.Net Core API

1. **What is .Net Web API Core?**
2. **Why should we use .Net Core Web API?**

* Open source web framework
* Cloud optimized, lightweight and modular framework
* Can be deployed on cloud as well as on-premises
* Cross platform (Windows, Mac, Linux)
* ASP.Net Core and .Net Core are different (as like ASP.Net and .Net Framework)

.NET Core

ASP.NET Core

Core Framework

Core CLR

.Net Compiler (Roslyn)

1. **What featured does it include?**

* **Supports multiple platforms** – no need to build applications in different platform to support different platforms
* **Fast** - due to not depend on System.Web.dll for browser communication. Allows to include packages as per our need so not over-burdened so improve performance and scalability.
* **In built IoC Container** – automatic dependency injection, which makes application maintainable and testable.
* **Allows integration with all modern UI framework** – Angular JS, React JS, Bootstrap etc
* **Hosting** on any platform(IIS, Apache, Linux)
* **Smaller deployment package size,** because it is lightweight than .Net framework.

1. **Difference between .Net Core Runtime and .Net Core SDK?**

.Net Core Runtime - used to run .Net core application only

.Net Core SDK - includes tools and libraries to develop .Net Core applications

1. **wwwroot folder uses**

* treated as web root folder, all static files can be stored here
* can be accessed with relative path
* can be also served on http request directly
* all other files outside cannot be accessed and blocked
* for example, css, js, image files and other templates like email tempates

can be accessed like, <http://localhost:18810/css/filename.css>

* Can rename this folder and configure it in program.cs file like, UseWebRoot(“FolderName”)
* Include in middleware for using staticfiles on request: app.UseStaticFiles();

1. **Program.cs file**

* .Net Core application is actually a console app which starts executing from public static void main() in Program class.
* Create web host server object (IWebHost)
* Configure root directory
* Configure startup class of the application
* Build and run the application on any platform
* Also, called appSettings.json internally for configuration as like (web.config)
* Configure default centralized logging

1. **Startup.cs**

* Located in root folder
* Can work like global.asax
* It is executed when application starts
* Can also be configured Startup class in program.cs file
* Includes 2 public methods: ConfigureServices (optional) and Configure (must to have)
* ConfigureServices: register dependent classes here (works like IoC Container) once configured (as a Service) here you can use class anywhere in the application
* Configure(): used to configure request pipeline for application, can be helpful to define middleware for each request execution, also helpful for performance improvement
* At run time, the ConfigureServices method is called before the Configure method. This is so that you can register your custom service with the IoC container which you may use in the Configure method.

1. **Built in IoC Container in detail**

* This container is represented by IServiceProvider implementation supports dependency injection by default. Managed by built in container called as services
* Two types of services are provided:

**Framework Services**: services which are part of ASP.Net Core F/W like IApplicationBuilder, IHostingEnvironment, ILoggerFactory etc.

**Application Services**: custom types, classes (Created by us)

* Service lifetime: 3 kinds of lifetime supported as below

**Singleton**: will create and share single instance of the service throughout application

**Transient**: will create a new instance of the specified service type every time you ask for.

**Scoped**: will create a new instance of the specified service type once per request and will be shared in a single request

* Methods: AddSingleton(), AddTransient(), AddScoped()
* Supports Constructor injection (recommended) and method injection (for method injection, use attribute [FromServices]
* Does not support property injection
* Can also get instance manually from HttpContext:

var services = this.HttpContext.RequestServices;

1. **Where can we configure middle for every request?**

* Middleware in Configure()
* It is a component (Class), can have multiple middleware.
* Can configure HttpHandler and HttpModules here as part of request pipeline just like Global.asax (Can be executed on every request)
* We can set the order of multiple middlewares (by using Next() method)
* Several built in middlewares:
  + **Authentication**: Adds authentication support.
  + **CORS**: Configures Cross-Origin Resource Sharing.
  + **Routing**: Adds routing capabilities for MVC or web form
  + **Session**: Adds support for user session.
  + **StaticFiles**: Adds support for serving static files and directory browsing.
  + **Diagnostics**: Adds support for reporting and handling exceptions and errors.

