**.NET CORE** (22st Oct 2019)

**.NET Core Versions:**

Version 1.0

Version 1.1

Version 2.0

Version 2.1

Version 2.2 (LTS – Long Term Support)

Version 3.0 (Current)

**.NET Core Features:**

Light weight,

Cross platform support,

CLI support,

Support various IDEs (Notepad, VS Code, VS Studio etc.)

Support developing Console, ASP.NET (MVC, WEB API and SPA), UWP and Class Lib etc.

No Web Forms,

No WPF and WinForms

No WCF and WF

**In 3.0 version, it supports WPF**

Built-in DI system

We can replace built-in DI system with 3rd party DI system (Autofac)

Tag Helpers are used instead HTML helpers

HTML helpers are not replaced

Razor page in .NET core 2.0

Blazor in used in 3.0 version

Configuration is JSON based

appSetting.json file

.NET Core can read config parameters from:

Environment variables,

Command line args (Highest priority)

JSON files,

XML files,

INI files,

Key Vault (Azure),

KeyPerFile

Completely compatible to cloud deployment

Application Templates:

Console

ASP.NET: MVC, Web API, SPA

Class library

Standard libraries can be used with .NET Framework and .NET Core

UWP

To Create Project:

>> dotnet new <templateName> -n <projectName>

>> dotnet new console -n SampleCoreApp

To check template names >> dotnet new

To restore >> dotnet restore

To build >> dotnet build

There are two ways to run project:

>> dotnet run (from cs project location)

dotnet <.dll name> (from \bin\debug\ location)

>> dotnet SampleCoreApp.dll

To install package

>>dotnet add package <packageName>

To publish

>>dotnet publish

On Specific path need to publish then:

>>dotnet publish -o <path>

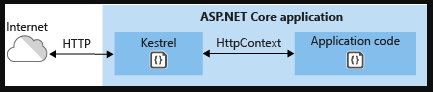
To install c# extension go to the

VS extension => search by C#

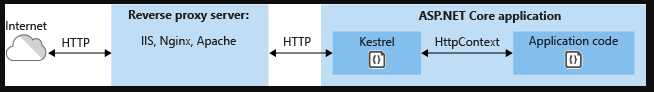
If not installed then install this one.

**ASP.NET MVC:**

.NET Core web applications uses a built-in web server called **Kestrel**.



**Kestrel** is a cross-platform light weight web server.



**Program.cs**: It contains a main method that starts the Host for MVC app.

Creates and Starts a Web Host object.

Web Host is responsible to host the application code.

Execution start from this file.

Types of Host in .NET Core:

WebHost - (ASP.NET MVC, Web API) http request comes

Generic Host –No http request pipeline (Web jobs, Thread) Background process.

**Startup.cs:** Configure the services for the application

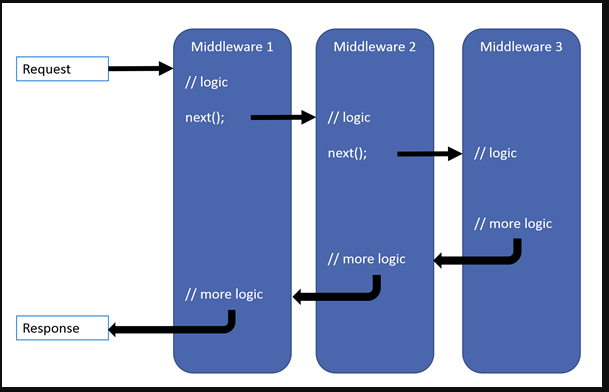
ConfigureServices() method is used to configure the services required in the app (reusable code)

Creates a request pipeline for handling requests.

Configure() method is used to configure the request pipeline.

Request pipeline defines how to handle the http request.

It contains a set of middleware functions that process the request and response.



**AppSettings.json:**

Configuration file that contains the appsettings and connection strings.

**Wwwroot:**

This folder is the default static files folder.

**.csproj file:**

Contains the framework version and package information.

**Metapackage** – “Microsoft.AspNetCore.App”

When developing MVC and API apps we need to use a bunch of dependencies.

