# Difference between Dot Net Frame work and Core

# 

## Difference Between MVC and ASP.Net

## 

# What is MVC?

MVC is a framework used to create web applications. The web application base builds on Model-View-Controller pattern which separates the application logic from UI, and the input and events from the user will be controlled by the Controller.

**Advantages**

* A main advantage of MVC is separation of concern. Separation of concern means we divide the application Model, Control   and View.
* We can easily maintain our application because of separation of concern.  
  In the same time we can split many developers work at a time. It  will not affects  one developer work to another developer work.
* It supports TTD (test-driven development). We can create an application with unit test. We can write won test case.
* Latest version of MVC Support default responsive web site and mobile templates.  
  We can create own view engine. It is syntax is very easy compare to traditional view engine.

**Disadvantage**

* Cannot see design page preview like .aspx page. Every time want to run then see the design.
* Understanding flow of application is very hard one. It is little bit of complex to implement and not suitable for small level applications.

It's deployment is little bit hard one.

## MVC application life

MVC application life cycle has two main phases first creating the request object and second sending our response to the browser.

**Step 1 Fill route: -**MVC requests are mapped to route tables which in turn specify which controller and action to be invoked. So if the request is the first request the first thing is to fill the route table with routes collection. This filling of route table happens in the global.asax file.

**Step 2 Fetch route:-**Depending on the URL sent "UrlRoutingModule" searches the route table to create "RouteData" object which has the details of which controller and action to invoke.

**Step 3 Request context created: -**The "RouteData" object is used to create the "RequestContext" object.

**Step 4 Controller instance created: -**This request object is sent to "MvcHandler" instance to create the controller class instance. Once the controller class object is created it calls the "Execute" method of the controller class.

**Creating Response object: -**This phase has two steps executing the action and finally sending the response as a result to the view.

**Step 5 Execute Action: - The "ControllerActionInvoker**" determines which action to executed and executes the action.

**Step 6 Result sent: -**The action method executes and creates the type of result which can be a view result , file result , JSON result etc.

# Routing & Action Result

**Routing** is a mechanism in **MVC** that decides which action method of a controller class to execute. MVC routing sends request to appropriate controller and action method based on URL and configured Routes

Route defines the URL pattern and handler information. All the configured routes of an application stored in RouteTable and will be used by Routing engine to determine appropriate handler class an incoming request.

## Configure a Route

You can register a route in **RouteConfig** class, which is **App\_Start** folder.

the route is configured using the MapRoute() extension method of RouteCollection, where name is "Default", url pattern is *"{controller}/{action}/{id}"* and defaults parameter for controller, action method and id parameter. Defaults specifies which controller, action method or value of id parameter should be used if they do not exist in the incoming request URL.

If the URL doesn't contain anything after domain name then the default controller and action method will handle the request.

## Multiple Routes

You can also configure a custom route using MapRoute extension method. You need to provide at least two parameters in *MapRoute, route name and url pattern. The Defaults parameter is optional.*

## Route Constraints

## You can also apply restrictions on the value of parameter by configuring route constraints.

routes.MapRoute(

name: "Student",

url: "student/{id}/{name}/{standardId}",

defaults: new { controller = "Student", action = "Index", id = UrlParameter.Optional, name = UrlParameter.Optional, standardId = UrlParameter.Optional },

constraints: new { id = @"\d+" }

);

## Register Routes

Now, after configuring all the routes in RouteConfig class, you need to register it in the Application\_Start() event in the Global.asax. So that it includes all your routes into RouteTable.

rotected void Application\_Start()

{

RouteConfig.RegisterRoutes(RouteTable.Routes);

}

}

# Action Result

MVC framework includes various result classes, which can be return from an action methods. There result classes represent different types of responses such as html, file, string, json, JavaScript etc.

The Action Result class is a base class of all the above result classes, so it can be return type of action methods which returns any type of result listed above. However, you can specify appropriate result class as a return type of action method.

The Index() method of StudentController in the above figure uses View() method to return ViewResult (which is derived from ActionResult). The View() method is defined in base Controller class. It also contains different methods, which automatically returns particular type of result as shown in the below table.

| Result Class | Description | Base Controller Method |
| --- | --- | --- |
| ViewResult | Represents HTML and markup. | View() |
| EmptyResult | Represents No response. |  |
| ContentResult | Represents string literal. | Content() |
| FileContentResult, FilePathResult, FileStreamResult | Represents the content of a file | File() |
| JavaScriptResult | Represent a JavaScript script. | JavaScript() |
| JsonResult | Represent JSON that can be used in AJAX | Json() |
| RedirectResult | Represents a redirection to a new URL | Redirect() |
| RedirectToRouteResult | Represent another action of same or other controller | RedirectToRoute() |
| PartialViewResult | Returns HTML | PartialView() |
| HttpUnauthorizedResult | Returns HTTP 403 status |  |

As you can see in the above table, View method returns ViewResult, Content method returns string, File method returns content of a file and so on. Use different methods mentioned in the above table, to return different types of results from an action method

# **Action Selectors**

Action selector is the attribute that can be applied to the action methods. It helps routing engine to select the correct action method to handle a particular request. MVC 5 includes the following action selector attributes:

1. ActionName
2. NonAction
3. ActionVerbs

## ActionName

ActionName attribute allows us to specify a different action name than the method name. Consider the following example.

Example: ActionName

public class StudentController : Controller

{

public StudentController()

{

}

[ActionName("find")]

public ActionResult GetById(int id)

{

// get student from the database

return View();

}

}

In the above example, we have applied ActioName("find") attribute to GetById action method. So now, action name is "find" instead of "GetById". This action method will be invoked on *http://localhost/student/find/1* request instead of *http://localhost/student/getbyid/1* request.

## NonAction

NonAction selector attribute indicates that a public method of a Controller is not an action method. Use NonAction attribute when you want public method in a controller but do not want to treat it as an action method.

For example, the GetStudent() public method cannot be invoked in the same way as action method in the following example.

Example: NonAction

public class StudentController : Controller

{

public StudentController()

{

}

[NonAction]

public Student GetStudent(int id)

{

return studentList.Where(s => s.StudentId == id).FirstOrDefault();

}

}

## ActionVerbs

In this section, you will learn about the ActionVerbs selectors attribute.

The ActionVerbs selector is used when you want to control the selection of an action method based on a Http request method. For example, you can define two different action methods with the same name but one action method responds to an HTTP Get request and another action method responds to an HTTP Post request.

The following table lists the usage of http methods:

| **Http method** | **Usage** |
| --- | --- |
| GET | To retrieve the information from the server. Parameters will be appended in the query string. |
| POST | To create a new resource. |
| PUT | To update an existing resource. |
| HEAD | Identical to GET except that server do not return message body. |
| OPTIONS | OPTIONS method represents a request for information about the communication options supported by web server. |
| DELETE | To delete an existing resource. |
| PATCH | To full or partial update the resource. |

# **Model &View & Controller**

Model represents domain specific data and business logic in MVC architecture. It maintains the data of the application. Model objects retrieve and store model state in the persistance store like a database.

Model class holds data in public properties. All the Model classes reside in the Model folder in MVC folder structure.

**Controller**

Controller class contains public methods called **Action** methods. Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.

In ASP.NET MVC, every controller class name must end with a word "Controller".

