**Dot Net Core**

**1. What is .NET Core Framework, and how does it work?**

.NET Core framework provides an open-source, accessible, and general-purpose platform to create and run applications onto different operating systems. The framework follows the object-oriented programming principles that we can use C#, .NET, VB, Perl, Cobol, etc., programming languages. The framework provides various built-in tools such as packages, classes, libraries, APIs, and other functionalities. We can create a diverse range of applications.

It works as follows:

Once you have finished developing codes for required applications, you need to compile those application codes to Common Intermediate Language.

The framework uses an assembly file to store the compiled code with an extension (.dll or .exe)

Now, the Common Language Runtime (CLR) of the framework convert the compiled code to machine code (executable code) using the Just In Time (JIT) compiler.

At last, we can execute this executable code on any specific architecture used by developers.

**2. What is the latest version of .NET Core? Share one specific attribute.**

The latest version of .NET Core is .NET Core 7.0, and its release date is July 12 2022, according to Microsoft Documentation. The newest release includes the .NET Runtime and ASP.NET Core Runtime. It has introduced Android, iOS, and macOS SDKs for developing native applications. You can check this documentation to know the setup instructions and develop .NET MAUI applications.

**3. Share specific features of .NET Core?**

.NET Core has these 4 specific features:

Cross-platform: It supports various platforms and is executable on windows, macOS, and Linux. You can easily port the codes from one platform to another platform.

Flexibility: You can easily include codes in the desired app or install them per requirements. It means you can use one single consistent API model for all .NET applications with the help of the same library on various platforms.

Open Source: You can use it by downloading it from the Github library. You don't need to pay to purchase a license. The framework has been licensed under MIT and Apache.

Command-line tools: You can efficiently execute applications at the command line.

**4. What is .NET Core used for?**

You can use .NET Core in many ways:

For developing and building web applications and services that run on diverse operating systems

For creating Internet of Things applications and mobile backends

For using any development tools on any operating system

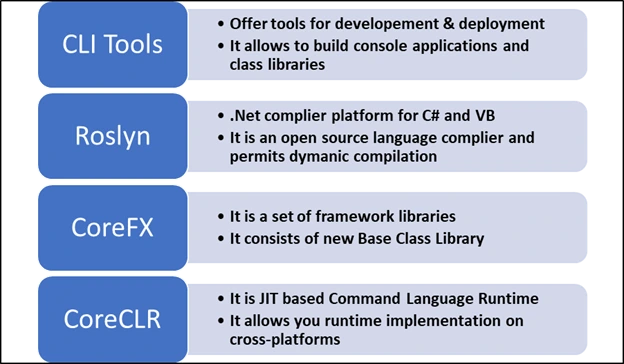
For creating and deploying applications to the cloud or other on-premises services.

Flexibility, high performance, and lightweight features allow for the development of applications quickly in containers deployable on all operating systems.

**5. Discuss critical components in .NET Core?**

Since .NET Core is a modular platform thus, its components could be stacked into these three layers:

A .Net runtime: It consists of different runtime libraries that allow you to perform functions such as type safety, load assemblies, garbage collections etc.

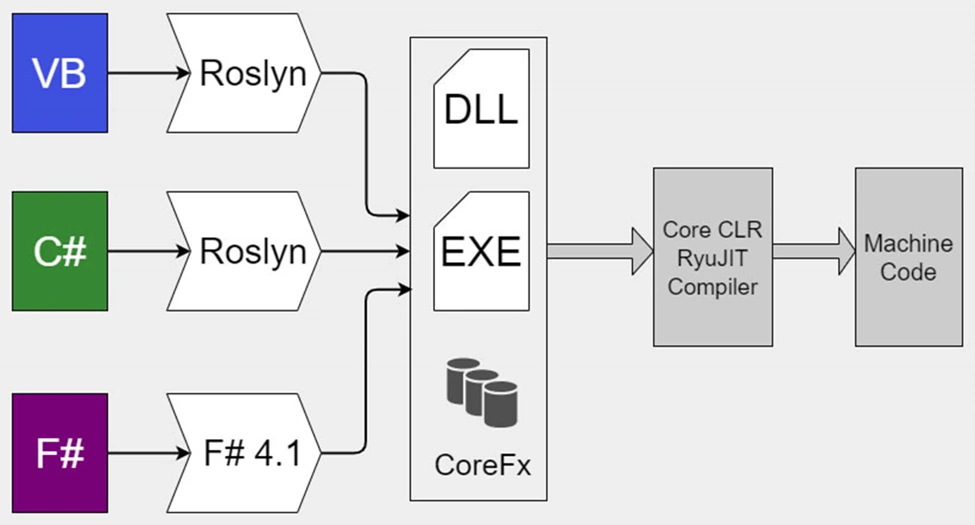
A collection of Framework libraries: It also consists of libraries that offer utilities, primitive data types, etc. A collection of SDK tools and compilers: It permits you to work with .NET Core SDK quickly. 

**6. What is .NET Core CoreFX?**

CoreFX is the introductive class library for .NET Core. It consists of collection types, file systems, console, JSON, and XML for class library implementation. You can use this code as a single portable assembly. Since it provides platform-neutral code, thus you can share it across different platforms.

**7. What is CoreCLR?**

CoreCLR is the .NET execution engine in .NET Core. It consists of a garbage collector, JIT compiler, low-level classes, and primitive data types. Garbage collection and machine code compilation are its primary functions.

The following image shows .NET Core Compilation. You can clearly write codes in different languages that compliers like Roslyn would comply with. The compiler will generate the respective CIL code used by the JIT compiler for further compilation. Since CoreCLR is embedded in the JIT compiler, it would eventually generate machine code. 

