**What is Kestral C# Asp.net Core?**

Kestrel is a cross-platform web server for ASP.NET Core. It is supported on all platforms and versions that .NET Core supports.

It is included by default as internal server in ASP.NET Core. Kestrel can be used, by itself as an edge server i.e Internet-facing web server that can directly process the incoming HTTP requests from the client. In Kestrel, the process used to host the app is dotnet.exe.

Kestrel is not used with **InProcess** hosting model.

With **Out of Process** Hosting model, Kestrel can be used in one of the following 2 ways.

Kestrel can be used as the internet facing web server

Kestrel can also be used in combination with a reverse proxy server

When we run the asp.net core application using the .NET core CLI, Kestrel is the only web server that is used to handle and process the incoming HTTP request.

# What is routing in C# ASP.NET Core?

Routing is used to map requests to route handlers.

Routes are configured when the application starts up, and can extract values from the URL that will be used for request processing.

# What is the difference between IApplicationBuilder.Use() and IApplicationBuilder.Run() C# Asp.net Core?

We can configure middleware in the Configure method of the Startup class using IApplicationBuilder instance.

Run() is an extension method on IApplicationBuilder instance which adds a terminal middleware to the application's request pipeline.

The Run method is an extension method on IApplicationBuilder and accepts a parameter of RequestDelegate.

**signature of the Run method**

public static void Run(this IApplicationBuilder app, RequestDelegate handler)

**signature of RequestDelegate**

public delegate Task RequestDelegate(HttpContext context);

# What is the role of IWebHostEnvironment interface in C# ASP.NET Core?

IWebHostEnvironment Provides information about the web hosting environment an application is running in.

belongs to namespace Microsoft.AspNetCore.Hosting

The IWebHostEnvironment interface need to be injected as dependency in the Controller and then later used throughout the Controller.

The IWebHostEnvironment interface have two properties.

* **WebRootPath** − Path of the www folder(Gets or sets the absolute path to the directory that contains the web-servable application content files)
* **ContentRootPath** − Path of the root folder which contains all the Application files(Gets or sets an IFileProvider pointing at WebRootPath.)

# What are the benefits of choosing ASP.NET Core over ASP.NET?

ASP.NET Core is a high-performance, cross-platform, and open-source framework. It allows you to build modern, cloud-enabled, and Internet-connected apps.

With ASP.NET Core, you can:

* Build web applications and services, Internet of Things (IoT) apps, and backends for mobile applications.
* Work on your favorite operating system such as Windows, macOS, or Linux, and choose the tools and IDEs of your choice.
* Develop on and Deploy to the cloud or on-premises.
* Take advantage of containers and Docker to ease the deployment and distribution of your application.
* Run on the modern, fast, lightweight .NET Core framework.

Some of the significant benefits of the ASP.NET Core framework over the ASP.NET framework are:

* **High Performance**: ASP.NET Core framework is designed from scratch, keeping performance in mind. The ASP.NET team has focused on making the default web server, Kestrel, as fast as possible. TechEmpower, which has been running benchmarks on various frameworks, lists the ASP.NET Core with Kestrel as the fastest over 400 frameworks.
* **Cross-Platform**: ASP.NET Core runs on the cross-platform .NET 5.0 platform. It is not tied to a Windows operating system, like the legacy ASP.NET framework. You can develop and run production-ready ASP.NET Core apps on Linux or a Mac. If you decide to use Linux, you don't have to pay for Windows licenses, resulting in significant cost savings.
* **Open Source**: ASP.NET Core is open-source and actively developed on GitHub by thousands of developers all over the world, along with Microsoft. All the source code is hosted on GitHub for anyone to see, change and contribute back.

Along with these significant benefits, Microsoft lists the following benefits on its documentation page.

ASP.NET Core provides the following benefits:

* A unified story for building web UI and web APIs.
* Designed for testability.
* Razor Pages makes coding page-focused scenarios easier and more productive.
* Blazor lets you use C# in the browser alongside JavaScript. Share server-side and client-side app logic all written with .NET.
* Ability to develop and run on Windows, macOS, and Linux.
* Open-source and community-focused.
* Integration of modern, client-side frameworks and development workflows.
* Support for hosting Remote Procedure Call (RPC) services using gRPC.
* A cloud-ready, environment-based configuration system.
* Built-in dependency injection.
* A lightweight, high-performance, and modular HTTP request pipeline.
* Ability to host on the following:
  + Kestrel
  + IIS
  + HTTP.sys
  + Nginx
  + Apache
  + Docker
* Side-by-side versioning
* Tooling that simplifies modern web development.