Every controller in MVC must derived from this abstract Controller class. This base Controller class contains helper methods that can be used for various purposes.

**Action methods**

All the public methods of a Controller class are called Action methods. They are like any other normal methods with the following restrictions:

1. Action method must be public. It cannot be private or protected
2. Action method cannot be overloaded
3. Action method cannot be a static method.

.

## Razor View Engine

Microsoft introduced the Razor view engine and packaged with MVC 3. You can write a mix of html tags and server side code in razor view. Razor uses @ character for server side code instead of traditional <% %>. You can use C# or Visual Basic syntax to write server side code inside razor view. Razor view engine maximize the speed of writing code by minimizing the number of characters and keystrokes required when writing a view. Razor views files have .cshtml or vbhtml extension.

ASP.NET MVC supports following types of view files:

| **View file extension** | **Description** |
| --- | --- |
| .cshtml | C# Razor view. Supports C# with html tags. |
| .vbhtml | Visual Basic Razor view. Supports Visual Basic with html tags. |
| .aspx | ASP.Net web form |
| .ascx | ASP.NET web control |

# **MVC Model Binding**

Model binding is a mechanism [ASP.NET MVC](https://www.asp.net/mvc) uses to create parameter objects defined in controller action methods. The parameters can be of any type, from simple to complex ones. It simplifies working with data sent by the browser because data is automatically assigned to the specified model. Without this mechanism, developers would need to manually assign values to model properties, which would result in writing almost the same code in every action method.

ASP.NET MVC model binder allows you to map Http Request data with the model. Http Request data means when a user makes a request with form data from the browser to a Controller, at that time, Model binder works as a middleman to map the incoming HTTP request with Controller action method.

Model binding is a two-step process. First, it collects values from the incoming http request and second, populates primitive type or complex type with these values.

By default, the DefaultModelBinder searches for the values in four locations, in the following order:

| **SOURCE** | **DESCRIPTION** |
| --- | --- |
| Request.Form | Values submitted in a form |
| RouteData.Values | Route values, e.g. *id* in /Employee/Edit/{id} |
| Request.QueryString | Data extracted from the query string portion of the URL |
| Request.Files | Uploaded files |

The search stops as soon as the value is found. In this case, the binder will first look in the Form collection. Since there was no form submitted and thus no value named id in that collection, it will look in the RouteData dictionary. The binder finds what it was looking for and does not go on to search in QueryString and Files collections. The ControllerActionInvoker now has all that’s necessary to invoke the method.

**Model Class validation useing ModelState.IsValid propertie**

ModelState.IsValid is responsible for the Model data validation.

Suppose that If we will enter the wrong data which will not map with Model class Field data type at that time it will return false.

ASP.net MVC provides various options to map the http request (browser request or form data) /view’s data to model properties via controller action method.

1. Form Collection
2. Model binding

## Bind Attribute

ASP.NET MVC framework also enables you to specify which properties of a model class you want to bind. The [Bind] attribute will let you specify the exact properties a model binder should include or exclude in binding.

In the following example, Edit action method will only bind StudentId and StudentName property of a Student model.

# HTML Helpers

* **HTML Helpers** are methods that return a string.
* Helper class can create HTML controls programmatically. HTML Helpers are used in View to render HTML content.
* It is not mandatory to use HTML Helper classes for building an ASP.NET MVC application.
* We can build an ASP.NET MVC application without using them, but HTML Helpers helps in the rapid development of a view.
* HTML Helpers are more lightweight as compared to ASP.NET Web Form controls as they do not use ViewState and do not have event models.
* MVC has built-in Helpers methods
* We can create custom HTML helpers.

## Type of HTML Helpers?

HTML Helpers are categorized into three types,

1. Inline HTML helpers
2. Built-in HTML helpers
   1. Standard HTML Helpers
   2. Strongly Typed HTML helpers
   3. Templated HTML helpers
3. Custom HTML helpers

## Advantages of using Inline HTML Helper in MVC 5

* It is reusable on the same view.
* It reduces the code repetition
* It is simple to create and easy to use.
* It is easy to customization the method based on the requirement.

**Syntax of Inline Html Helper**

@helper HelperName(Parameters list..)

{

// code.....

}

# DataAnnotations

ASP.NET MVC uses DataAnnotations attributes to implement validations. DataAnnotations includes built-in validation attributes for different validation rules, which can be applied to the properties of model class. ASP.NET MVC framework will automatically enforce these validation rules and display validation messages in the view.

The DataAnnotations attributes included in *System.ComponentModel.DataAnnotations* namespace. The following table lists DataAnnotations validation attributes.

| **Attribute** | **Description** |
| --- | --- |
| Required | Indicates that the property is a required field |
| StringLength | Defines a maximum length for string field |
| Range | Defines a maximum and minimum value for a numeric field |
| RegularExpression | Specifies that the field value must match with specified Regular Expression |
| CreditCard | Specifies that the specified field is a credit card number |
| CustomValidation | Specified custom validation method to validate the field |
| EmailAddress | Validates with email address format |
| FileExtension | Validates with file extension |
| MaxLength | Specifies maximum length for a string field |
| MinLength | Specifies minimum length for a string field |
| Phone | Specifies that the field is a phone number using regular expression for phone numbers |

The Html.ValidationMessage() is an extension method, that is a loosely typed method. It displays a validation message if an error exists for the specified field in the ModelStateDictionary object.

## ASP.NET MVC: ValidationMessageFor

The Html.ValidationMessageFor() is a strongly typed extension method. It displays a validation message if an error exists for the specified field in the ModelStateDictionary objec

## ASP.NET MVC: ValidationSummary

The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object.

The ValidationSummary can be used to display all the error messages for all the fields. It can also be used to display custom error messages. Th

# Use Layout View & Partiel View

### layout view

An application may contain common parts in the UI which remains the same throughout the application such as the logo, header, left navigation bar, right bar or footer section. ASP.NET MVC introduced a Layout view which contains these common UI parts, so that we don't have to write the same code in every page. The layout view is same as the master page of the ASP.NET webform application.

You must be wondering that how would the View know which layout view to use?

You can set the layout view in multiple ways, by using \_ViewStart.cshtml or setting up path of the layout page using Layout property in the individual view or specifying layout view name in the action method.

### **\_ViewStart.cshtml**

\_ViewStart.cshtml is included in the Views folder by default. It sets up the default layout page for all the views in the folder and its subfolders using the Layout property. You can assign a valid path of any Layout page to the Layout property.

For example, the following \_ViewStart.cshtml in the **Views** folder, sets the Layout property to "~/Views/Shared/\_Layout.cshtml". So now, \_layout.cshtml would be layout view of all the views included in Views and its subfolders. So by default, all the views derived default layout page from \_ViewStart.cshtml of Views folder.

### **Specify Layout Page in ActionResult Method**

You can also specify which layout page to use in while rendering view from action method using View() method.

public class HomeController : Controller

{

public ActionResult Index()

{

return View("Index", "\_myLayoutPage");

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

### Rendering Methods

ASP.NET MVC layout view renders child views using the following methods.

| Method | Description |
| --- | --- |
| RenderBody() | Renders the portion of the child view that is not within a named section.  Layout view must include RenderBody() method. |
| RenderSection(string name) | Renders a content of named section and specifies whether the section is required. RenderSection() is optional in Layout view. |

## Partial view

Partial view is a reusable view, which can be used as a child view in multiple other views. It eliminates duplicate coding by reusing same partial view in multiple places. You can use the partial view in the layout view, as well as other content views.

