**ASP.NET CORE**

**ASP.NET Core is** an open source and cloud-optimized web framework for developing modern web applications that can be developed and run on Windows, Linux and the Mac.

1. Open Source
2. Cross-Platform framework which runs on macOS, Linux and Windows.
3. A unified framework for building web UI and WEB APIs.
4. Built in Dependency Injection.
5. Light weight and modular HTTP request pipeline.

**ASP.NET CORE Doesn’t have**:

1. Global.asax
2. Web.config – this will take care by appsettings.json or custom configuration files.
3. Modules and Handlers
4. ASP.Net Page Life Cycle

**Does not depends on IIS Hosting**

Kestrel will act as an internal web server for processing request.

**There are two types of Routing in ASP.NET CORE**

1. Convention Based

Defined in the Startup.cs file. Represents all the possible routes in your system.

1. Attribute

Attribute Routing Tokens provides tokens for [area], [controller] and [action].

**Route Constraints**

Route Constraints are used to restrict the type of passed value to an action.

Restrict id with int, [HttpGet({id:int})]

**Optional Parameters**

You can define your route parameter as optional in routes by adding a question mark (?).

{id:int}

**Understanding ASP.NET Tag Helper**

Microsoft introduce three helper object – 1. HtmlHelper 2. UrlHelper 3. AjaxHelper. Tag Helpers can replace the Razor Cryptic Syntax with @. Tag Helpers actually reduce the coding amount in HTML.

**ASP.NET Core Filters**

1. Authorization Filters
2. Resource Filters
3. Action Filters
4. Exception Filters
5. Result Filters

**View Components**

It is quite similar to the Partial View in terms of reusability and reduce code repetition.

Views helps to establish a SOC design (Separation of Concerns) within the MVC application.

**ASP.NET CORE Middleware**

Middleware is a just class or component which fired on every request received in the Asp.Net Core applications. Middleware is just similar to like these HTTPHandlers and HTTPModules, which can be configured and executed in the both side. Every section of the middleware can handle either request or response or both.

**BUILT IN MIDDLEWARE**

1. Authentication
2. CORS
3. Diagnostics
4. Forwarded Headers
5. HTTP method override
6. HTTPs Redirection
7. MVC
8. Response Caching
9. Routing
10. Session
11. Static Files
12. URL Rewriting
13. Web Sockets

**Program.cs**

This is the entry point to our application.

1. BuildWebhost

A static method, which configures, build and then returns the reference to Host.

1. Kestrel

The above method tells the host to use the Kestrel web server. Kestrel is cross platform managed HTTP server based. This server allows ASP.NET CORE application ruin on OS other than windows.

1. UseContentRoot

Set the current directory as the root of the application. Load optional configuration from appsettings.json, appsettings.enviroment.json, Environment variables, and command line arguments.

1. Enable Logging

This reads the configuration rules specified in the configuration files and configures logging for console and debug output.

1. UseIISIntegration

This method integrates the Kestrel runs with IIS.

1. Build

The Built method of the WEBHOSTBUILDER builds the WEBHOST object that will host the application.

1. Run

The Run method of the WEBHOSTBUILDER start the HOST and begins listening for HTTP Requests.

**Startup.cs**

The WebHost calls the UseStartup method before building the Host. Then startup class define Configure and ConfigureServices service method.

It configures the request pipeline which handles all requests made to the application.

It configures the services for Dependency injection.

1. ConfigureServices

The ConfigureServices allow us to add or register services to the application. Ask for these service via dependency Injection.

1. Configure

The Configure method allow us to configure the HTTP request pipeline. The HTTP request pipeline specifies the how the application should respond to HTTP requests. The Components that make up the request pipeline are called middleware.

**Model Binding**

Model Binder is well designed bridge between the HTTP request and the C# action methods. It makes it easy for developers to work with models.

**ASP.NET models**

Models is responsible for the application’s state and non-UI specific behavior.

**Types**

1. **Domain Model**

Complex business logic within a domain model.

1. **View Model**

That includes just the data a view requires for display. Also simplify model binding.

1. **API Model**

Exposing an API Models.

1. **Persistence Model**

Similar as Domain Model and mapping with different models (e.g. repository classes)

**Extension Method**

Extension method allow you to inject additional methods without modifying, deriving or recompiling the original class, struct or interface. Extension methods can be added to your own custom class.

**Yield**

1. It helps to provide custom iteration without creating temp collections.
2. It helps to do stateful iteration.

**IEnumerabel**

IEnumerable is an interface defining a single method GetEnumerator() that returns an IEnumerator interface. This work for read-only access to a collection that implements that IEnumerable can be used with a foreach statements.