**8. How is .NET Core SDK different from .NET Core Runtime?**

.NET Core SDK builds applications, whereas .NET Core Runtime runs the application. Consider SDK is a collection of all tools and libraries you need to develop .NET Core applications quickly like a compiler, CLI. Consider Runtime as a virtual machine consisting of runtimes libraries and helps you run those applications.

**9. Where should you not use .NET Core?**

Consider these application areas where you should prevent using .NET Core

Avoid using current .NET framework applications in productions or migration because there is a possibility when you are unable to execute third libraries from apps running on the .NET core. Although, these libraries are executable from the .NET framework.

Avoid using .NET Core in designing loosely coupled and new large monolithic applications. It is because of computability issues while consuming libraries with the .NET framework. You can create such applications by running on the top of the .NET framework and with the help of CLR libraries.

Any applications that need sub frameworks like WPF, WebForms, Winforms as .NET Core don't support these.

Prevent trying .NET Core in applications requiring higher level frameworks such as WCF, Entity Framework, and Windows Workflow Foundation.

**10. What are the advantages of .NET Core?**

Cross-platform development and deployment: It can support application development on different platforms such as Windows, Linux, Mac, etc. Also, the deployment is supported on multiple platforms through containerization(Docker, Kubernetes, Service Fabric). This makes .NET completely portable and runnable on different platforms.

Open-source: All .NET source code and documentation is freely available for download and contribution. This results in faster software releases, enormous support, and usage of the latest tools in development.

Supports a plethora of applications: It has the capabilities to support a wide range of application types such as desktop, web, AI, cloud, mobile, IoT, gaming, etc.

Secure: Provides easy-to-incorporate security measures like authentication, authorization, and data protection. It has mechanisms to protect the sensitive-data like keys, passwords, connection strings, etc. For e.g. in terms of authentication, ASP.NET Core Identity allows you to integrate your app with all major external providers.

High performance: With every new release of the .NET core, the performance is improved for the benefit of users. For example, in .NET 5, the garbage collection is improved for faster speed, scalability, and reduction in memory resets’ cost. Detailed account of performance improvement in .NET 5.

Flexible: Provides the flexibility to use any database and infrastructure as per choice. It provides the ability to change, evolve and grow easily according to external factors.

**11. What is Kestrel?**

Kestrel is an event-driven, I/O-based, open-source, cross-platform, and asynchronous server which hosts .NET applications. It is provided as a default server for .NET Core therefore, it is compatible with all the platforms and their versions which .NET Core supports.

Usually, it is used as an edge-server, which means it is the server which faces the internet and handles HTTP web requests from clients directly. It is a listening server with a command-line interface.

Advantages of Kestrel are:

Lightweight and fast.

Cross-platform and supports all versions of .NET Core.

Supports HTTPS.

Easy configuration

**12. What do you know about .NET Core middleware?**

Middleware is a layer, software, or simple class through which all the requests and responses have to go through. The middleware is assembled of many delegates in an application pipeline. Each component(delegate) in the pipeline of the middleware decides :

To pass the request to the next component.

Perform some processing on the request before or after passing it.

**13. What are Razor Pages in .NET Core?**

Razor Pages is a new server-side framework which works on a page-based approach to render applications in .NET Core. They are stored as a physical .cshtmlfile.

They have the HTML and code in a single file, without the need to maintain separate controllers, view models, action methods, etc

Razor Pages framework is flexible, lightweight, cohesive, page-based, easy to learn and maintain compared to MVC. It can be used in conjunction with traditional MVC (Model-View-Controller) architecture or Web-API controllers.

**14. What are service lifetimes in .NET Core?**

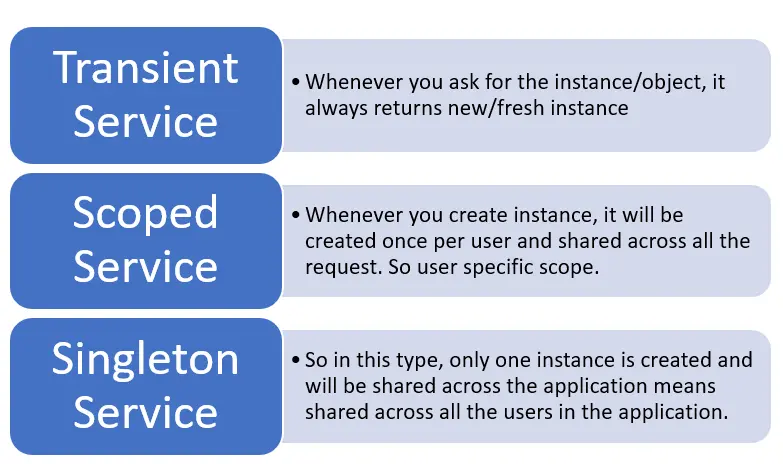
.NET Core supports a design pattern called ‘Dependency Injection’ which helps in the implementation of IoC(Inversion of Control). During registration, dependencies require their lifetime to be defined. The lifetime of service decides under what condition the instance of the service will be created and till what time it will be live.

There are three types of service lifetimes supported by .NET Core:

Transient Service: Instance is created each time it is requested.

Scoped Service: User-specific instance is created once per user and shared across all the requests.

Singleton Service: Single Instance is created once a lifetime of the application.



**15. Explain Docker in .NET Core.**

Docker is an open platform for developing, shipping, and running applications. It allows you to quickly isolate your applications from the infrastructure to transmit software. You should leverage this feature for managing infrastructure and deploying codes fast. It would help reduce the time needed between writing and running codes in infrastructure.