Render Partial View

You can render the partial view in the parent view using html helper methods: Partial() or RenderPartial() or RenderAction(). Each method serves different purposes. Let's have an overview of each method and then see how to render partial view using these methods.

### **Html.Partial()**

@Html.Partial() helper method renders the specified partial view. It accept partial view name as a string parameter and returns MvcHtmlString. It returns html string so you have a chance of modifing the html before rendering.

### **Html.RenderPartial()**

The RenderPartial helper method is same as the Partial method except that it returns void and writes resulted html of a specified partial view into a http response stream directly.

### **Html.RenderAction()**

The RenderAction helper method invokes a specified controller and action and renders the result as a partial view. The specified Action method should return PartialViewResult using the Partial() method.

# ViewBag

1. ViewBag is used to pass data from the controller to the respective view
2. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
3. It is also available for the current request only
4. If redirection occurs, then its value becomes null
5. Doesn’t require typecasting for complex data type

## ViewData

1. ViewData is used to pass data from controller to view
2. It is derived from ViewDataDictionary class
3. It is available for the current request only
4. Requires typecasting for complex data type and checks for null values to avoid error
5. If redirection occurs, then its value becomes null

## TempData

1. TempData is derived from TempDataDictionary class
2. TempData is used to pass data from the current request to the next request
3. It keeps the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain the data when we move from one controller to another controller or from one action to another action
4. It requires typecasting for complex data type and checks for null values to avoid error. Generally, it is used to store only one time messages like the error messages and validation messages

# **ASP.NET MVC- Filters**

ASP.NET MVC Filter is a custom class where you can write custom logic to execute before or after an action method executes. Filters can be applied to an action method or controller in a declarative or programmatic way. Declarative means by applying a filter attribute to an action method or controller class and programmatic means by implementing a corresponding interface.

| **Filter Type** | **Description** | **Built-in Filter** | **Interface** |
| --- | --- | --- | --- |
| Authorization filters | Performs authentication and authorizes before executing action method. | [Authorize], [RequireHttps] | IAuthorizationFilter |
| Action filters | Performs some operation before and after an action method executes. |  | IActionFilter |
| Result filters | Performs some operation before or after the execution of view result. | [OutputCache] | IResultFilter |
| Exception filters | Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline. | [HandleError] | IExceptionFilter |

Action filter executes before and after an action method executes. Action filter attributes can be applied to an individual action method or to a controller. When action filter applied to controller then it will be applied to all the action methods in that controller.

## Error Hnadlin in MVC

* Try-Catch
* Override onException from Controller class
* HandleError Attribute
* CustomException by HandleErrorAttribute – (Exception contxt),
* We cannot log the Exception using the HandleError attribute.
* It is not possible to handle the exceptions raised outside the controllers.
* Exception Handling is not possible Based on Scenario.

Custom filter for logging errors ‘

1. Create an error log folder under root directory.
2. Creates a custom filter calls that implement IExceptionFilter and FilterAttribute
3. Over ride onexception(ExceptionContext filtrcontext )
4. **if** **(**!filterContext.ExceptionHandled**)**

* **using** System;
* **using** System.IO;
* **using** System.Web;
* **using** System.Web.Mvc;
* **namespace** ExceptionFilterInMVC.Models
* **{**
* **public** class LogCustomExceptionFilter : FilterAttribute, IExceptionFilter
* **{**
* **public** **void** OnException**(**ExceptionContext filterContext**)**
* **{**
* **if** **(**!filterContext.ExceptionHandled**)**
* **{**
* var exceptionMessage = filterContext.Exception.Message;
* var stackTrace = filterContext.Exception.StackTrace;
* var controllerName = filterContext.RouteData.Values**[**"controller"**]**.ToString**()**;
* var actionName = filterContext.RouteData.Values**[**"action"**]**.ToString**()**;
* string Message = "Date :" + DateTime.Now.ToString**()** + ", Controller: " + controllerName + ", Action:" + actionName +
* "Error Message : " + exceptionMessage
* + Environment.NewLine + "Stack Trace : " + stackTrace;
* //saving the data in a text file called Log.txt
* //You can also save this in a dabase
* File.AppendAllText**(**HttpContext.Current.Server.MapPath**(**"~/Log/Log.txt"**)**, Message**)**;
* filterContext.ExceptionHandled = **true**;
* filterContext.Result = **new** ViewResult**()**
* **{**
* ViewName = "Error"
* **}**;
* **}**
* **}**
* **}**
* **}**

As you can see, the above class is derived from the **FilterAttribute**class. This is required we want to use this class as like a Filter attribute. The class implementing the **IExceptionFilter** interface and its **OnException()**method. This method contains the logic for storing the exception information in a text (i.e. .txt) file.

## **How to Register Custom Exception Filter in ASP.NET MVC?**

As a filter, you can use it at three different levels i.e. at the action level, at the Controller level and globally. Let’s register this filter globally so that it will be applicable for all the action methods of all controllers of our application. To register it globally, open the **FilterConfig** class which is present in the **App\_Start** folder and then modify the class as shown below. If the FilterConfig class is not already present there then just create a class file with the name **FilterConfig.cs** within the **App\_Start** folder.

**FilterConfig.cs**

* **using** System.Web.Mvc;
* **using** ExceptionFilterInMVC.Models;
* **namespace** ExceptionFilterInMVC.App\_Start
* **{**
* **public** class FilterConfig
* **{**
* **public** static **void** RegisterGlobalFilters**(**GlobalFilterCollection filters**)**
* **{**
* filters.Add**(new** LogCustomExceptionFilter**())**;
* **}**
* **}**

**Action filter attribute**

## Custom Action Filter

You can create custom action filter by two ways. First, by implementing IActionFilter interface and FilterAttribute class. Second, by deriving ActionFilterAttribute abstract class.

This will allow the developer to log information .

 Custom ActionFilter for Logging

public class LogAttribute : ActionFilterAttribute

public override void OnActionExecuted(ActionExecutedContext filterContext)

{

Log("OnActionExecuted", filterContext.RouteData);

}

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

Log("OnActionExecuting", filterContext.RouteData);

}

public override void OnResultExecuted(ResultExecutedContext filterContext)

{

Log("OnResultExecuted", filterContext.RouteData);

}

public override void OnResultExecuting(ResultExecutingContext filterContext)

{

Log("OnResultExecuting ", filterContext.RouteData);

}

an apply Log attribute to any Controller or action methods where you want to log the action.

[Log]

public class StudentController : Controller

{

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

# **Web Api**

**Wriring Logs**

# **Microservices**

**Microservices** are a design pattern in which applications are composed of small, independent modules that communicate with each other using well-defined contracts. Each **microservice** focuses on a single concept.

Microservices are independent services directly calling their own database. Whereas in monolithic architecture, the database remains the same for all the functionalities.

