**What is Context Class?**

The class that derives DbContext is called context class in entity framework. DbContext is an important class in Entity Framework API. It is **a bridge between your domain or entity classes and the database**.

**What is the ASP.NET Core?**

ASP.NET 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.

**What is difference between ASP.NET and ASP.Net Core?**

1. ASP.NET Core is an open source framework.
2. .NET application supports only on Windows while >NET Core is a cross platform supporting, means we can run .NET framework code only on windows while we can run .NET core framework code into windows/Linux/MacOS
3. ASP.NET uses only .NET platform to develop the applications while ASP.NET Core uses both .NET and .NET core for developing applications
4. Performance of ASP.NET Core is better
5. In ASP .NET Core there is no need to compile every time it automatically compiles.
6. .Net core has Built-in supports for Dependency Injection
7. In .Net core There is no web.config file. We can store the custom configuration into an appsettings.json file

**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.

**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 related items. It is not necessary that class name must "Startup", it can be anything, we can configure startup class in Program class.

**What is the use of ConfigureServices method of startup class/ Where would custom Dependency Injection objects be added?**

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.

**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 such as routing, authentication, session, etc. as well as third-party middleware.

**Where to keep configuration information in ASP.NET Core?**

Now there is no web.config file in ASP.NET Core so we have to store configuration information in appsettings.json file, which is JSON file keep information as key-pair in JSON format.

**What is launchsetting.json in ASP.NET Core?**

This json file holds project specific settings associated with each debug profile, Visual Studio is configured to use to launch the application, including any environment variables that should be used. You can define framework for your project for compilation and debugging for specific profiles. This file is placed in Properties folder. This file is only used on local development machine. We do not need it for publishing our asp.net core application.

**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>

**What is dependency injection?**

Dependency injection helps to achieve decoupled/loosely coupled architecture.

If I change at one place it will refplect at many more places.

**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;

}

}

**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 are three types 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 (it creates number of instances of service based on how many time it is injected to controller)

ASP.NET Core will create 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 (it creates only one instance of service no matter how many times it is injected to controller)

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.

**How to handle Error/Exception in ASP.Net core?**

To handle exceptions and display user friendly messages, we need to install Microsoft.AspNetCore.Diagnostics NuGet package and add middleware in the Configure() method.

The Microsoft.AspNetCore.Diagnostics package includes following extension methods to handle exceptions in different scenario:

1. UseDeveloperExceptionPage: The UseDeveloperExceptionPage extension method adds middleware into the request pipeline which displays developer friendly exception detail page. This helps developers in tracing errors that occur during development phase.

As this middleware displays sensitive information, it is advisable to add it only in development environment.

2. UseExceptionHandler: In MVC Core application, we might want some other controller to handle all exceptions and display custom user-friendly error messages. The UseExceptionHandler extension method allows us to configure custom error handling route. This is useful when an application runs under production environment.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment() || env.IsStaging())

{

app.UseDeveloperExceptionPage();

}

else

{

//This error page will display on Prod environment

//We can log this exception in text files using third party logging providers in try catch block.

app.UseExceptionHandler("/Home/Error");

}

}

Visual Studio automatically creates Error.cshtml under Home folder when you create ASP.NET Core project with MVC template.

2. By Using Exception Filter

For this we have to inherit IExceptionFilter Interface and we have to implement OnException method

Register the ExceptionFilter in ConfigureServices method in Startup.cs file