Three main functions:

Quick and constant delivery of applications

Responsive deployment and scaling

Efficiently run more workloads on the same hardware.

Take care following points while using Docker in .NET Core

You can use the Docker client's CLI for managing images and containers

You must adequately integrate Docker images, containers, and registries while designing and containerising applications or microservices

Use Dockerfile for rebuilding images and distribute them with others

**16. What is .NET Core CLI?**

.NET Core CLI is part of .NET SDK that provides a cross-platform toolset to develop, create, and run .NET Core applications. You can install multiple versions of the toolset on your machine. You can use the following standard syntax to use CLI.

It provides four types of commands

Basic commands: All commands required to develop applications like new, restore, build, run, etc.

Project Modification commands: It allows you to use existing packages or add packages for developing applications.

Advanced commands: It gives various commands to perform additional functions such as deleting nuget.

Tool management commands: You can use these commands to manage tools.

**17. What is Hosting Environment Management?**

It is a new feature of .NET Core that permits you to work with multiple environments with no friction. You can use this feature through the available interface, Hosting Environment. The interface has been open since the first run of the application. Its execution depends on the environment variable and switch between the configuration files during runtime.

The interface reads a specific environment variable named "ASPNETCORE\_ENVIRONMENT" and checks its value. Check its following values:

If value: Development – You are running the application in Dev mode

If value: Staging – You are running the application in staging mode

**18. Briefly explain Garbage Collection, its benefits, and its condition.**

Garbage collection is another powerful feature of .NET Core. The primary function of this feature is to manage memory allocation and release. The .NET Core has "Zero Garbage Collector" to execute this function. You can call it Automatic Memory Manager.

Benefits:

You don't need to put effort into releasing memory manually

Efficient object allocation on the heap

Ensure memory security by ensuring object's usage

You can reclaim objects that are no longer needed, free the memory, and use it for other purposes.

Three conditions that allow garbage collection

The system has low physical memory

In case of an acceptable threshold

When the GC method has been called

**19. Discuss CTS types in .NET Core.**

Common Type System or CTS standard defines and explains how to use data types in the .NET framework. The "System.Object" is the base type that derives other types in the singly rooted object hierarchy. It is a collection of data types, and Runtime uses it to implement cross-language integration.

You can categorise this into two types:

Value types: This data type uses an object's actual value to represent any object. If you assign instance of value type to a variable, that variable is given a fresh copy of the value.

Examples: Built-in value types, User-defined value types, Enumeration, Structure

Reference types: This data type uses a reference to the object's value to represent the objects. You can say it follows the concept of pointers. It doesn't create any copy if you assign a reference type to a variable that further points to original values.

**20. Explain CoreRT.**

In .NET Core, CoreRT has been used as a native toolchain that performs compilation to translation. In other words, it compiles CIL byte code to machine code. The CoreRT uses ahead-of-complier, RyuJIT for compilation. You can also use it with other compilers to perform native compilation for UWP apps.

As a developer, you can utilise its following benefits:

It is easy to work with one single file generated during compilation along with app, CoreRT, and managed dependencies.

It works fast because of the prior execution of compiled code. You don't need to generate machine code or load the JIT compiler at runtime.

Since it uses an optimised compiler, thus it generates faster output from higher quality code.

**21. Why is Startup Class important?**

The Startup is a critical class in the application. The following points make it imperative:

It describes the pipeline of the web applications.

You can use individual startups for each environment.

It helps to perform the registration of all required middleware components.

Reading and checking thousands of lines in different environments is tough, but you can use various startup classes to resolve it.

**22. What is the best way to manage errors in .NET Core?**

There are mainly four ways to manage errors in .NET Core for web APIs.

1. **Developer Exception Page**
2. **Exception Handler Page**
3. **Exception Handle Lambda**
4. **UseStatusCodePages**

But, in all these four, the best way is "Developer Exception Page" as it provides detailed information (stacks, query string parameters, headers, cookies) about unhandled request exceptions. You can easily enable this page by running your applications in the development environment. This page runs early in the middleware pipeline, so you can easily catch the exception in middleware.

**23. IS MEF still available in .NET Core?**

Yes, MEF or Managed Extensibility Framework is still available. This library plays a major role in developing lightweight and extensible applications. You can easily use extensions without configuration. You can restore the extensions within and outside the application. You can smoothly perform code encapsulation and prevent fragile complex dependencies.

It has been considered outdated but is still available. If you want to use it, you must use it using some plugins systems and namespaces like "System.Composition", "System.ComponnetModel.Composition", and "Microsoft.Composition".

**24. What is response caching in .NET Core?**

During response caching, cache-related headers are mentioned in the HTTP responses of .NET Core MVC actions. Using these headers, we can specify how the client/proxy machine will cache responses to requests. This, in turn, reduces the number of client/proxy requests to the web server because the responses are sent from the cache itself.

As we can see in the below diagram, the first request has a complete cycle from client browser to proxy server and then subsequently to web server. Now, the proxy server has stored the response in the cache. For all the subsequent requests, the proxy server sends the response from the cache itself. Hence, the number of proxy/client requests to the web server is reduced.

**25. What is a generic host in .NET Core?**

The generic host was previously present as ‘Web Host’, in .NET Core for web applications. Later, the ‘Web Host’ was deprecated and a generic host was introduced to cater to the web, Windows, Linux, and console applications.