# **Advantages of microservices**

* Cleaner code, Easy to Maintain, Unit test and Code reuse.
* **Improved fault isolation**: Larger applications can remain mostly unaffected by the failure of a single module.
* **Testable** :since its small its easy for testing.
* **Loosely Coupled**:independent of other service
* **Eliminate vendor or technology lock-in**: Microservices provide the flexibility to try out a new technology stack on an individual service as needed. There won’t be as many dependency concerns and rolling back changes becomes much easier. With less code in play, there is more flexibility.
* **Ease of understanding:**With added simplicity, developers can better understand the functionality of a service.
* **Smaller and faster deployments**: Smaller codebases and scope = quicker deployments, which also allow you to start to explore the benefits of Continuous Deployment.
* **Scalability**: Since your services are separate, you can more easily scale the most needed ones at the appropriate times, as opposed to the whole application. When done correctly, this can impact cost savings.

# **Disadvantages of microservices**

* **Communication between services is complex**: Since everything is now an independent service, you have to carefully handle requests traveling between your modules. In one such scenario, developers may be forced to write extra code to avoid disruption. Over time, complications will arise when remote calls experience latency.
* **More services equals more resources**: Multiple databases and transaction management can be difficult.
* **Global testing is difficult**: Testing a microservices-based application can be cumbersome. In a monolithic approach, we would just need to launch our WAR on an application server and ensure its connectivity with the underlying database. With microservices, each dependent service needs to be confirmed before testing can occur.
* **Debugging problems can be harder**: Each service has its own set of logs to go through. Log, logs, and more logs.
* **Deployment challengers**: The product may need coordination among multiple services, which may not be as straightforward as deploying a WAR in a container.
* **Large vs small product companies**: Microservices are great for large companies, but can be slower to implement and too complicated for small companies who need to create and iterate quickly, and don’t want to get bogged down in complex orchestration.

Stateless Protocol:  
Stateless Protocols are the type of network protocols in which Client send request to the server and server response back according to current state. It does not require the server to retain session information or a status about each communicating partner for multiple request.

**Silent features of Stateless Protocols:**

* Stateless Protocol simplify the design of Server.
* The stateless protocol requires less resources because system do not need to keep track of the multiple link communications and the session details.
* In Stateless Protocol each information packet travel own it’s own without reference to any other packet.
* Each communication in Stateless Protocol is discrete and unrelated to those that precedes or follow.

**Stateful Protocol:**  
In Stateful Protocol If client send a request to the server then it expects some kind of response, if it does not get any response then it resend the request. [FTP (File Transfer Protocol)](https://www.geeksforgeeks.org/computer-network-file-transfer-protocol-ftp/), [Telnet](https://www.geeksforgeeks.org/difference-ssh-telnet/) are the example of **Stateful Protocol**.

**Silent features of Stateful Protocol:**

* Stateful Protocols provide better performance to the client by keeping track of the connection information.
* Stateful Application require Backing storage.
* Stateful request are always dependent on the server-side state.
* TCP session follow stateful protocol because both systems maintain information about the session itself during its life.

# Comparisons between Stateless and Stateful Protocol:

| **STATELESS PROTOCOL** | **STATEFUL PROTOCOL** |
| --- | --- |
| Stateless Protocol does not require the server to retain the server information or session details. | Stateful Protocol require server to save the status and session information. |
| In Stateless Protocol, there is no tight dependency between server and client. | In Stateful protocol, there is tight dependency between server and client |
| The Stateless protocol design simplify the server design. | The Stateful protocol design makes the design of server very complex and heavy. |
| Stateless Protocols works better at the time of crash because there is no state that must be restored, a failed server can simply restart after a crash. | Stateful Protocol does not work better at the time of crash because stateful server have to keep the information of the status and session details of the internal states. |
| Stateless Protocols handle the transaction very fastly. | Stateful Protocols handle the transaction very slowly. |
| Stateless Protocols are easy to implement in Internet. | Stateful protocols are logically heavy to implement in Internet. |

Create a class implementin ActionFilterAttribute

Over ride OnActionExcected and OnActionExcecuting

And trace.write to log details

Another way of creating LogAttribute class is by implementing IActionFilter interfaceSo now, you can apply [Log] attributes

Global Exception Handling

Create “CustomHandler” folder in this folder we are going to add “GlobalExceptionHandler” class.

Create a class “GlobalExceptionHandler” which will inherit from “ExceptionHandler” abstract class inside this we are going to implement Handle method

Override the handle method of ‘ExceptionHandler’

 add a class “ErrorMessageResult” which will inherit from “IHttpActionResult” interface. This class will have a Parameterized Constructor which takes 2 parameters 1. HttpRequestMessage ,2. HttpResponseMessage the HttpResponseMessage which we took parameters will be used by ExecuteAsync to create HttpResponseMessage.

Then this HttpResponseMessage we are going to assign it to “context.Result”

## Registering Exception handler

We are going to Register “GlobalExceptionHandler” in WebApiConfig class, such that any web API exception can be handled globally.

config.Services.Replace(typeof(IExceptionHandler), new GlobalExceptionHandler());

#### **Validation filters**

And then we can validate all our models in our ValidationActionFilter.

public class ValidationActionFilter: ActionFilterAttribute

{

public override void OnActionExecuting(HttpActionContext actionContext)

{

var modelState = actionContext.ModelState;

if (!modelState.IsValid) {

actionContext.Response = actionContext.Request

.CreateErrorResponse(HttpStatusCode.BadRequest, modelState);

}

}

}

## HttpResponseException

This exception class allows us to return HttpResponseMessage to the client. It returns HTTP status code that is specified in the exception Constructor. We have more control over the response because we can pass the entire response message (using HttpResponseMessage) to the Constructor of HttpResponseException.

## HttpError

CreateErrorResponse method of Request object helps us to return meaningful error code and message to the client. CreateErrorResponse creates an instance of Http

returnRequest.CreateErrorResponse(HttpStatusCode.NotFound, message); Error object and returns it as HttpResponseMessage object.

## Create a LoggerHandler

With the GlobalExceptionHanlder, there’s still a chance that an exception will not be caught here. However, we’re guaranteed to be able to log it. And we’re able to set up as many exception loggers as we need. I’m just going to use [log4net’s](https://stackify.com/log4net-guide-dotnet-logging/) logger in this example.

public class Log4NetExceptionLogger: ExceptionLogger

{

private ILog log = LogManager.GetLogger(typeof(Log4NetExceptionLogger));

public async override Task LogAsync(ExceptionLoggerContext context, System.Threading.CancellationToken cancellationToken)

{

log.Error("An unhandled exception occurred.", context.Exception);

await base.LogAsync(context, cancellationToken);

}

public override void Log(ExceptionLoggerContext context)

{

log.Error("An unhandled exception occurred.", context.Exception);

base.Log(context);

}

And again, make sure to add it to your configuration.

public static class WebApiConfig

{

public static IAppBuilder RegisterApiConfig(this IAppBuilder app, HttpConfiguration config)

{

config.MapHttpAttributeRoutes();

config.Services.Add(typeof(IExceptionLogger), new Log4NetExceptionLogger());

return app;

}

}

## Exception Filters

Exception filters can be used to handle unhandled exceptions which are generated in Web API. The exception filter can be able to catch the unhandled exceptions in Web API. This filter is executed when an action method throws the unhandled exception.