**GetEnumerator**()

This methods returns an enumerator that iterates through a collection.

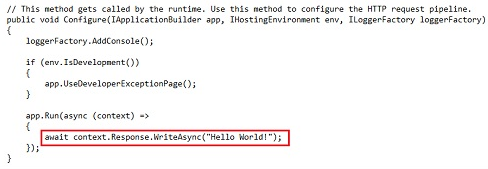
**First Blank core application**

Step 1: Open MS Visual Studio 2017 and click file -> New project.

Step 2: Select Asp.Net Core Web Application (.Net Core) from Project dialog box. Enter Name and click OK button.

Step 3: Select Empty Templates from Asp.Net Core Templates.

Step 4: Open Startup.cs and lets change the string “Hello World! This ASP.NET Core Application”.



**Read from AppSettings.json**

Step 1: Right click on your project and select Add -> New Item.

Step 2: Select Installed -> Code and Select JSON file. Enter the name AppSettings.json and click Add button.

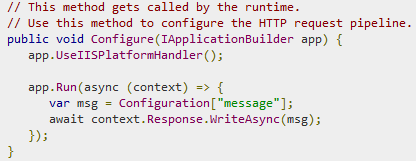
Step 3: Add the message as a key and message text into the AppSettings.json. (“Hello world! This message is from configuration file…”)

Step 4: Open the Startup.cs and change.



**Roles of Middleware’s**

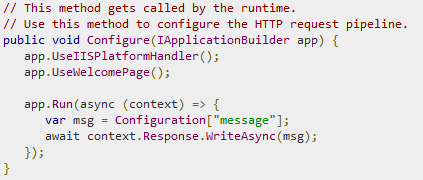
1. **IISPlatformHandler** – IISPlatformHandler allow us to work with Windows Authentication. It will look at every incoming request.
2. **app.Run** – the Run method allow us to pass in another method, which we can use to process every single response.



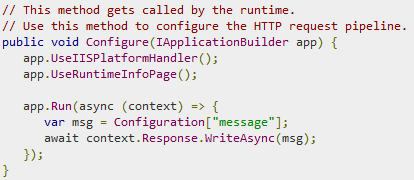
Step 1: Right click on project and Select Manage NuGet Packages.

Step 2: Search Microsoft.aspnet.diagnostics – ASP.NET core middleware for exception handling, exception display pages and diagnostic information. Below are the Diagnostics middleware’s.

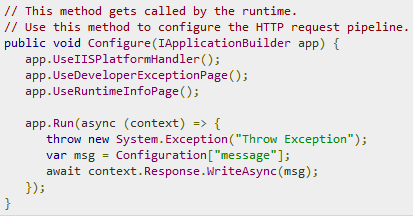
1. **app.UseWelcomePage** – Run your application and you will see the Welcome screen.



1. **app.UseRuntimeInfoPage** – only responses to requests that come in for a specific URL. If the incoming request does not match that URL, this middleware request pass through IISPlatformHandler middleware to the next piece of middleware.



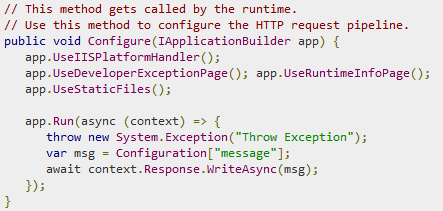
1. **app.UseDeveloperExceptionPage** – handle the HTTP 500 error.



Step 1: Right click on project and Select Manage NuGet Packages.

Step 2: Search **Microsoft.AspNet.StaticFiles** – find the static file middleware’s.

1. **app.StaticFile** – Run the html static files.



**Setup the MVC Application**

Step 1: Right click on project and Select NuGet Package Manager.

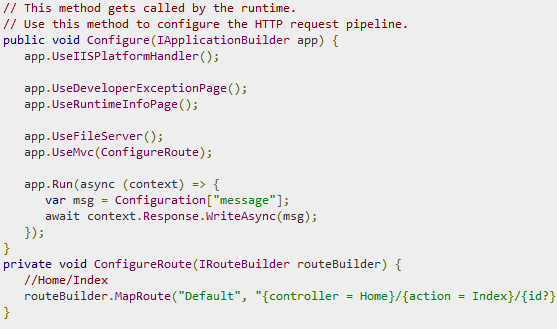
Step 2: Search **Microsoft.AspNet.Mvc** –

Step 3: Open Startup.cs and change.

