**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>();

}

**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.

**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.

**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.

Diagrama, Esquemático

Descripción generada automáticamente

**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.

**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);

}

**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 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.

**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();

….

….

}

}

**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

**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 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>();