1. What is the ASP.NET Core?

[ASP.NET Core](https://www.dotnettricks.com/training/masters-program/aspnet-core) is not an upgraded version of ASP.NET. ASP.NET Core is completely rewriting that work with .net Core framework. It is much faster, configurable, modular, scalable, extensible and 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 application, mobile application, IoT application.

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

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

* + Built-in supports for[Dependency Injection](https://www.dotnettricks.com/learn/dependencyinjection)
  + Built-in supports for the logging framework and it can be extensible
  + Introduced new, fast and cross-platform web server - Kestrel. So, a web application can run without IIS, Apache, and Nginx.
  + Multiple hosting ways are supported
  + It supports modularity, so the developer needs to include the module required by the application. However, [.NET Core framework is also providing the meta package that includes the libraries](https://www.dotnettricks.com/learn/netcore)
  + Command-line supports to create, build and run the application
  + There is no web.config file. We can store the custom configuration into an appsettings.json file
  + There is no Global.asax file. We can now register and use the services into startup class
  + It has good support for asynchronous programming
  + Support WebSocket and SignalR
  + Provide protection against CSRF (Cross-Site Request Forgery)

1. What are the advantages of ASP.NET Core over ASP.NET?

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

* + It is cross-platform, so it can be run on Windows, Linux, and Mac.
  + There is no dependency on framework installation because all the required dependencies are ship with our application
  + ASP.NET Core can handle more request than the ASP.NET
  + Multiple deployment options available withASP.NET Core

1. What is Metapackages?

The framework .NET Core 2.0 introduced Metapackage that includes all the supported package 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 provide by ASP.NET core.

1. 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.

1. What is the startup class in ASP.NET core?

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 rated items. It is not necessary that class name must "Startup", it can be anything, we can configure startup class in 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>();

}

1. What is the use of ConfigureServices method of startup class?

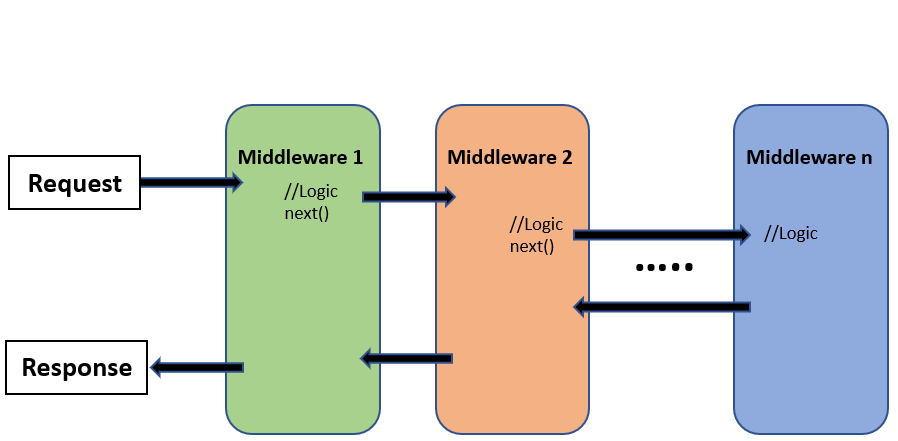
This is an optional method of startup class. It can be used to configure the services that are used by the application. This method calls first when the application is requested for the first time. Using this method, we can add the services to the DI container, so services are available as a dependency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

1. What is the use of the Configure method of 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](https://www.dotnettricks.com/learn/aspnetcore/middleware-custom-pipeline) such as routing, authentication, session, etc. as well as third-party middleware.

1. What is middleware?

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



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

We can use both the methods in Configure methods of startup class. Both are used to add middleware delegate 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.Run method never calls the subsequent ore next middleware. After IApplicationBuilder.Run method, system stop adding middleware in request pipeline.

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

It is used for branching the pipeline. It branches the ASP.NET Core pipeline based on request path matching. If 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);

}

1. 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 URL) that used to process the request. Using the route, routing can find route handler based on URL. All the routes are registered when the application is started. There are two types of [routing](https://www.dotnettricks.com/learn/aspnetcore/routing) supported by ASP.NET Core

* + The conventional routing
  + Attribute routing

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

1. 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 ASP.NET Core application, we need to add this package to csproj file and add the Session middleware to 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();

….

….

}

}

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

There are following JSON files in ASP.NET Core :

* + global.json
  + launchsettings.json
  + appsettings.json
  + bundleconfig.json
  + bower.json
  + package.json

1. What is tag helper in ASP.NET Core?

It is a feature provided by 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 C# classes that used to generate the view by adding the HTML element. The functionality of tag helper is very similar to 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">

1. How to disable Tag Helper at element level?

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

**Example**

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

1. 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 feature 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 file, but it is not really code-behind file. It is class inherited from PageModel class.

1. How can we do automatic model binding in Razor pages?

The Razor pages provide the option to bind property automatically when posted the data using 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 request.

**Example**

public class Test1Model : PageModel

{

[BindProperty]

public string Name { get; set; }

}

1. How can we inject the service dependency into the controller?

There are three easy steps to add 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 Service container (service can either 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;

}

}

1. How to specify service lifetime for register service that 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 by ASP.NET Core framework. There is three type of lifetimes.

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 AddSingleton method of IServiceCollection. ASP.NET Core creates service instance at the time of registration and subsequence request use this service instance. Here, we do not require to implement Singleton design pattern and single instance maintained by the ASP.NET Core itself.

**Example**

services.AddSingleton<IHelloWorldService, HelloWorldService>();

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 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 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, 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 singleton object.

services.AddScoped<IHelloWorldService, HelloWorldService>();