Create a class and inherit   ExceptionFilterAttribut class

**public** **class** CustomExceptionFilter: ExceptionFilterAttribut

1. override  **public** **override** **void** OnException(HttpActionExecutedContextactionExecutedContext)

To apply the exception filter to all Web API controllers, the filter needs to register to GlobalConfiguration.Configuration.Filters collection.

1. **public** **static** **class** WebApiConfig
2. {
3. **public** **static** **void** Register(HttpConfigurationconfig)
4. {
5. config.Filters.Add(newCustomExceptionFilter());
6. }
7. [CustomExceptionFilter]
8. **public** HttpResponseMessage Get([FromUri]**int** key)
9. [CustomExceptionFilter]
10. **public** **class** EmployeeController : ApiController

# **API**

The API Gateway acts as a proxy between the API consumer and the API servers.  An API Gateway is a server that is the single entry point into the system. The API Gateway encapsulates the internal system architecture and provides an API that is personalized to each client. It has responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling.

Pros and cons of API Gateway

Using the API Gateway itself has its own pros and cons. In the previous section, we have described the advantages of using the API Gateway already. I will still try to make them in points as the pros of the API Gateway.

**Pros**

* Micro service can focus on business logic
* Clients can get all the data in a single hit
* Authentication, logging, and monitoring can be handled by the API Gateway
* Gives flexibility to use completely independent protocols in which clients and microservice can talk
* It can give tailor-made results, as per the clients needs
* It can handle partial failure

Addition to the preceding mentioned pros, some of the trade-offs are also to use this pattern.

**Cons**

* It can cause performance degrade due to lots of happenings on the API Gateway
* With this, discovery service should be implemented
* Sometimes, it becomes the single point of failure
* Managing routing is an overhead of the pattern
* Adding additional network hope in the call
* Overall. it increases the complexity of the system
* Too much logic implementation in this gateway will lead to another dependency problem

So, before using the API Gateway, both of the aspects should be considered. Decision of including the API Gateway in the system increases the cost as well. Before putting effort, cost, and management in this pattern, it is recommended to analysis how much you can gain from it.

# **Dot Net Core**

It is a free open source cross-platform framework that runs on Windows, macOS, and Linux operating systems.

.NET Core Framework can be used to build different types of applications such as mobile, desktop, web, cloud, IoT, machine learning, microservices, game, etc.

## .NET Core Characteristics

**Cross-platform**

**Open-source Framework**

**Wide-range of Applications**

**Supports Multiple Languages:** You can use C#, F#, and Visual Basic programming languages to develop

**CLI Tools:** .NET Core includes [CLI tools](https://www.tutorialsteacher.com/core/net-core-command-line-interface) (Command-line interface) for development and continuous-integration.

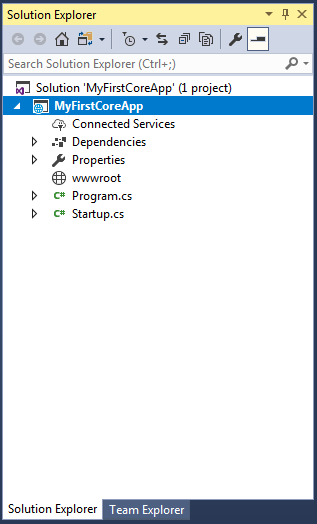
**Flexible Deployment:** .NET Core application can be deployed user-wide or system-wide or with [Docker Containers](https://docs.microsoft.com/en-us/dotnet/core/docker/introduction).

**IoC Container:** It includes the built-in IoC container for automatic dependency injection which makes it maintainable and testable.

**Hosting:** ASP.NET Core web application can be hosted on multiple platforms with any web server such as IIS, Apache etc. It is not dependent only on IIS as a standard .NET Framework.

**Code Sharing:** It allows you to build a class library that can be used with other .NET frameworks such as .NET Framework 4.x or Mono. Thus a single code base can be shared across frameworks.

# **ASP.NET Core - Project Structure**



ASP.NET Core project files and folders are synchronized with physical files and folders. If you add a new file or folder in project folder then it will directly reflect in the solution explorer. You don't need to add it in the project explicitly by right clicking on the project.

## .csproj

ASP.NET Core 1.0 does not create .csproj file, instead, it uses .xproj and project.json files to manage the project. This has changed in ASP.NET Core 2.0. Visual Studio now uses .csproj file to manage projects. We can edit the .csproj settings by right clicking on the project and selecting **Edit <project-name>.csproj** as shown below.

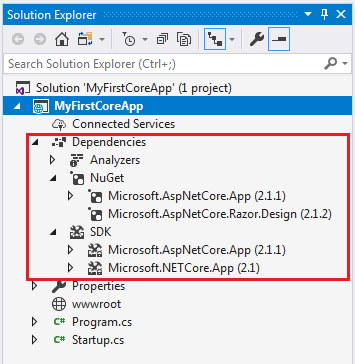
The .csproj for the above project looks like below.

[](https://www.tutorialsteacher.com/Content/images/core/csproj.png)Edit .csproj

The csproj file includes settings related to targeted .NET Frameworks, project folders, NuGet package references etc.

