# Dot net Core MVC Notes with Project

# Chapter 1

## Introduction to ASP.NET Core

ASP.NET core is cross platform, high performance, open source framework for building modern, cloud enabled web applications and services.

## Features Of ASP.NET Core

- **Cross Platform** - can be hosted on windows, linux and Mac
- **Hosting -** Can be hosted on different servers such as kestrel, IIS, NGINX, docker etc
- **Open Source -** Free for use
- **Cloud Enabled -** Supports microsoft AZURE out of the box

## Parts of Asp.NET CORE

| | |
|---|---|
| **ASP.NET Core MVC**<br><br>For creating medium to complex web application | **ASP.NET Core Web API**<br><br>For creating RESTful services for all type of client applications |
| **ASP.NET Core Razor Pages**<br><br>Used for creating simple page focused web application | **ASP.NET Core Blazor**<br><br>For creating web application with C# code for both frontend and backend |

## Benefits of ASP.NET Core

| **ASP.NET Web Forms** | **ASP.NET MVC** | **ASP.NET Core** |
|---|---|---|
| Performance issue due to server events and view state | Performance issue due to some dependencies | Faster |
| 2002 | 2009 | 2016 |
| IIS | IIS | Various Options |
| Not Support Cross platform | Not Support Cross Platform | Support Cross Platform |
| Event Driven Programing Approach | MVC Pattern | MVC Pattern |
| DI will be complicated | DI will be easy | Built in support for DI |

## StartUp.cs

StartUp.cs file is the entry point of out application. Every asp.net core application asp.net core 5 and earlier have startup.cs file. This file contains two methods

- **ConfigureServices -** this file contains services that our app is using
- **Configure -** Configure method contains middleware pipelines.

After dot net core 6 startup.cs and program.cs files gets unified and whatever service registering and middleware core we need to write in program.cs file.
**WebApplication.CreateBuilder()**

- This method is used to get an instance of a WebApplicationBuilder instance
- Then builder.Service.AddSingleton<MyService>() is used to register services for dependency injection.
- builder.Build() method gives us instance of configured web application
- And further that instance is used to set up middleware pipeline, routing and further request handling logic.

### Questions for this assignment

1. What is Kestrel and what are the advantages of Kestrel in Asp.Net Core?

**Ans** : Kestrel is high performance and cross platform default server comes with .Net core.  It is listing server by command line interface
Features of Kestrel are

- It is lightweight.
- Cross platform and support all version of .Net
- Supports HTTP and HTTPS

2. What is the difference between IIS and Kestrel? Why do we need two web servers?

**Ans:**

| **Kestrel** | **IIS** |
|---|---|
| Cross Platform | Windows only |
| Opensource | Licensing required |
| Lightweight and easy configurable | Comes with rich functionality like URL rewriting |

3. What is the purpose of launchSettings.json in asp.net core?

**Ans:** launchSetting.json file contains configuration information which describes how to start ASP .Net core application. It mainly contains run time profiles, application url and environment information.

4. What is generic host or HostBuilder in .NET Core?

**Ans:** genric host also called as HostBuilder used to manage below task

- Dependancy Injection
- Server lifeycle Management
- Configuration
- Logging

Previously there was web host available for .Net Core web application later it is deprecated and introduced genric host to cater web, windows, linux and console application.

5. What is the purpose of the .csproj file?

**Ans:** Project file is the most important file in application. It tells .NET how to build (Compile) the project. It contains information about package dependency, .NET version used to create applications, any 3rd party libraries information , configuration settings, Type of application we are building (web, windows, console etc).

6. What is IIS?

**Ans:** IIS stands for Internet Information Service. It is a powerful web server developed by Microsoft which is used to host web applications. IIS we can also use as load balancer to maintain websites scalability and reliability. We can also implement reverse proxy using IIS. The only limitation of IIS is it is only available with windows. We can configure it as per requirement.

7. What is the “Startup” class in ASP.NET core prior to Asp.Net Core 6?

**Ans:** Startup.cs file is entry point of application which contains two methods Confiure() and ConfigureServices().
Configure method is used to define middleware pipeline
ConfigureService method is used to register services in IOC container
.NET 6 onwards startup.cs file unified with program.cs file.

8. What does WebApplication.CreateBuilder() do?

**Ans:** Webapplication.CreateBuilder() is a method of .Net core which is used to configure and initialize an instance of an application. We have a WebApplicationBuilder class which is used to configure service and middleware so this method gives us an instance of that class. This method does the following things.