Whenever a new application is started we are required to take care of the below points:

Dependency Injection

Configuration

Logging

Service lifetime management

.NET generic host called ‘HostBuilder’ helps us to manage all the above tasks since it is built on the original abstraction of these tools.

**26. What is routing in .NET Core?**

It is a process through which the incoming requests are mapped to the corresponding controllers and actions. The .NET Core MVC has a routing middleware to perform this task. This middleware matches the incoming HTTP requests to the executable request-handling code. We can define the routing in the middleware pipeline in the ‘Startup.Configure’ file.

As we can see in the below code snippet, there are two methods or pair of middleware to define routing:

UseRouting: Adds route which matches the middleware pipeline.

UseEndpoints: Adds end execution point to the middleware pipeline and runs the delegate of the endpoint.

**27. What is Dependency Injection in .NET Core? Explain its advantages.**

.NET Core has been designed to support Dependency Injection(DI), which means the application is loosely coupled. It is a technique to introduce Inversion Control(IoC) between the classes and their dependencies. In other words, the object maintains only that dependency which is required during that particular task. A dependency is an object on which another object depends, by dependency injection, the application becomes better testable, maintainable, and reusable.

Dependency Injection has three steps:

An interface or base class is present to provide an abstraction for dependency implementation.

Dependency is registered in a service container, a built-in container IServiceProvider is present in .NET Core.

Service is injected into the constructor of the class where dependency is used.

Advantages of Dependency Injection:

Code is flexible, implementation can be changed without much overhead.

Code becomes easy to test because of the use of interfaces.

Code is loosely coupled, clean, and easy to maintain.

**28. What role does IIS manager play for ASP.NET MVC?**

The application deployment process requires a windows server with an installed IIS manager. You need to use the IIS manager to perform deployment after the development of the applications. Without deployment, you can't bring any application to the market; thus, the IIS manager plays a primary role in completing this process. Click this link to know all steps of deployment using IIS manager.

Another deployment option is to use the Docker environment, which first deploys the docker package on any server machine and then implements the next deployment stage.

**29. Discuss role-based authentication in ASP.NET MVC?**

Roles define the permission to access something. A user can access any resource if they have permission. Role-based authentication is essential to ensure the security of applications and their data. It defines the role of providers and membership. The main task of providers is to give permission and assign roles to users to ensure authentication and authorization.

**30. How would you differentiate ASP.NET from ASP.NET MVC?**

Check the following points to understand how ASP.NET is different from ASP.NET MVC:

ASP.NET is a web platform, whereas ASP.NET MVC is an application framework for building web applications.

ASP.NET offers a layer that resides on the web server's top layer for creating web applications and services. Conversely, ASP.NET MVC framework stays on top of ASP.NET to design web applications with the help of ASP.NET's APIs.

ASP.NET is based on a simple event-driven programming model, whereas ASP.NET MVC is based on the "Model-View-Controller" architectural model.

**31. Which feature of ASP.NET Core MVC has been used as a new way of exposing server-side code that renders HTML elements?**

The new "Tag helper" feature of ASP.NET Core MVC helps expose server-side code that renders HTML elements. It brings the same features of "HTML Razor helpers, " which looks like standard HTML elements. There is no need to switch context between HTML and Razor Syntax. Tag helpers are objects, and you can bound them to the models and dynamically render HTML elements according to their properties. Some of the common Tag-helper objects are as follows:

asp-action – To use action methods

asp-for – To use model binding

asp-route-id – To use route expression

asp-validation-summary – For validations

If you are a front-end designer working on CSS, JS frameworks or libraries, this feature can help you to quickly change or update the "View" without knowing the programming language. Additionally, they are reliable and reusable, which could be used in multiple views.

**32. What is the view component feature?**

View Component is another new feature that has been considered a powerful version of partial views. It is used for solving many problems. The primary function of this feature is to split the complex views into reusable parts. With the help of partial views, you can also access the parent page's view model.

But, one drawback of this feature is that it can't access the page model and can operate on the passed arguments. Thus, the best application of this feature is to use it to render reusable pieces of pages that might consist of logic. Use this feature through dependency injection, which makes it robust and reusable.

**33. What do you mean by MVC application life cycle?**

MVC application life cycle has two stages of executing applications.

First stage: Creating the request object

Fill the route table using route collection

Fetch the route to create "RouteData" object

Now use this object to create "RequestContext" object

At last, create a controller instance to control the class instance

Second stage: Creating the response object

Execute the action

Send the response to the browser as a result

**34. What are the different return types used by the "controller action" method in MVC**

In ASP.NET MVC, controllers, controller actions, and action results are linked. You can consider that the action is a method on the controller which is called whenever someone requests URL in the browser address bar. The controller responds to the requests and also exposes controller actions. In simple, this action returns action results in different return types. Check this following table to know these return types, which inherit from the base Action Result class.

Return Type Meaning

ViewResult It represents HTML and markup

EmptyResult It represents no result

RedirectResult It represents a redirection to a new URL

JsonResult It represents a JavaScript Object Notation result which could be used in an AJAX application

JavaScriptResult It represents a JavaScript script

ContentResult It represents a text result

FileContentResult It represents a downloadable file (with the binary content)

FilePathResult It represents a downloadable file (with a path)

FileStreamResult It represents a downloadable file (with a file stream)

**35. What is Scaffolding in ASP.NET MVC?**

One of the essential concepts of ASP.NET MVC that help developers like me generate code to perform basic operations – Create, Read, Update, Delete. You can make changes in the codes as per needs. That's why, we call it a "code-generation framework" for developing MVC applications. It helps enhance the code which interacts with the data model of applications. It also supports reducing the development time to execute data operations.