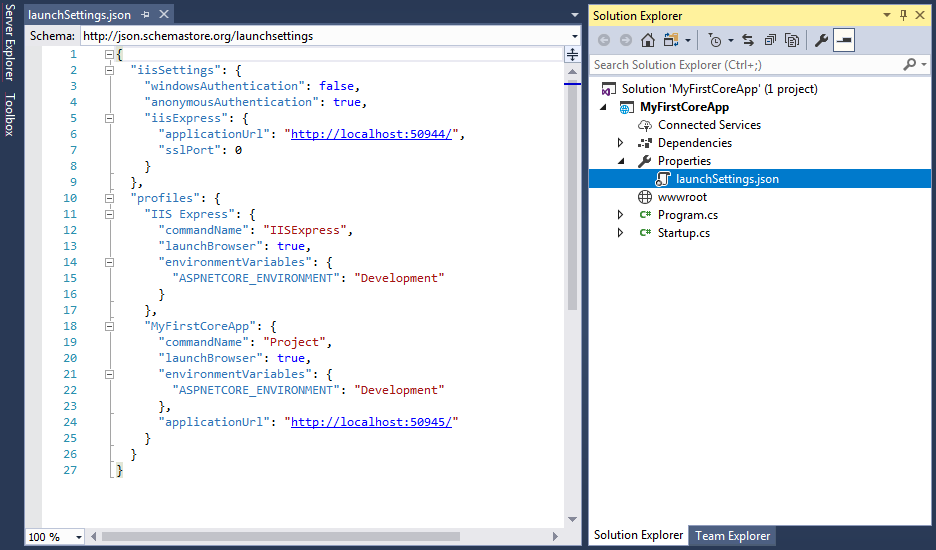
**Dependencies**

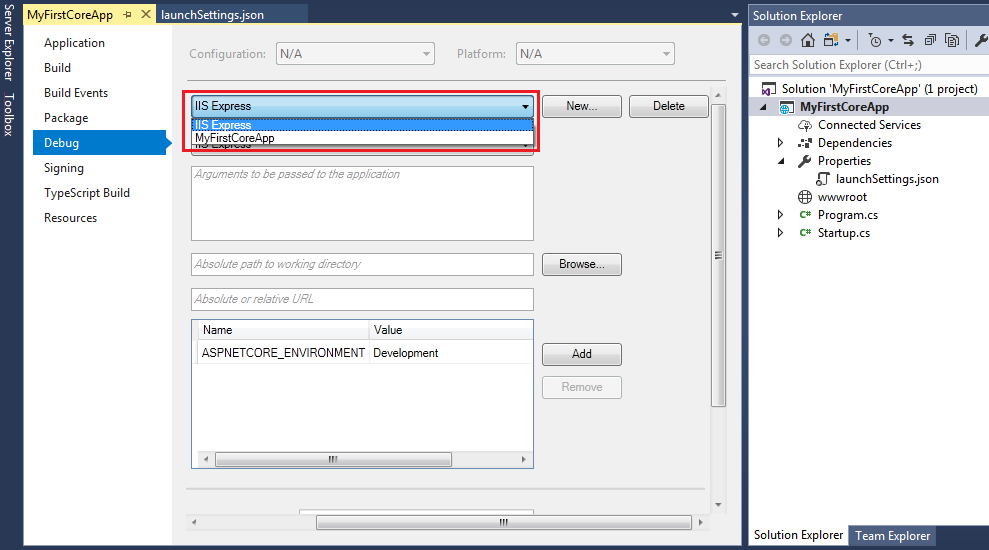
The Dependencies in the ASP.NET Core 2.1 project contain all the installed server-side NuGet packages, as shown below.

[](https://www.tutorialsteacher.com/Content/images/core/dependencies2.png)

**Properties:**

The Properties node includes launchSettings.json file which includes Visual Studio profiles of debug settings. The following is a default launchSettings.json file.

[](https://www.tutorialsteacher.com/Content/images/core/launchsettings2.png)launchSettings.json

We can also edit settings from the debug tab of project properties. Right click on the project -> select Properties -> click Debug tab.[](https://www.tutorialsteacher.com/Content/images/core/project-debug.png)Project Properties

In the debug tab, select a profile which you want to edit as shown above. You may change environment variables, url etc.

## ASP.NET Core - wwwroot Folder

By default, the **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

Generally, there should be separate folders for the different types of static files such as JavaScript, CSS, Images, library scripts etc. i

You can access static files with base URL and file name. For example, we can access above site.css file in the css folder by *http://localhost:<port>/css/app.css*.

Remember, you need to include a middleware for serving static files in the Configure method of Startup.cs.

## Rename wwwroot Folder

You can rename wwwroot folder to any other name as per your choice and set it as a web root while preparing hosting environment in the program.cs.

For example, let's rename wwwroot folder to Content folder. Now, call UseWebRoot() method to configure Content folder as a web root folder in the Main() method of Program class as shown below.

public class Program

{

public static void Main(string[] args)

{

var host = new WebHostBuilder()

.UseKestrel()

.UseContentRoot(Directory.GetCurrentDirectory())

**.UseWebRoot("Content")**

.UseIISIntegration()

.UseStartup<MyStartup>()

.Build();

host.Run();

}

}

# **ASP.NET Core - Program.cs**

ASP.NET Core web application is actually a console project which starts executing from the entry point public static void Main() in Program class where we can create a host for the web application

Program.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Hosting;

namespace MyFirstCoreApp

{

public class Program

{

public static void Main(string[] args)

{

BuildWebHost(args).Run();

}

public static IWebHost BuildWebHost(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>()

.Build();

As you can see above, the Main() method calls method expression BuildWebHost() to build web host with pre-configured defaults. The BuildWebHost expression can also be written as a method that returns IWebHost as shown below.

public static void Main(string[] args)

{

BuildWebHost(args).Run();

}

public static IWebHost BuildWebHost(string[] args)

{

return WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>()

.Build();

}

Let's understand hosting steps.

The WebHost is a static class which can be used for creating an instance of IWebHost and IWebHostBuilder with pre-configured defaults. The CreateDefaultBuilder() method creates a new instance of WebHostBuilder with pre-configured defaults. Internally, it configures Kestrel, IISIntegration and other configurations. The following is CreateDefaultBuilder() method.

# **ASP.NET Core - Startup Class**

ASP.NET Core application must include Startup class. It is like Global.asax in the traditional .NET application. As the name suggests, it is executed first when the application starts.

The startup class can be configured using UseStartup<T>() method at the time of configuring the host in the Main() method of Program class

The startup class can be configured using UseStartup<T>() method at the time of configuring the host in the Main() method of Program class

Start up class two public methods: **ConfigureServices** and **Configure**.

The Startup class must include a Configure method and can optionally include ConfigureService method.

## ConfigureServices()

The Dependency Injection pattern is used heavely in ASP.NET Core architecture. It includes built-in IoC container to provide dependent objects using constructors.

The ConfigureServices method is a place where you can register your dependent classes with the built-in IoC container. After registering dependent class, it can be used anywhere in the application. You just need to include it in the parameter of the constructor of a class where you want to use it. The IoC container will inject it automatically.

## Configure()

The Configure method is a place where you can configure application request pipeline for your application using IApplicationBuilder instance that is provided by the built-in IoC container.

ASP.NET Core introduced the middleware components to define a request pipeline, which will be executed on every request. You include only those middleware components which are required by your application and thus increase the performance of your application.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

the Configure method includes three parameters IApplicationBuilder, IHostingEnvironment, and ILoggerFactory by default. These services are framework services injected by built-in IoC container.

At run time, the ConfigureServices method is called before the Configure method. This is so that you can register your custom service with the IoC container which you may use in the Configure method.

# **.NET Core Command-Line Interface**

The .NET Core command-line interface (CLI) is a new cross-platform tool for creating, restoring packages, building, running and publishing .NET applications

The .NET Core CLI is installed with .NET Core SDK for selected platforms. So we don't need to install it separately on the development machine. We can verify whether the CLI is installed properly by opening command prompt in Windows and writing dotnet and pressing Enter. If it displays usage and help as shown below then it means it is installed properly.

## Command Structure

The following is a command structure.

dotnet <command> <argument> <option>

| Basic Commands | Description |
| --- | --- |
| [new](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new) | Creates a new project, configuration file, or solution based on the specified template. |
| [restore](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-restore) | Restores the dependencies and tools of a project. |
| [build](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-build) | Builds a project and all of its dependencies. |
| [Run](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-run) | Runs source code without any explicit compile or launch commands. |
| [publish](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-publish) | Packs the application and its dependencies into a folder for deployment to a hosting system. |
| [test](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-test) | Executes unit tests. |
| [vtest](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-vstest) | Runs tests from the specified files. |
| [pack](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-pack) | Packs the code into a NuGet package. |
| [clean](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-clean) | Cleans the output of a project. |
| [sln](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-sln) | Modifies a .NET Core solution file. |
| [help](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-help) | Display help on the specified command |
| [store](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-store) | Stores the specified assemblies in the runtime package store. |

| Project Modification Commands | Description |
| --- | --- |
| [add package](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package) | Adds a package reference to a project. |
| [add reference](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-reference) | Adds project-to-project (P2P) references. |
| [remove package](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-remove-package) | Removes package reference from the project. |
| [remove reference](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-remove-reference) | Removes project reference |
| [list reference](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-list-reference) | Lists all project-to-project references |

| Advanced Commands | Description |
| --- | --- |
| [nuget delete](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-nuget-delete) | Deletes or unlists a package from the server. |
| [nuget locals](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-nuget-locals) | Clears or lists local NuGet resources. |
| [nuget push](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-nuget-push) | Pushes a package to the server and publishes it. |
| [msbuild](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-msbuild) | Builds a project and all of its dependencies. |
| [dotnet install script](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-install-script) | Script used to install the .NET Core CLI tools and the shared runtime. |

## Create a New Project

## The following command creates a new console application named MyConsoleApp to MyProjects directory. The -o or --output option is used to specify an output directory where the project should be generated.