These packages are bundled into a virtual package (metapackage)

Metapackage name is “Microsoft.AspNetCore.App”

**Controller:**

MVC controller and Web Api controller are created here.

**Model:**

Database models and view models are created here.

**Views:**

All html templates are created here.

\_ViewImports: Contains the common import statements.

**Middleware:**

Validating request   
 Filtering response

Download the sdk installer from below link:

<https://github.com/dotnet/core/blob/master/release-notes/2.2/2.2.4/2.2.4.md#downloads>

Can be cared using:

Use() method

Used to created custom middleware.

It takes a function as arg, that has two params -

Run() method

Request terminator (Short-Circuit middleware)

Map() method

Used to

MapWhen() method

UseMiddleware() method

IApplicationBuilder app => Microsoft.AspNetCore.Builder

HttpContext context => Microsoft.AspNetCore.Http

RequestDelegate \_next => Microsoft.AspNetCore.Http

**Built-in Middleware:**

UseDeveloperExceptionPage()

Used to display detailed error page while running in development.

Apply this middleware conditionally, while running in development.

UseExceptionHandler()

Redirect the error response to an error page.

It can be used to send custom error details.

UseHsts()

Enable HTTP strict transport security protocol.

UseHttpsRedirection()

Automatically redirects all http request to https.

UseCookiePolicy()

Adds one layer of security for cookie usage.

Asks the use to allow or deny cookie creation.

UseMvc() OR UseMvcWithDefaultRoute()

Executes the controller actions based on the route request.

Checks the route table and decides which controller and action need to be executed.

Routing URL templates can be defined in the middleware.

UseMvcWithDefaultRoute() applies a middleware with default route template.

UseMvc applies a routing middleware that allows custom routing template.

UseStaticFiles()

Is required to server the static files for the application.

Default static file folder is wwwroot.

This can be used multiple times in a request pipeline.

Use parameters to define static file directories and options other than wwwroot.

UseDefaultFiles()

Use before UseStaticFiles() and UseMvc() middleware.

Serve the default file when root URL is requested.

Can configure custom default documents also.

If we are creating parameterized

UseFileServer()

Is the combination of serving StaticFiles, DefaultDocuments and Directorybrowsing

**Services:**

Services are reusable codes in an application.

Services need to be registed in ConfigureServices method of Startup class.

.NET core DI system creates and inject the instance of a service wherever is required.

IoC container will take care of the object life time and scope.

Services are registered in the ServicesCollection object (services)

Service can be registered with the following scopes and life time:

Singleton (maintain the states)

A single instance of service is created and maintained till the end of application.

Services.AddSingleton() method to register a service as singleton.

Scoped

A single instance per request scope.

Transient

It create a new instance every time wherever it is required.

Short lived objects.

Build-in DI system can be replaced by third party IoC containers (eg: Autofac)

Using method injection in action:

[FromServices] IConfiguration configuration

AutoFac

Need to install below package for Autofac:



**Configuration Sources:**

Configuration means the parameters passed to the application for execution.

Such as application setting, connection string etc.

Passed as key-value pair format.

We have many configurations sources in .NET app:

**Command line Args [High priority]**

**Environment variable**

In-Memory collection

**AppSettings.json**

XML files

INI files

Key Per file

Azure key vault

AppSettings.json

ConnctinStrings:{“sqlConnection:’’, data source=localhost;integrated Security=true”

After deployment I can update the ENV variables with same key

ConnctinStrings:{“sqlConnection:’’, data source=192.12.21.54;integrated Security=true”

Dotnet “MyApp.dll” –sqlConnection “sqlConnection:’’, data source=192.12.21.54;integrated Security=true;”

When the default WebHost Builder created a Web Host, it comes with some preconfigured configuration sources such as CMD Line Args, Env Variables and AppSettings.json

var envData = configuration.GetValue<string>("VisualStudioDir");

var title = configuration.GetValue<string>("ProjectDetials:Title");

var project = configuration.GetSection("ProjectDetials");

var proTitle = project["Title"];

var description = project["Description"];

Using class map the appsetting in startup.cs file

Create a class with necessary properties Ex. AppConfiguration

services.Configure<AppConfiguration>(Configuration);

Inject this into Controller constructor

(IOptions<AppConfiguration> options)

1. Inmemory
2. AppSetting
3. Evn
4. Cmd

**ASP.NET MVC:**

Every controller inherits from the Controller class.