Additionally, the framework includes multiple templates, including page templates, field templates, , entity page templates, and filter templates. You can call them Scaffold templates. These templates permit you to design and build a fully functional website.

**36. What is the role of Action Filters?**

The central role of Action Filters in ASP.NET MVC is the execution of filtering logic after an action method is called. You can call these filters to "custom attributes" which helps to clarify declarations of pre-action or post-action behaviour to the controller's action methods. These attributes are derived from the "System.Attribute" which could be attached to classes, methods, fields, or properties. You can utilise any of these filters to implement filtering.

Filter type Function

OutputCache It caches the output of a controller action for a specific period

HandleError It handles errors raised when a controller action executes

Authorise It enables you to restrict access to a particular user or role

**37. How to intercept exceptions using ASP.NET MVC?**

An intercepting exception is an essential part of application development and execution. The exception handling's job is to respond to exceptional conditions. ASP.NET MVC has various ways to intercept exceptions, including

HandleError attribute on controllers and action method – A simple method to handle errors and exception

Try-catch-finally – A simple three blocks to catch the exception

Overriding OnException Method – A void method that takes an argument as an object of ExceptionContext to manage exception

Setting a goal exception handling filter – You have to take care of HandleErrorAttribute and need to add it RegisterGlobalFilters

Extending HandleErrorAttribute – It permits you to create your Exception Handler to manage the errors

**38. What is ASP.NET MVC? Explain its components.**

It is a lightweight and open-source web development framework, which is used to decouple data(Model), interface (View), and logic(Controller). It provides a pattern-based way to create dynamic websites and supports TDD-based development.

An MVC(Model-View-Controller) architectural pattern separates the application into three components and provides separation of concerns.

Model: Represents the state of application/logic, where the business logic and implementation logic is encapsulated.

View: It is responsible for providing the view through the user interface.

Controller: Handles user interaction, works in tandem with model and view components.

**39. What are the advantages of ASP.NET MVC?**

Provides full control over HTML rendering.

Provides separation of concerns(SoC).

Reduction of complexity by dividing the application into three components.

Supports test-driven development(TDD).

Easy integration with JavaScript, JSON, jQuery, etc.

Uses the latest technology and supports the latest trends.

**40. Why use an area in ASP.NET MVC?**

Any large ASP.NET MVC project has many controllers, views, and model classes. With time, it will become very difficult to manage it using the default MVC project structure.

The area is used to physically partition the large application into small functional units. Each unit has its own MVC folder structure and model, view, and controller folders.

**41. What is the difference between ViewData and ViewBag in ASP.NET MVC?**

ViewData and ViewBag in ASP.NET MVC are used for transferring data from controller to view.

ViewData:-

It is a dictionary object of the ‘ViewDataDictionary’ class having key-value.

Faster than ViewBag.

Type conversion code is required.

ViewBag-

It is a wrapper around ViewData and is a dynamic property.

Slower than ViewData.

Dynamic hence type conversion code is not required.

Describe the request flow in the ASP.NET MVC framework.

The request flow has below stages in the MVC framework:

Routing: It is the first step which matches the pattern of the request’s URL against the URL present in the route table.

MvcHandler: It starts the processing of the request using the ProcessRequest method.

Controller: Uses ‘IControllerFactory’ instance and calls the ‘Execute’ method, where ‘IControllerFactory’ is a default controller factory or a custom factory can be defined.

Action execution: After controller instantiation, ‘ActionInvoker’ defines which action to be performed on the controller.

View result: The ‘action’ method prepares the response and then returns a result.

View engine: ‘IViewInterface’ of the view engine selects a view engine to render the result.

View: ‘ViewResult’ returns and renders an HTML page on the browser.

**42. What are Metapackages?**

The framework .NET Core 2.0 introduced Metapackage which 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 provided by ASP.NET core.

In other words, the Metapackages of .NET Core describes the set of packages that are used together and acts as a parent of the child grouping structure. The Metapackages are referenced just like any other NuGet package naming convention such as "NETStandard.Library". An by referencing the meta-package, you have, then all its child packages will be having the reference of its dependent packages accordingly.

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

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

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

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

global.json

launchsettings.json

appsettings.json

bundleconfig.json

bower.json

package.json

**45. What is tag helper in ASP.NET Core?**

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 a C# class that is used to generate the view by adding the HTML element.

**46. How to disable Tag Helper at the element level?**

We can disable Tag Helper at the element level using the opt-out character ("!"). This character must apply to open and close the Html tag.

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

**47. What’s the difference between synchronous and asynchronous programming in ASP.NET Core?**

Synchronous programming in ASP.NET Core blocks the execution of source code until a task is completed. In contrast, asynchronous programming allows the execution of code to continue while a task is being processed in the background.

Asynchronous programming is useful for long-running operations that would otherwise block the application's main thread, such as reading from a file or making a network request. Asynchronous programming is typically achieved using the async and await keywords in C#. The async keyword defines an asynchronous method, which can be called by other code and will run in the background. The await keyword indicates that the calling code should wait for the asynchronous method to complete before continuing.

**48. How can you implement background work in an ASP.NET Core application?**

The IHostedService interface in ASP.NET Core defines a background task or service as part of the application's lifetime. It’s typically used for monitoring, logging, or data processing tasks that must run continuously, even when the application is not processing requests. Classes that implement the IHostedService interface are added to the application's service collection using dependency injection, and they are started and stopped automatically by the application's host.

The IHostedService interface defines two methods: StartAsync and StopAsync. The StartAsync method is called when the application starts and is used to start the background task or service. The StopAsync method is called when the application is stopped or restarted. It’s used to stop the background task or service, releasing acquired resources.