**Dotnet new console –n projectname –o C:\my**

## add Package Reference

C:\MyConsoleApp>dotnet add package Newtonsoft.json

**Restore Packages**

To restore packages or to update existing packages

dotnet restore

**Build Project**

To build a new or existing project, apply C:\MyConsoleApp>dotnet build command.

**Run project**

To run our console project, apply dotnet run command as shown below.

# **ASP.NET Core - Middleware**

A middleware is nothing but a component (class) which is executed on every request in ASP.NET Core application. In the classic ASP.NET, HttpHandlers and HttpModules were part of request pipeline. Middleware is similar to HttpHandlers and HttpModules where both needs to be configured and executed in each request

Typically, there will be multiple middleware in ASP.NET Core web application. It can be either framework provided middleware, added via NuGet or your own custom middleware. We can set the order of middleware execution in the request pipeline. Each middleware adds or modifies http request and optionally passes control to the next middleware component.

Configure Middleware

We can configure middleware in the Configure method of the Startup class using IApplicationBuilder instance.

The following example adds a single middleware using **Run method** which returns a string "Hello World!" on each request.

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

//configure middleware using IApplicationBuilder here..

app.Run(async (context) =>

{

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

});

// other code removed for clarity..

}

In the above example, Run() is an extension method on IApplicationBuilder instance which adds a terminal middleware to the application's request pipeline. The above configured middleware returns a response with a string "Hello World!" for each request.

**Run method**

The Run method is an extension method on IApplicationBuilder and accepts a parameter of RequestDelegate. The RequestDelegate is a delegate method which handles the request.

**Use() extension**

To configure multiple middleware, use Use() extension method. It is similar to Run() method except that it includes next parameter to invoke next middleware in the sequence.

**Add Built-in Middleware Via NuGet**

ASP.NET Core is a modular framework. We can add server side features we need in our application by installing different plug-ins via NuGet. There are many middleware plug-ins available which can be used in our application.

The followings are some built-in middleware:

| Middleware | Description |
| --- | --- |
| Authentication | Adds authentication support. |
| CORS | Configures Cross-Origin Resource Sharing. |
| Routing | Adds routing capabilities for MVC or web form |
| Session | Adds support for user session. |
| StaticFiles | Adds support for serving static files and directory browsing. |
| Diagnostics | Adds support for reporting and handling exceptions and errors. |

Let's see how to use Diagnostics middleware.

Diagnostics Middleware

Let's install and use Diagnostics middleware. Diagnostics middleware is used for reporting and handling exceptions and errors in ASP.NET Core, and diagnosing Entity Framework Core migrations errors.

Open project.json and add Microsoft.AspNetCore.Diagnostics dependency if it is not added. Wait for some time till Visual Studio restores the packages.

This package includes following middleware and extension methods for it.

| Middleware | Extension Method | Description |
| --- | --- | --- |
| DeveloperExceptionPageMiddleware | UseDeveloperExceptionPage() | Captures synchronous and asynchronous exceptions from the pipeline and generates HTML error responses. |
| ExceptionHandlerMiddleware | UseExceptionHandler() | Catch exceptions, log them and re-execute in an alternate pipeline. |
| StatusCodePagesMiddleware | UseStatusCodePages() | Check for responses with status codes between 400 and 599. |
| WelcomePageMiddleware | UseWelcomePage() | Display Welcome page for the root path. |

We can call respective Use\* extension methods to use the above middleware in the configure method of Startup class.

Let's add welcomePage middleware which will display welcome page for the root path.

Example: Add Diagnostics Middleware

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

app.UseWelcomePage();

//other code removed for clarity

}

## Add Custom Middleware in ASP.NET Core Application

The custom middleware component is like any other .NET class with Invoke() method. However, in order to execute next middleware in a sequence, it should have RequestDelegate type parameter in the constructor.

create middleware class and select Add -> New Item. This will open Add New Item popup. Search for word "middleware" in the top right search box as shown below.

This will add a new class for the middleware with extension method as shown below.

As you can see above, the Invoke() method is not asynchronous. So, change it to asynchronous and write your custom logic before calling next();

You may need to install the Microsoft.AspNetCore.Http.Abstractions package into your project

public class MyMiddleware

{

private readonly RequestDelegate \_next;

public MyMiddleware(RequestDelegate next)

{

\_next = next;

}

public Task Invoke(HttpContext httpContext)

{

return \_next(httpContext);

}

}

// Extension method used to add the middleware to the HTTP request pipeline.

public static class MyMiddlewareExtensions

{

public static IApplicationBuilder UseMyMiddleware(this IApplicationBuilder builder)

{

return builder.UseMiddleware<MyMiddleware>();

}

}

**Add Custom Middleware**

Now, we need to add our custom middleware in the request pipeline by using Use extension method as shown below.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

app.UseMyMiddleware();

app.Run(async (context) =>

{

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

});

}

# **Configure the Default File to be Served on the Root Request**

As we learned in the Set Default File section, app.UseDefaultFiles() middleware serves the following files on the root request.

1. Default.html
2. Default.htm
3. Index.html
4. Index.htm

Suppose, you want to set home.html as a default page which should be displayed on the root access. To do that, specify DefaultFilesOptions in the UseDefaultFiles method as shown below.

public class Startup

{

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

DefaultFilesOptions options = new DefaultFilesOptions();

options.DefaultFileNames.Clear();

options.DefaultFileNames.Add("home.html");

app.UseDefaultFiles(options);

app.UseStaticFiles();

app.Run(async (context) =>

{

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

});

}

}

Now, this will display home.html from wwwroot folder on the root request *http://localhost:<port>*.

ASP.NET Core uses an environment variable called ASPNETCORE\_ENVIRONMENT to indicate the runtime environment.

# Exception handling

## Install Microsoft.AspNetCore.Diagnostics Package

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
2. UseExceptionHandler

### **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.

### **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.

# **Serving Static Files**

ASP.NET Core application cannot serve static files by default. We must include Microsoft.AspNetCore.StaticFiles middleware in the request pipeline

Create static file in wwwroot folder

Now, to serve the above Default.html static file, we must add StaticFiles middleware in the Configure()

app.UseStaticFiles();

As you can see above, the app.UseStaticFiles() method adds StaticFiles middleware into the request pipeline. The UseStaticFiles is an extension method included in the StaticFiles middleware so that we can easily configure it.

## Set Default File

## call UseDefaultFiles() method before UseStaticFiles() in the Configure method as shown below.

**Serve static files from different folder than wwwroot folder in ASP.NET Core**

You can configure middleware to serve static files from other folders along with default web root folder wwwroot.

configure the StaticFiles middleware in the Configure() method of Startup class as shown below.

app.UseStaticFiles(new StaticFileOptions() {

FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "Content")),

RequestPath = new PathString("/Admin")

});

# **NET Core Application Types**

We can create two types of applications in .NET Core.