- Configure host for applications such as configure kestrel server, configure integration with IIS or any reverse proxy server.
- Adds and configures services in DI containers; these services include logging, routing and other necessary features.
- Set up application configuration based on settings defined in appsetting.json file or any other sources.
- Configure logging mechanism like console, azure app service or custom logging provider.
- Gives instance of application builder class to configure services and middleware.

- **Chapter 2**

**Introduction to HTTP**
HTTP stands for hypertext transfer protocol which contains a certain set of rules for communication between browser and server. HTTP sends requests from browser to server and gets responses in return from server to browser.

**Response Status Code**

| | | |
|---|---|---|
| **101** | Switching Protocol | Ex switching from http to https |
| **200** | OK | Request is successful |
| **302** | Found | Indicates redirection |
| **304** | Not Modified | Indicates some files are loaded from cache memory |
| **400** | Bad Request | Indicates some required data sent to request is incorrect or missing. |
| **401** | Unauthorised | Credentials are incorrect |
| **404** | Not Found | URL does not exists |
| **500** | Internal Server Error | Any runtime error in code represents 500 |

**HTTP Request**
When we send or seek information from browser to server it is called HTTP Request.

**Query String**
Query string is syntax where you can send parameters along with the url. Part after the question mark in the below example is the query string.
Ex /dashboard?id=1

**HTTP Request Types**

| | |
|---|---|
| **GET** | Retrieve information |
| **POST** | Insert information into database |
| **PUT** | Update information |
| **PATCH** | Partially update information |
| **DELETE** | Used to delete resource from databse |

# Chapter 3

## Middleware

Middleware is code logic that is injected into the application pipeline to handle request and response. Each request goes through this pipeline and middleware decides whether to pass the request to the next middleware or not. Middleware are chained one after another and executed in the same sequence as they are added. Middleare can be a request delegate(anonymous method or lambda expression) or a class. You can define middleware in the program.cs file. Once you call builder.Build() method you will get an application builder object. This object is used to create middleware.

## app.Run()

The extension method called Run is used to execute a terminating/short-circuiting middleware that does not forward requests to the next middleware.

Ex

```
app.Run(async (HttpContext context) =>
{
   await context.Response.WriteAsync("Hello world");
});
```

## app.Use()

app.Use() extension method we can pass requests to the next middleware as well as it is also able to short circuit requests.

Ex

```
app.Use( async (HttpContext context, RequestDelegates next)=>
{
        //Some logic
        await next(context);
});
```

## Custom Middleware

Custom Middleware is a class that contains some complex logic and used to separate middleware logic from lambda expression (request delegates) to reusable class. The Middleware class needs to implement an interface called IMiddleware which contains the InvokeAsync method.

```
Class MiddlewareClassName : IMiddleware
{
        Public async Task InvokeAsync (HttpContext context, RequestDelegate nex)
        {
```

```
                    //some logic
                    Await next(context);
          }
}
```

## How to add Custom Middleware

| | |
|---|---|
| Create Class | First step is to create a class first that contains required logic that should be inherited by IMiddleware interface like below example.<br><br>namespace MyFirstApp.CustomMiddleware<br>{<br>    public class MyCustomMiddleware : IMiddleware<br>    {<br>        public async Task InvokeAsync(HttpContext context, RequestDelegate next)<br>        {<br>            await context.Response.WriteAsync("my custom middleware - start");<br><br>            await next(context);<br>            await context.Response.WriteAsync("my custom middleware - ends");<br>        }<br>    }<br>} |
| Register Service | Register newly created middleware class.<br><br>builder.Services.AddTransient&lt;MyCustomMiddleware&gt;(); |
| Use middleware | app.UseMiddleware&lt;MyCustomMiddleware&gt;(); |
| Use of Extension method | Also you can use extension method which is used to invoke middleware with single method call<br><br>public static class CustomMiddlewareExtension<br>    { |

| | |
|---|---|
| | `public static IApplicationBuilder UseMyCustomMiddleware`<br>`(this IApplicationBuilder app)`<br>`{`<br>`return app.UseMiddleware<MyCustomMiddleware>();`<br><br>`}`<br>`}`<br><br><br>`app.UseMyCustomMiddleware();` |

**Right Order of Middleware**

1. ExceptionHandler
2. HSTS(Http strict transport security)
3. HttpsRediretion
4. Static Files
5. Routing
6. CORS
7. Athentication
8. Authorization
9. Custom Middlewares
10. EndPoint

**Middleware UseWhen**

Like use extension method we have UseWhen which check if certain condition is true then it will process with some other middleware branch.