegate Action<ILoggingBuilder> to configure logging providers. To add custom logging providers, start off by removing all default providers. To do this call ClearProviders(). In our example, we add the Trace Souce and Debug providers instead. We also add the Console logging provider to add logs to console too.

## What is the difference between Pages and Views ?

Pages is a new page framework that was introduced in .NET core 2.0. Pages is an evolution of the old WebFroms and provides an easier way to generate pages as opposed to asp.net MVC.

The Razor Page is very similar in concept to the standard Model View Controller framework. In main difference is that the Model and the controller are included inside the page. Razor Pages provide a simpler development approach when compared to the standard MVC framework.

**https://www.qfles.com/interview-question/asp-net-core-interview-questions**

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](https://www.qfles.com/interview-question/angular-interview-questions)
* Blazor allow you to use [C# code](https://www.qfles.com/interview-question/c-sharp-interview-questions) in browser with [JavaScript code](https://www.qfles.com/interview-question/java-script-interview-questions).

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](https://www.qfles.com/interview-question/csharp-design-patterns-interview-questions) that's used as a technique to achieve the [Inversion of Control (IoC)](https://www.qfles.com/interview-question/solid-principles-interview-questions) 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](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-5.0)

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

For more about request processing pipeline for ASP.NET MVC visit [Request Processing Pipeline](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0#middleware-order).

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**](https://www.qfles.com/interview-question/csharp-design-patterns-interview-questions) - 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.

### Other Interview Questions For .NET Developers:

* [Angular Questions for .NET Developers](https://www.qfles.com/interview-question/angular-interview-questions)
* [Azure Cloud Questions for .NET Developers](https://www.qfles.com/interview-question/microsoft-azure-interview-questions)

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](https://www.qfles.com/blogs/technology/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](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-5.0)

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](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-5.0)

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](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-3.1)

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](https://www.qfles.com/interview-question/docker-interview-questions)
* For more about .NET and Docker visit [.NET and Docker](https://docs.microsoft.com/en-us/dotnet/core/docker/introduction) and [Docker images for ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images?view=aspnetcore-5.0)

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](https://docs.microsoft.com/en-us/aspnet/core/performance/caching/overview?view=aspnetcore-6.0).

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](https://www.qfles.com/interview-question/solid-principles-interview-questions).

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)](https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-3.1)

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](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-6.0).

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?

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