Every action has a return type of IActionResult.

In attribute routing the [RoutePrefix] is replaced by [Route] attribute.

Action selector (HttpGet, HttpPost) attributes can be used for defining URI for action.

**Tag Helpers:**

A tag helper is a C# class that can be used to create custom tags and attributes.

While rendering the page TagHelper will be converted in to HTML element.

Asp.net core provides a set of built-in tag helpers, but still we can create custom tag helpers.

Custom tab helper classes inherits from TagHelper class.

Tag helpers need to be registered (imported) in the \_ViewImports file using @AddTagHelper attribute.

**Caching in ASP.NET:**

Used for application performance improvement.

Instead of regenerating the data or response we can store the data and response in a storage. For a temporary period of time.

**Caching methods in ASP.NET core:**

**Response Caching** (OutPut)/(ResponseCache),

ResponseCache attribute is used for caching the action response.

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

**In-Memory Data Caching**,

Locally cache the data withing the application machine.

This is not reliable, because if app crashes the cached data will expire.

Add In memory cache service to the services collection by calling AddMemoryCache().

Inject the **IMemoryCache** in the Controller to get access to the cache object.

**Distributed Caching**

Which store the cache data outside the application.

We can we SqlServer or Redis server to cache data.

We use **IDistributedCache** interface to get access to the distributed cache service.

**Distributed cache providers:**

InMemory DistributedCache,

Only for the Development purpose, not for Production.

SqlServer DistributedCache,

>> dotnet sql-cache create "Data Source=10.2.33.35;Initial Catalog=chandan;Integrated Security=True" "dbo" "CacheTableA"

Redis DistributedCache

Package: Microsoft.Extensions.Caching.StackExchangeRedis

**State Management:**

MVC Query Strings,

Hidden Fields,

Cookies,

Session,

TempData,

Application

.NET Core:

Query Strings,

Hidden Fields,

Cookies,

Session,

Session is by default not enabled in .NECT Core.

One of the Distributed cache must be enabled to use session.

Session data is stored in distributed cache for reliability.

To use the session we need to add the session service in the service collection. (services.AddSession();)

Also we need to use the session middleware to work with session. (app.UseSession();)

Session stored data in String and Byte [] format, because of Distributed storage.

TempData,

Asp.net core provides two tempdata providers:

Cookie based tempdata provider (Default) services.AddMvc().AddCookieTempDataProvider()

Small data

Session based tempdata provider

services.AddMvc().AddSessionStateTempDataProvider()

large data

DI,

We have to use singleton service objects for state management.

Caching,

HttpContext.Items

**.NET Core App Deployment Types:**

FDD - Framework Dependent Deployment.

Dontet app.dll

Shared libraries and framework are not part of app deployment.

Runtime must be installed in machine.

Application can be run using “dotnet myapp.dll” command.

SCD – Self Contained Deployment.

Runtime is also published into the deployment folder.

No runtime required in the machine.

Specify the RID in the .csproject file

<RuntimeIdentifiers>win10-x64</RuntimeIdentifiers>

Publish using the following command-

Syntax: dotnet publish –c Release –r <rid-value>

Example: dotnet publish –c Release –r win10-x64

FDE – Framework Dependent Executables.

Shared libraries and framework are not part of app development.

Runtime must be installed in machine.

It creates a self-executable .exe files in the publish folder.

It can be run directly by command prompt or double click.

**Entity Framework Core:**

Entity classes and DbContext class.

Configure the DbContext service in ConfigureServices() method.

For production we use SqlServer as database source.

Database sources:

In-Memory

SqlServer

Microsoft.EntityframeworkCore namespace.

Commands for generate DB:

Add-Migration “InitialCreate”

Update-Database