Choosing ASP.NET Core lets you develop applications using new technologies such as Razor Pages and Blazor, in addition to the traditional Model-View-Controller approach.

However, it doesn't mean you have to switch from the ASP.NET framework right away. Though it's a more desirable choice in many aspects, you don't have to switch if you are happy with your current setup and don't have any pain points. This is especially true if you are maintaining a legacy ASP.NET application that is no longer actively developed. ASP.NET 4.x is a mature framework that provides the services needed to build enterprise-grade, server-based web apps on Windows.

Here are some situations where the older ASP.NET framework is still a better choice.

* You are running on Windows servers and don't need cross-platform support for your applications.
* You want a stable environment to work in that doesn't change frequently.
* Have tight deadlines and release schedules.
* You are maintaining a legacy app that's not getting any new features.

# What is the purpose of Program.cs file in C# ASP.NET Core project?

ASP.NET Core web application is actually a console project which starts executing from the entry point public static void Main() in Program class where we can create a host for the web application.

# What is ASP.NET Core? Explain how it's different from the ASP.NET framework.

ASP.NET Core is an open-source web application framework developed by Microsoft. It’s cross-platform and runs on Windows, Mac, and Linux. Though Microsoft primarily develops it, many developers worldwide contribute to it. It’s completely free to use for commercial or hobby applications, and there are no fees or licensing costs.

ASP.NET is used to build high-performance, dynamic, and scalable web applications. You can also use it to create web APIs that can be consumed by client applications, such as mobile or embedded devices, or even web applications.

ASP.NET makes building web applications easy by providing a structure around which you can build your applications. It includes helper classes and functions that do a lot of routine processing for you, which saves you from writing a lot of this code yourself. This allows you to focus on the business logic of your applications without worrying about boilerplate code that’s common to all web applications. Using ASP.NET Core, you can write web applications or web APIs faster and more secure than if you try to build everything from scratch.

A standard ASP.NET application consists of various layers. When a client such as a browser makes an HTTP request, the framework accepts that request, processes it, and then forwards the data from the request to the application code. This application code can be a controller (for an MVC app or a web API) or a Razor page (for Razor apps). These handlers then use the application’s domain logic to build the response. The framework finally sends the response to the client.

ASP.NET Core provides you libraries that handle the following responsibilities:

* Accept HTTP requests, process them, and send a response.
* Create dynamic web pages i.e., pages that display different data depending on various factors such as current users or input.
* Provide a structure for your application, easing the development and maintenance
* Serve static files such as images or other documents.
* Handle logging, caching, authentication, and authorization.

Microsoft first released the ASP.NET framework in 2002 as part of version 1.0 of the .NET framework. Since then, it has gone through many iterations and evolutions. ASP.NET Core is the latest evolution which was released in June 2016. It is fundamentally a new framework and contains significant architectural changes than the original ASP.NET framework. Microsoft rewrote the whole technology stack to enable various optimizations, speed improvements, and cross-platform support.

ASP.NET Core runs on the .NET Core platform, a lightweight, cross-platform, high-performance version of the original .NET framework. Going forward, Microsoft will be developing solely on the ASP.NET Core framework. The legacy ASP.NET framework will still receive bug fixes and security patches, but no new features will be added. Microsoft recommends that you should use the new ASP.NET Core framework for all new web development.

# How C# ASP.NET Core Middleware is different from HttpModule?

HttpModules are configured via web.config or global.asax Developer don’t have control on order of execution.

As order of modules is mainly based on application life cycle events. The execution order remains same for requests and responses.

HttpModules helps you to attach code specific to a application events. HttpModules are tied to System.web.

Middleware are configured in Startup.cs code rather than web.config file (entry point for application)

Unlike HttpModules, there is full control of what get’s executed and in what order. As they are executed in the order in which they are added.

Order of middleware for responses is the reverse from that for requests.

Middleware is independent of these events.

Middlewares are host independent.

Built in Middlewares of Asp.Net core

**Authentication**    Provides authentication support.

**CORS**                  Configures Cross-Origin Resource Sharing.

**Routing**              Define and constrain request routes.

**Session**              Provides support for managing user sessions.

**Diagnostics**       Includes support for error pages and runtime information.

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

ASP.net Core is re-architected from prior versions of ASP.net,where the configuration was relying on System.Configuration and xml configurations in web.config file. In ASP.net Core, a new easy way to declare and access the global settings for solution, project specific settings,client specific settings etc..The new configuration model, works with XML,INI and JSON files.