**49. How does ASP.NET Core handle concurrency and parallelism?**

ASP.NET Core provides several mechanisms for handling concurrency and parallelism depending on the application's specific requirements. Some common mechanisms used in ASP.NET Core applications are:

Asynchronous programming: ASP.NET Core supports asynchronous programming by using the async and awaits keywords. Asynchronous programming allows multiple tasks to be executed concurrently without blocking the main thread, improving the application's responsiveness.

Parallel programming: ASP.NET Core supports parallel programming using the Parallel class and the Task Parallel Library (TPL). Parallel programming allows multiple tasks to be executed concurrently across multiple processors, improving the application's performance.

Locking and synchronization: ASP.NET Core provides several mechanisms for locking and synchronization, including the lock keyword, the Interlocked class, and the Monitor class. These mechanisms allow multiple threads to access shared resources safely and prevent race conditions.

Concurrency control: ASP.NET Core supports concurrency control through transactional memory and the optimistic concurrency control (OCC) pattern. Concurrency control ensures that multiple threads can access and modify shared resources without interfering with each other.

**50. How do you implement caching in ASP.NET Core?**

Response caching in ASP.NET Core is a technique used to improve the performance and scalability of web applications by caching the ASP.NET Core MVC responses returned by the server for a specific period. Caching the response can help reduce the number of requests made to the server, as clients can reuse the cached response instead of requesting the same resource again.

Response caching works by adding a caching layer between the client and the server. When a client requests a resource, the caching layer checks whether the response for the request has been cached. If the response is cached, the caching layer returns the cached response to the client. If the response is not cached, the request is forwarded to the server, and the server generates the response and caches it for future use.

**51. How do you handle errors in .NET Core?**

.NET Core provides a built-in middleware for handling errors called the Exception Handling Middleware. Developers can also write custom middleware to handle specific types of errors.

**52. What is a namespace in .NET?**

In .NET, a namespace is a way to organize and group related classes, structures, interfaces, enumerations, and other types within a common naming context. It provides a way to avoid naming conflicts between types and helps in better organization of the code.

**53. What is an assembly in .NET?**

In .NET, an assembly is a unit of deployment that contains one or more files, such as executable files, DLLs, or resource files, that form a logical unit of functionality. It is the smallest unit of deployment and versioning in .NET, and can be shared among multiple applications.

**54. What are the different types of hosting models supported in .NET Core?**

.NET Core supports several different hosting models, which allow developers to deploy and run their applications in a variety of environments. The following are the different types of hosting models supported in .NET Core:

In-process hosting: In-process hosting allows an appliWhat is NET Core features?cation to be hosted within the same process as the hosting application. This is the default hosting model used by ASP.NET Core applications. Out-of-process hosting: Out-of-process hosting allows an application to be hosted in a separate process from the hosting application.

**55. How do you implement middleware in .NET Core?**

Steps to implement middleware in .NET Core are:

Create a middleware class: Create a new class that implements the IMiddleware interface or the Microsoft.AspNetCore.Http.IMiddleware interface.

Implement the middleware logic: In the middleware class, implement the logic for handling requests and responses. This could involve adding headers, modifying the response, or performing some other task.

Register the middleware: In the Configure method of the Startup class, use the UseMiddleware extension method to register the middleware.

**56. Explain the concept of Razor Pages in .NET Core.**

Razor Pages is a way of building web applications in .NET Core that makes it easier for developers to create web pages.

With Razor Pages, developers can create a page that combines the HTML markup and the C# code needed to generate the page in a single file. This approach simplifies the programming model by removing the need for a separate controller and view. Instead, each Razor Page has its own page model that handles the business logic for that page. Razor Pages are built on top of the existing Razor view engine that is used for rendering HTML, which makes it easier for developers who are already familiar with Razor syntax to work with them.

**57. How does .NET Core support cross-platform development?**

.NET Core is designed from the ground up to support cross-platform development, which means that developers can write code that can run on multiple platforms, including Windows, macOS, and Linux. This is achieved through several key features of the .NET Core platform:

Cross-platform runtime: .NET Core includes a runtime that can be installed on multiple platforms, allowing .NET Core applications to run on any platform that supports the runtime.

Cross-platform tooling: .NET Core includes a set of command-line tools that can be used on multiple platforms, including the .NET Core CLI and Visual Studio Code.

**58. How do you configure logging in .NET Core?**

In .NET Core, logging can be configured using the built-in logging framework, which provides a simple and extensible way to log messages from your application. The following steps outline how to configure logging in .NET Core:

Add the required logging provider NuGet packages to your project.

These packages include Microsoft.Extensions.Logging.Console, Microsoft.Extensions.Logging.Debug, Microsoft.Extensions.Logging.EventLog, Microsoft.Extensions.Logging.Abstractions, and others, depending on your needs.

In your application's Program.cs file, create a HostBuilder instance and configure logging by calling the ConfigureLogging() method on the HostBuilder object.

**59. What are the different parts of an Assembly?**

In .NET Core, an assembly is the primary unit of deployment, and it contains one or more files that make up a logical unit of functionality.

The different parts of an assembly in .NET Core include:

Manifest: This is a metadata that describes the assembly, including its version number, strong name, dependencies, and security permissions. The manifest is stored in the assembly's main file, which has the extension .dll or .exe.

MSIL code: This is the actual executable code of the assembly, which is written in Common Intermediate Language (CIL) and is stored in one or more files with the extension .dll or .exe. The MSIL code is compiled by the Just-In-Time (JIT) compiler into native code that can be executed by the CPU.

