<https://www.tutorialsteacher.com/core/aspnet-core-program>

In Solution Explorer, you will see the Startup.cs file. If you have worked with previous versions of ASP.NET Core, you will probably expect to see a global.asax file, which was one place where you could write codes to execute during startup of a web application.

* You would also expect to see a web.config file containing all the configuration parameters your application needed to execute.
* In ASP.NET Core those files are all gone, and instead of configuration and startup code are loaded from Startup.cs.
* There is a Startup class inside the file and in this class you can configure your application and even configure your configuration sources.

No

Global.asax file

No web.config file

In the Startup class, there are two methods where most of our work will take place. The Configure method of the class is where you build your HTTP processing pipeline.

* This defines how your application responds to requests. Currently this application can only say Hello World! and if we want the application to behave differently, we will need to change the pipeline around by adding additional code in this Configure method.
* For example, if we want to serve the static files such as an index.html file, we will need to add some code to the Configure method.
* You can also have an error page or route requests to an ASP.NET MVC controller; both of these scenarios will also require to do some work in this Configure method.
* In the Startup class, you will also see the **ConfigureServices()** method. This helps you configure components for your application.

Read Value from appSettings.json file

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

**"AllowedHosts": "\*",**

**"Message" : "Hello I am from json file"**

**}**

**---------------------**

In Startup.cs file

public class Startup

{

public IConfiguration Configuration { get; private set; }

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

-------------------------------------------------

public void Configure(IApplicationBuilder app)

{

app.UseStaticFiles();

app.Map("/newbranch", a => {

a.Map("/branch1", brancha => brancha

.Run(c => c.Response.WriteAsync("Running from the newbranch/branch1 branch!")));

a.Map("/branch2", brancha => brancha

.Run(c => c.Response.WriteAsync("Running from the newbranch/branch2 branch!")));

a.Run(c => c.Response.WriteAsync("Running from the newbranch branch!"));

});

app.Use(async (context, next) =>

{

await context.Response.WriteAsync("Hello, World!");

await context.Response.WriteAsync("<br>This is after another Hello, World!");

**await context.Response.WriteAsync(Configuration["Message"]);**

});

----------------------------------------------------

Create Index Page

app.UseDefaultFiles();

app.UseStaticFiles();

OR

app.UseFileServer();

The order in which you install the middleware is important because if you had UseDefaultFiles after UseStaticFiles, you would not get the same result.

If you are going to use UseDefaultFiles and UseStaticFiles, you might also want another piece of middleware that is inside the Microsoft.aspnet.staticfiles, NuGet package, and that is the **FileServer middleware**. This essentially includes the Default Files and the Static Files in the correct order.

Exception Handling

app.UseDeveloperExceptionPage();

app.Run((context) =>

{

throw new Exception("ERROR");

});