Different configuration json files in ASP.net Core There are mainly 6 configuration JSON files in ASP.net Core.

global.json

launchsettings.json

appsettings.json

bundleconfig.json

project.json

bower.json

# What is the use of the Configure() method of startup class in C# Asp.net Core?

The configure method is present inside startup class of ASP.NET Core application

The Configure method is a place where you can configure application request pipeline for your application using IApplicationBuilder instance that is provided by the built-in IoC container

The Configure method by default has these three parameters IApplicationBuilder, IWebHostEnvironment and ILoggerFactory .

At run time, the ConfigureServices method is called before the Configure method. This is to register custom service with the IoC container which can be used in Configure method.

IWebHostEnvironment :Provides information about the web hosting environment an application is running in.

IApplicationBuilder:Defines a class that provides the mechanisms to configure an application's request pipeline.

# Explain the purpose of the Startup class in ASP.NET Core

The Startup class configures your application's services and defines the middleware pipeline.

Generally speaking, the Program class is where you configure your application's infrastructure, such as the HTTP server, integration with IIS, and configuration sources. In contrast, the Startup class defines which components and features your application uses and the middleware pipeline for your app.

# What is the use of UseIISIntegration in C# Asp.net Core?

All ASP.NET Core applications require a WebHost object that essentially serves as the application and web server. WebHostBuilder is used to configure and create the WebHost. You will normally see UseKestrel() and UseIISIntegration() in the WebHostBuilder setup code.

## **What do these do?**

**UseKestrel()** − This registers the IServer interface for Kestrel as the server that will be used to host your application.

In the future, there could be other options, including WebListener which will be Windows only.

**UseIISIntegration()** − This tells ASP.NET that IIS will be working as a reverse proxy in front of Kestrel.

This then specifies some settings around which port Kestrel should listen on, forwarding headers, and other details.

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

Middleware are software components that are assembled into an application pipeline to handle requests and responses.

Each component chooses whether to pass the request on to the next component in the pipeline, and can perform certain actions before and after the next component is invoked in the pipeline.

Map extensions are used as convention for branching the pipeline.

The Map extension method is used to match request delegates based on a request’s path. Map simply accepts a path and a function that configures a separate middleware pipeline.

In the following example, any request with the base path of /maptest will be handled by the pipeline configured in the HandleMapTest method.

# What is Metapackage in C# Asp.net Core?

It is known that Microsoft.AspNetCore package is one of the packages added to many ASP.NET Core templates.

The Microsoft.AspNetCore package is repeatedly included as one of the usual project dependencies when opening a new ASP.NET Core project. It delivers many of the crucial packages to position up a basic ASP.NET Core application.

Though, this package does not contain any actual dlls or code itself, it merely contains a series of dependencies on additional packages. By adding this package to your project, you bring in all the relevant packages along with their dlls on which it depends and it is called a metapackage.

Specifically, the packages it lists are −

Microsoft.AspNetCore.Diagnostics

Microsoft.AspNetCore.Hosting

Microsoft.AspNetCore.Routing

Microsoft.AspNetCore.Server.IISIntegration

Microsoft.AspNetCore.Server.Kestrel

Microsoft.Extensions.Configuration.EnvironmentVariables

Microsoft.Extensions.Configuration.FileExtensions

Microsoft.Extensions.Configuration.Json

Microsoft.Extensions.Logging

Microsoft.Extensions.Logging.Console

Microsoft.Extensions.Options.ConfigurationExtensions

NETStandard.Library

The versions of these packages you will receive depends on which version of the Microsoft.AspNetCore package you install.

These dependencies deliver the primary basic libraries for setting up a basic ASP.NET Core server that uses the Kestrel web server and includes IIS Integration.

In terms of the application itself, with the help of this package alone you can load application settings and environment variables into configuration, use the IOptions interface, and configure logging to the console.

For middleware, only the Microsoft.AspNetCore.Diagnostics package is included, which would allow adding middleware such as the ExceptionHandlerMiddleware, the DeveloperExceptionPageMiddleware and the StatusCodePagesMiddleware.

To complete an application, we cannot use only Metapackage because it does not provide sufficient controls but we can use Microsoft.AspNetCore.Mvc or Microsoft.AspNetCore.MvcCore package to add MVC capabilities to our application, and also some other packages would be needed.

The metapackage just try to use a number of packages that can be applied to many applications so that we don’t need to load more dependencies but it actual does not do that because it requires other packages as well. Thus, if the number of packages is large then the dependencies increases which impacts the real use of metapackage. For example, one of the dependencies on which the Microsoft.AspNetCore depends is the NETStandard.Library package, which is also a metapackage and hence the dependencies increases.