Type metadata: This is information about the types defined in the assembly, including their names, properties, methods, and fields. Type metadata is used by the CLR to load and execute types at runtime.

Resources: These are data files, such as images, icons, and configuration files, that are embedded in the assembly and can be accessed at runtime using the System.Reflection.Assembly class.

**60. Is Garbage collection an ongoing process? When does it occur?**

Yes, Garbage collection is an ongoing process that occurs in any of the following situations:

Low physical memory

When memory space used by allocated objects surpasses an acceptable threshold

When the GC. Collect method is called, however, this only happens in rare testing situations since Garbage collection in itself is an ongoing process.

**61. What is the key difference between Runtime and SDK in .NET Core?**

The major difference between .NET Core SDK and .NET Core Runtime is that the former is functional in the building/ development of applications, while the latter is a virtual machine responsible for the application execution and running.

**62. What is .NET Core SDK?**

The set of tools and libraries used in the development process is referred to as SDK. They help developers in creating .NET Core libraries and applications.

**63. What is .NET Standard?**

The .NET standard is a set of APIs meant to establish unity in the .NET ecosystem. It does this by compiling the base class libraries for different frameworks/ platforms. In simple terms, it creates a single uniform layer and set of APIs which support the whole of the .NET ecosystem.

Mention the main architectural components of .NET Core

There are 3 main architectural components:

.NET Core Runtime- The main purpose is to ensure the smooth functioning of the app/ program by providing type safety, native interop services, assemblies, garbage collection, etc.

Framework Libraries- These include base libraries with components like app composition types and other fundamental utilities.

SDK compilers (Roslyn) and command line tools- These facilitate quick development of programs/ applications on .NET Core.

**64. What are Empty migrations?**

When you want to add migrations without making any changes in the model it might lead to the creation of code files with empty classes. These can be customized to perform operations not related to the core Entity Framework (EF) model.

**65. What is the IGCToCLR interface?**

The IGCToCLR interface is used to communicate with the runtime environment. It passes an argument to the InitializeGarbageCollector function.

**66. What is the purpose of webHostBuilder()**

The webHostBuilder() as the name suggests is a factory used to create a web host for a web application. This function also configures the bits needed by the web host to run the application. It is a part of Microsoft.AspNet.Hosting namespace.

**67. Why would you generate SQL scripts in .Net core?**

When you want to add migrations, you will have to deploy and apply them to the database to bring it to action. This is where you need to generate SQL scripts that ensure accuracy in the application of the migrations and consequent schema changes of the databases.

**68. How do you decide when to use .NET Standard Class Library as against .NET Core Library**

When you use the .NET Standard library type you will reduce the .NET surface area and at the same time increase the number of apps that are compatible with your library.

You should use the .NET Core library type when the motive is to increase the .NET API surface area. But remember this also limits the compatibility of the library to only .NET Core applications.

**69. Explain the difference between Task and Thread in .NET**

In simple terms, a task is something you want to get done, and a thread is a way to accomplish it. So in this sense, a thread is a part of the task.

A thread represents the smallest unit of code processing at the OS level, with stacks and kernels. You can exercise a high degree of control over threads with Suspend() or Abort() or Resume() a thread. ThreadPool, as the name suggests is a wrapper comprising a pool of threads maintained in the runtime environment by CLR.

A task in comparison is executed by a TaskScheduler and cannot create its own OS threads. A default scheduler runs on the ThreadPool and also lets you know when the task finishes and returns a result.

**70. What's the difference between RyuJIT and Roslyn?**

Roslyn is a .NET Core compiler that compiles VB or C# code to the intermediate language (IL). Whereas, RyuJIT as the name suggests is a Just-In-Time compiler that works the other way around i.e. compiles the IL to native code.

**71. What is the hosting environment?**

The hosting environment contains application-specific details such as application functions, where it is stored, services for app management, etc. A HostingEnvironment is created in the ApplicationDomain, before the ASP's creation. In other words, a hosting environment is responsible for application management and app-specific functions.

This feature of .NET Core makes it possible for the developer to work with multiple environments, without creating any friction.

**72. What is JIT and how many types of JIT compilations do you know?**

Programs called compilers to convert all source code into executable code before it is brought to action. This is called compilation, which is when compilers convert application code into machine instructions (also object code). There are two steps to the whole process- first where the source code is converted to IL by language-specific compilers, and the second is converting IL to machine instructions by JIT compilers. These are called JIT compilers because only the executed IL code fragments are compiled to machine instructions, that too at the runtime, or should we say just-in-time for execution.

The .NET has three types of JIT compilers, namely,

Pre-JIT Compilers

Econo JIT Compilers

Normal JIT Compilers

**73. Is the 'debug' class the same as the 'trace' class?**

They aren't the same thing. Debugging refers to the process of finding errors in a code. While tracing refers to the process of charting out the execution pathway of a code, or getting other execution information. The trace class is broader in comparison as it can be used for both releasing builds as well as debugging the code.

**74. What is MEF in .NET Core?**

MEF stands for Managed Extensibility Framework and it is a library used for developing lightweight extensible applications. With the MEF, developers can discover extensions and use them without the need for configuration. Many believe that MEF is no longer available in .NET Core, however, it has been ported to the .NET Core as the System.Composition.

**75. What are UWP Apps in .Net Core?**

UWP stands for Universal Windows Platform and it is one of the numerous ways of client application creation for Windows. The WinRT APIs used in these apps result in powerful UIs and advanced asynchronous features. The common features of these apps are that they are secure, use a common API, and can install and uninstall the apps without risking 'machine rot'.