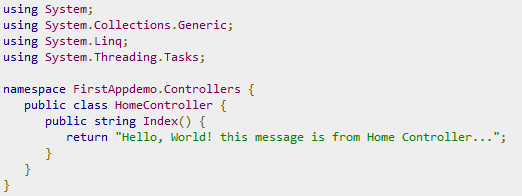


Step 4: Add **services.AddMVC();** into the ConfigurationServices function.

Step 5: Add **app.UseMvcWithDefaultRoute();** into the Configure function**. App.UseMvc()** require the Routing Configuration and below are the screen.



Step 6: Add HomeController with single public method that returns the string.

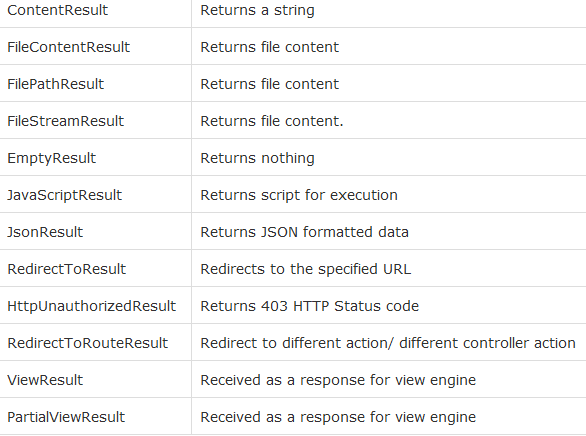


**Attribute Routing**

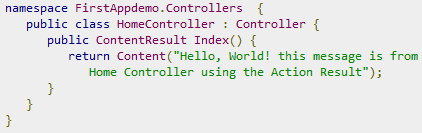


**What are the ActionResult?**

The ActionResult class is the base class for all the Action Results.



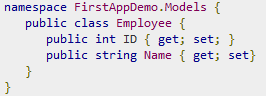
**Display MVC String Content.**



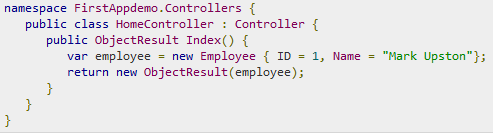
**Display MVC ObjectResult.**

Step 1: Right click on project and add Employee models class file.

Step 2: Add below code.

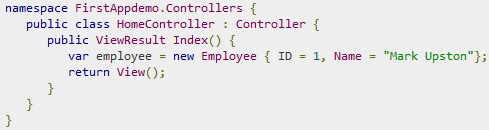


Step 2: Modified the Home Controller with below change.



**Add Views**

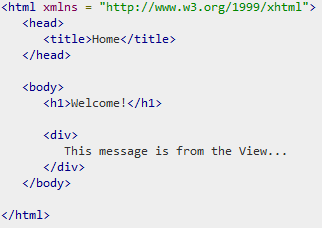
Step 1: Modified the Home Controller with below change.



Step 2: Right click home folder and Select Add -> New Items.

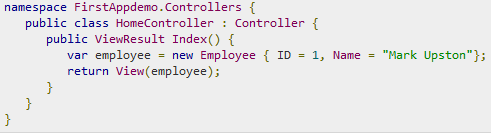
Step 3: Select Installed -> Asp.Net -> MVC View Page. Enter the page name and click Add.

Step 4: Open the View Page and change.



**Add View and Pass the Models data.**

Step 1: Modified the Home Controller with below change.



Step 2: Modified into view with below changes.



**Using Entity Framework in ASP.NET MVC Core.**

Step 1: Install the Entity Framework from NuGet Package Manager.

Step 2: Search EntityFramework – creating a database schema based on c#.

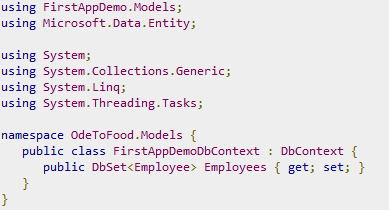
Step 3: Right click on Models folder and Add -> New Items -> Class. Enter the context name.

Step 4: Inherit the class with DbContext using Microsoft.Data.Entity.

DbContext – map to a specific database that has a schema that the DBContext understand.

DbSet<T> - map the model class or entity file.

Step 5: Now Implement a DbSet of Employee on that class.



Step 6: Modified the HomeController.



Step 7: Configure the Entity Framework Service. Add the Connection string inside AppSettings.json file.



Step 8: Open Startup.cs and modified ConfigureServices function.

