Q1.What is the ASP.NET Core?

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

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

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

1]Built-in supports for Dependency Injection

2]Built-in supports for the logging framework and it can be extensible

3]Introduced a new, fast and cross-platform web server - Kestrel. So, a web application can run without IIS, Apache, and Nginx.

4]Multiple hosting ways are supported

5]It supports modularity, so the developer needs to include the module required by the application. However, the .NET Core framework is also providing the meta package that includes the libraries

6]Command-line supports to creating, building, and running of the application

7]There is no web.config file. We can store the custom configuration into an appsettings.json file

8]There is no Global.asax file. We can now register and use the services in the startup class

9]It has good support for asynchronous programming

10]Support WebSocket and SignalR

11]Provide protection against CSRF (Cross-Site Request Forgery)

Q3]What are the advantages of ASP.NET Core over ASP.NET?

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

1]It is cross-platform, so it can be run on Windows, Linux, and Mac.

2]There is no dependency on framework installation because all the required dependencies are shipped with our application

3]ASP.NET Core can handle more requests than the ASP.NET

4]Multiple deployment options available withASP.NET Core

Q4]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.

Q5]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.

Q6]What is the startup class in ASP.NET core?

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

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

```
{
CreateWebHostBuilder(args).Build().Run();
}

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

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

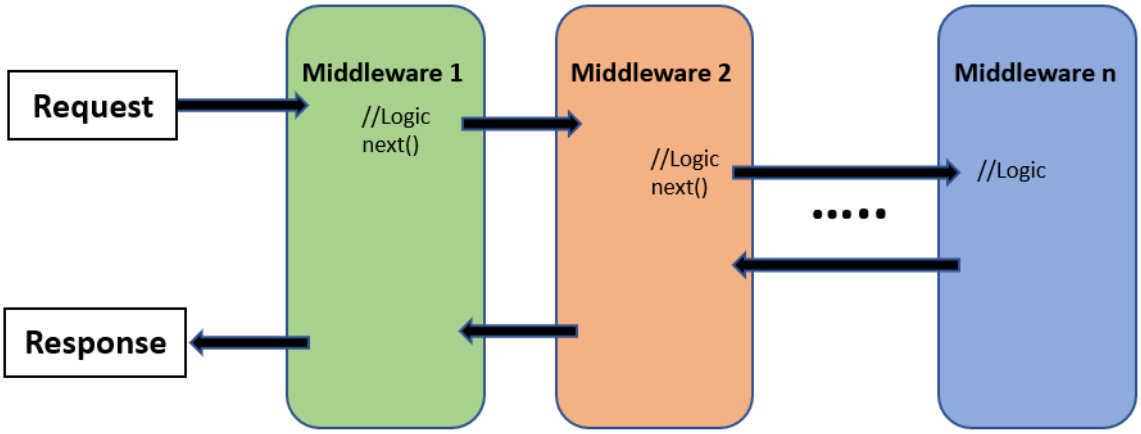
This is an optional method of startup class. It can be used to configure the services that are used by the application. This method calls first when the application is requested for the first time. Using this method, we can add the services to the DI container, so services are available as a dependency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

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

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

Q9]What is middleware?

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



While working with the ASP.NET Core framework, there are tons of built-in Middleware components available that are already made available that we can use directly that act as a plug and play components. If we don't want to use any of the in-built middleware, then we can also create our own Middleware components in asp.net core applications whenever we want. The most important point that you need to keep in mind is, that in ASP.NET Core a given Middleware component should only have a specific purpose which means it should be used for a single responsibility.

Q10]What is the difference between IApplicationBuilder.Use() and IApplicationBuilder.Run()?

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

Q11]What is the use of the "Map" extension while adding middleware to the ASP.NET Core pipeline?

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

```
public void Configure(IApplicationBuilder app)

{

app.Map("/path1", Middleware1);

app.Map("/path2", Middleware2);

}
```

Q12]What is routing in ASP.NET Core?

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

1]The conventional routing

2]Attribute routing

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

Q13]How to enable Session in ASP.NET Core?

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

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

....

....

app.UseSession();

....

....

}

}

Q14]What are the various JSON files available in ASP.NET Core?

global.json

1]launchsettings.json

2]appsettings.json

3]bundleconfig.json

4]bower.json

5]package.json

Q15]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. The functionality of the tag helper is very similar to the HTML helper of ASP.NET MVC.

Example://HTML Helper

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

//content with tag helper

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

//Equivalent HTML

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

Q16]How to disable Tag Helper at the element level?

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

Q17]What are Razor Pages in ASP.NET Core?

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

Example

@page

<h1> Hello, Book Reader!</h1>

<h2> This is Razor Pages </h2>

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

Q18]How can we do the automatic model binding in Razor pages?

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

Example

public class Test1Model : PageModel

{

[BindProperty]

public string Name { get; set; }

}

Q19]How can we inject the service dependency into the controller?

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

Step 1: Create the service

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

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

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

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

…

}

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

public class HomeController: Controller

{

IHelloWorldService _helloWorldService;

public HomeController(IHelloWorldService helloWorldService)

{

_helloWorldService = helloWorldService;

}

}

Q20]How to specify the service life for a registered service that is added as a dependency?

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

1]Singleton

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

Example

services.AddSingleton<IHelloWorldService, HelloWorldService>();

2]Transient

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

Example

services.AddTransient<IHelloWorldService, HelloWorldService>();

Scoped

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

Example

services.AddScoped<IHelloWorldService, HelloWorldService>();

Q21] What are the differences between .NET Core and .NET Framework?

.NET Core	.NET Framework
* Completely open-source.	Few components are open-source.
* Compatible with Linux, Windows,	Compatible with only Windows.
* and Mac operating systems.	
* Does not support desktop application development.	Supports web and desktop application
* Supports microservices development.	Does not support microservices development.
* Lightweight for Command Line Interface(CLI).	Heavy for Command Line Interface.

Q22]. What do you mean by state management?

Regarding .NET Core frameworks, state management is a kind of state control object to control the states of the object during different processes. Since stateless protocol, HTTP has been used, which is unable to retain user values; thus, different methods have been used to store and preserve the user data between requests.

Approach Name	**Storage Mechanism**
Cookies	HTTP Cookies, Server-side app code
Session state	HTTP Cookies, Server-side app code
Temp Data	HTTP Cookies, Session State
Query Strings	HTTP Query Strings
Hidden Fields	HTTP Form Fields

HTTPContext.Items Server-side app code

Cache Server-side app code

23] 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.

* Developer Exception Page
* Exception Handler Page
* Exception Handle Lambda
* 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.

24]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.

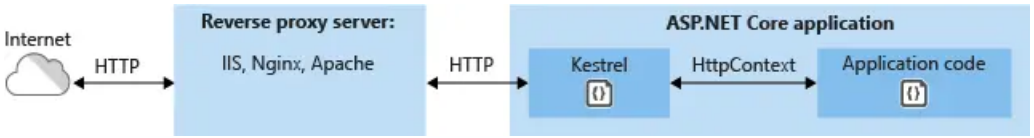
Whenever a new application is started we are required to take care of the below points:

* 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.

25]What is Kestrel?

Kestrel architecture



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