1. **Portable Application**

Portable applications are applications which expect .NET Core runtime on the deployment machines. It cannot be run on a machine which does not have .NET Core runtime installed

1. **Self-contained application**

Self-contained applications are applications which include .NET Core runtime when we publish it. It can run on a machine which does not have .NET Core runtime installed.

IOC Container

# Logging

ASP.NET Core has built-in support for logging APIs that is able to work with various logging providers. Using these built-in providers, we can send application logs to one or more destinations and also, we can plug in third party logging framework.

version (.net core 2.0) of framework, Logging is added into DI system by default. Also if we want to add any configuration or add any third party provider, configuration is done under main function (program.cs file) instead of startup class.

there are two important building blocks for implementing logging in a .NET Core based application:

1. Logging API
2. Logging Providers

## Logging API

Microsoft provides logging API as an extension in the wrapper [Microsoft.Extensions.Logging](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging) which comes as a NuGet package.

Microsoft.Extensions.Logging includes the necessary classes and interfaces for logging. The most important are the ILogger, ILoggerFactory, ILoggerProvider interfaces and the LoggerFactory class.

## ILoggerProvider

The ILoggerProvider manages and creates an appropriate logger, specified by the logging category.

We can create our own logging provider by implementing the ILoggerProvider interface.

### **ILogger**

The ILogger interface includes methods for logging to the underlying storage. There are many extension methods which make logging easy. Visit [ILogger](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.ilogger) for more information.

## Logging Providers

A logging provider displays or stores logs to a particular medium such as a console, a debugging event, an event log, a trace listener, and others. Microsoft provides various logging providers as NuGet packages.

The following table lists important logging providers.

| Logging Provider's NuGet Package | Output Target |
| --- | --- |
| [Microsoft.Extensions.Logging.Console](https://www.nuget.org/packages/Microsoft.Extensions.Logging.Console/) | Console |
| [Microsoft.Extensions.Logging.AzureAppServices](https://www.nuget.org/packages/Microsoft.Extensions.Logging.AzureAppServices/) | Azure App Services 'Diagnostics logs' and 'Log stream' features |
| [Microsoft.Extensions.Logging.Debug](https://www.nuget.org/packages/Microsoft.Extensions.Logging.Debug/) | Debugger Monitor |
| [Microsoft.Extensions.Logging.EventLog](https://www.nuget.org/packages/Microsoft.Extensions.Logging.EventLog/) | Windows Event Log |
| [Microsoft.Extensions.Logging.EventSource](https://www.nuget.org/packages/Microsoft.Extensions.Logging.EventSource/) | EventSource/EventListener |
| [Microsoft.Extensions.Logging.TraceSource](https://www.nuget.org/packages/Microsoft.Extensions.Logging.TraceSource/) | Trace Listener |

Microsoft has also collaborated with various logging framework teams (including third parties like NLog, Serilog, Loggr, Log4Net, and others) to extend the list of providers compatible with Microsoft.Extensions.Logging. The following are some thrid-party logging providers:

| Logging Provider | Description |
| --- | --- |
| [elmah.io](https://github.com/elmahio/Elmah.Io.Extensions.Logging) | Provider for the Elmah.Io service |
| [Loggr](https://github.com/imobile3/Loggr.Extensions.Logging) | Provider for the Logger service |
| [NLog](https://github.com/NLog/NLog.Extensions.Logging) | Provider for the NLog library |
| [Serilog](https://github.com/serilog/serilog-framework-logging) | Provider for the Serilog library |

## **Using Serilog MSSQL Sink**

We can add these sinks by using either NuGet package manager or .NET CLI using Package Manager.

1. PM> Install-Package Serilog.Sinks.MSSqlServer

Using .NET CLI:

1. > dotnet add package Serilog.Sinks.MSSqlServer

o configure this sink in C# code, we need to call WriteTo.MSSQLServer method during logger configuration. The logger configuration needs to be written in a program.Main method. There are two minimum configuration requirements:

1. var connectionString = @"Data Source=(local); Initial Catalog=Test;User ID=sa;Password=Passwd@12;";
2. var tableName = "Logs";
4. var columnOption = **new** ColumnOptions();
5. columnOption.Store.Remove(StandardColumn.MessageTemplate);
7. Log.Logger = **new** LoggerConfiguration()
8. .MinimumLevel.Information()
9. .MinimumLevel.Override("SerilogDemo", LogEventLevel.Information)
10. .WriteTo.MSSqlServer(connectionString, tableName, columnOptions: columnOption)
11. .CreateLogger();

## **How to add providers**

To enable the service of logging, we need to add the provider extension method in ConfigureLogging method of instance of WebHostBuilder class under main function. A logging provider takes a message which we create with ILogger object and display or store them. To use provider, we need to call "Add<Provider Name>" extension method under main function.

1. .ConfigureLogging((hostingContext, logging) =>
2. {
3. logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
4. logging.AddConsole();
5. logging.AddDebug();

## Store Logs in a Text File

To store logs in a file, install the NuGet package [**Serilog.Extensions.Logging.File**](https://github.com/serilog/serilog-extensions-logging-file)

Serillog includes an extension method for **ILoggerFactory**

Startup.cs file and add the ILoggerFactory parameter in the Configure() method. Then, call the AddFile() extension method to add Serillog file provider, as shown below. ASP.NET Core dependency injection will automatically pass an instance of the LoggerFactory for this parameter.

loggerFactory.AddFile("Logs/mylog-{Date}.txt");

## Log Levels

Log levels indicate the importance or severity of log messages. Built-in log providers include extension methods to indicate log levels.

| Log Level | Severity | Extension Method | Description |
| --- | --- | --- | --- |
| Trace | 0 | LogTrace() | Logs messages only for tracing purposes for the developers. |
| Debug | 1 | LogDebug() | Logs messages for short-term debugging purposes. |
| Information | 2 | LogInformation() | Logs messages for the flow of the application. |
| Warning | 3 | LogWarning() | Logs messages for abnormal or unexpected events in the application flow. |
| Error | 4 | LogError() | Logs error messages. |
| Critical | 5 | LogCritical() | Logs failures messages that require immediate attention. |

## Create Logs in the Controller

## Create Logs in the Controller

public class HomeController : Controller

{

private readonly ILogger \_logger;

public HomeController(ILogger<HomeController> logger){

\_logger = logger;

}

public IActionResult Index()

{

\_logger.LogInformation("Log message in the Index() method");

return View();

}

the ILogger<HomeController> parameter should bec included in the constructor. ASP.NET Core DI will pass the ILogger instance, which can be used to log in the Index() and About() action methods.

Id,page

The same can be achieved by passing ILoggerFactory in the constructor.

public HomeController(ILoggerFactory logFactory)

{

\_logger = logFactory.CreateLogger<HomeController>();

}

<https://www.c-sharpcorner.com/article/logging-framework-in-asp-net-core-2-0/>

<https://www.c-sharpcorner.com/article/file-logging-and-ms-sql-logging-using-serilog-with-asp-net-core-2-0/>