**76. What is MSBuild?**

Microsoft Build Engine (MSBuild) is a VS and Microsoft open-source platform for building applications. It helps in the automation of the software creation process including code compilation, testing, packaging, documentation, and deployment.

**77. What is CoreRT?**

The CoreRTis the .NET Core toolset that is responsible for compilation to translation processes. It uses the ahead-of-compiler RyuJIT to compile CIL byte core to machine code.

**78. Explain response caching**

Response caching is when the .NET Core MVC's HTTP responses are pre-specified in cache-related headers. These headers describe how intermediate or client machines should cache responses to requests. This hence reduces the volume of requests the client or proxy machine makes to the web server.

**79. What is the purpose of the wwwroot folder?**

The wwwroot folder contains static files and compiled assets, such as JavaScript, CSS, and images that your web application needs. Wwwroot is the only folder in the entire project that's exposed as-is to the browser.

**80. What is the purpose of the appsettings.json file?**

Appsettings.json contains all of the application's settings, which allow you to configure your application behavior.

**81. What is IIS?**

IIS stands for Internet Information Services. It is a powerful web server developed by Microsoft. IIS can also act as a load balancer to distribute incoming HTTP requests to different application servers to allow high reliability and scalability. It can also act as a reverse proxy, i.e. accept a client’s request, forward it to an application server, and return the client’s response. A reverse proxy improves the security, reliability, and performance of your application.

A limitation of IIS is that it only runs on Windows. However, it is very configurable. You can configure it to suit your application’s specific needs.

**82. What is a cookie?**

A cookie is a small amount of data that is persisted across requests and even sessions. Cookies store information about the user. The browser stores the cookies on the user’s computer. Most browsers store the cookies as key-value pairs.

**83. Explain the difference between app.Run and app.Use in ASP.NET Core.**

app.Use method adds a middleware delegate to the application's request pipeline. When you want to pass the context to the next middleware then prefer app.Use method.

app.Run method adds a terminal middleware delegate to the application's request pipeline. When you want to terminate the pipeline then prefer to use the app.Run method.

**84. 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.

**85. 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

**86. 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.

**87. 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.

**88. 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.

**89. 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.

**90. 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.

**91. 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.

**92. 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.

**93. 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.

**94. How is Mono different from .NET Core?**

.NET Core is a part of the .NET framework developed and maintained by Microsoft to facilitate the development of modern applications and other developer workflows. Mono is also an extension of the .NET framework, but this has been optimized by Xamarin, for Android, Windows, iOS, and macOS. The base library for .NET Core is CoreFX Class Library, and for Mono is Mono Class Library.

**95. What is CoreCLR?**

The CLR is the execution/ run-time engine in .NET Core, that is, it is responsible for the execution and running of programs in different programming languages. It consists of the following major components:

Garbage collector

JIT compiler

Low-level classes

Primitive data types.

CLS and CTS

**96. What is Kestrel?**

Kestrel is a cross-platform web server that is enabled by default in ASP.NET Core project templates. It supports- HTTPS, HTTP/2, opaque upgrade (to enable WebSockets), and Unix sockets. It is supported on all versions and platforms of .NET Core.

**97. What is the difference between .Net Core and Mono?**

* Mono is third party implementation of .Net Framework for Linux/Android/iOs
* .Net Core is Microsoft's own implementation for same.

#### **98. How do you handle configuration in a .NET Core application?**

In .NET Core, the built-in configuration system allows you to read configuration data from various sources such as JSON files, environment variables, command-line arguments, and more. The configuration system is based on the Microsoft.Extensions.Configuration package, which provides a flexible and extensible way to handle configuration data in your application.

#### **99. Can you explain the concept of Identity in .NET Core?**

Identity is a feature of ASP.NET Core that provides a way to handle authentication and authorization for web applications. It allows you to easily add support for user registration, login, and password management to your application.

In .NET Core, identity is implemented using the Identity library, which provides a set of classes and interfaces for handling user identities and roles.

#### **100. Can you explain the concept of SignalR in .NET Core?**

SignalR is a real-time communication library for ASP.NET Core that allows you to easily add real-time functionality to your web applications. It enables bi-directional communication between a client and a server, allowing the server to push updates to the client in real-time.

SignalR uses a variety of underlying technologies to provide real-time communication, including WebSockets, Server-Sent Events, and Long Polling. It automatically chooses the best transport based on the client's capabilities and the network conditions.

#### **101. How do you implement background tasks in a .NET Core application?**

Background tasks are a way to run long-running or scheduled operations in a .NET Core application without blocking the main thread. This allows the application to continue processing requests while the background task is running.

In .NET Core, background tasks can be implemented using the IHostedService interface, which allows you to create a service that runs in the background when the application starts.

#### **102. Can you explain the concept of Blazor in .NET Core?**

Blazor is a web framework for building client-side web applications using C# and .NET. It allows developers to use C# for both client-side and server-side logic, eliminating the need to use JavaScript for client-side logic.

Blazor provides two different hosting models:

1. Blazor WebAssembly: This model allows you to run C# code directly in the browser using WebAssembly. The application is downloaded to the client and runs in the browser with no need for a .NET runtime on the server.
2. Blazor Server: This model runs the application on the server and uses SignalR to handle real-time communication between the client and the server. The UI updates and events are handled by the server and sent back to the client over a connection.

Blazor uses a component-based approach to building web applications, similar to React or Angular. A component is a reusable piece of UI that can accept parameters and handle events. Components can be nested to create complex UIs and can also be shared across different pages of the application.