**Dot Net Document:**

**1. What’s .Net**

**.NET** is a software framework that is designed and developed by Microsoft and .NET is an open-source and cross-platform development platform for building many types of applications like desktop, web, cloud, mobile, gaming, IoT, and AI apps. .Net framework supports more than 60 programming language to develop the software applications

**2.What’s ASP.Net**

ASP.NET is a set of libraries and tools to build web applications including front-end websites, APIs, and Microservices.

**3.What’s C#.Net:**

C# (C sharp): A modern object-oriented programming language that belongs to the C language family. C# enables developers to build many types of secure and robust applications that run in . NET.

**4.Output**

To output values or print text in C#, you can use the WriteLine() method:

**5.Comments**

Multi-line comments start with /\* and ends with \*/.

Single-line comments start with two forward slashes (//).

**6.Variables**

Variables are containers for storing data values.

In C#, there are different **types** of variables (defined with different keywords), for example:

* int - stores integers (whole numbers), without decimals, such as 123 or -123
* double - stores floating point numbers, with decimals, such as 19.99 or -19.99
* char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* string - stores text, such as "Hello World". String values are surrounded by double quotes
* bool - stores values with two states: true or false

**7.Data Types**

A data type specifies the size and type of variable values. It is important to use the correct data type for the corresponding variable

**8.Type Casting**

Type casting is when you assign a value of one data type to another type.

**9.User Input**

The Console.ReadLine() method returns a string. Therefore, you cannot get information from another data type, such as int

**10.Operators**

Operators are used to perform operations on variables and values. Ex: +,-, \*, / , <=, >=, ==, &&, ||

**11.If...Else**

C# supports the usual logical conditions from mathematics:

* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false

**12.Methods**

A **method** is a block of code which only runs when it is called. You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

**13.Method Parameters**

Information can be passed to methods as parameter. Parameters act as variables inside the method.

They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

static void MyMethod(string fname)

**14.Strings(Length, Upper, Lower, Concat, Interpolation, Substring, IndexOf, Escape character, +, Replace)**

Strings are used for storing text.

A string variable contains a collection of characters surrounded by double quotes

## **15.String Interpolation**

Another option of string concatenation, is **string interpolation**, which substitutes values of variables into placeholders in a string. Note that you do not have to worry about spaces, like with concatenation

**16.Booleans**

A boolean type is declared with the bool keyword and can only take the values true or false

**17.Switch (default, goto case) – DEFINE DAYS**

Use the switch statement to select one of many code blocks to be executed.

* The switch expression is evaluated once
* The value of the expression is compared with the values of each case
* If there is a match, the associated block of code is executed
* The break and default keywords will be described later in this chapter

**18.Arrays(string arr, Int arr, foreach, sort, replace arr element, Linq – Min, max, sum)**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**

**19.For Loop**

Loop can be used to execute the code multiple times as how many times you want.

**20.Foreach:**

There is also a foreach loop, which is used exclusively to loop through elements in most of the collections and iteration can be defined based on the length of the collection object

**21.While/Do Loop**

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

The while loop loops through a block of code as long as a specified condition is True:

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

**22.Break/Continue**

It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

**23.Method Overloading**

With**method overloading**, multiple methods can have the same name with different parameters. We can also call this as compile time polymorphism.

**24.Classes/Objects**

A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods(member function which defines actions) into a single unit.

**25.Class Members**

Methods, Variables which are declared inside class

**26.Access Modifiers(Private, Public, Protected, Internal - Assemblies)**

Access modifier is used to define the scope of the variable

|  |  |
| --- | --- |
| public | The code is accessible for all classes |
| private | The code is only accessible within the same class |
| protected | The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about [inheritance](https://www.w3schools.com/cs/cs_inheritance.asp) in a later chapter |
| internal | The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter |

**27.Inheritence**

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in C# by which one class is allowed to inherit the features(fields and methods) of another class.

**28.Single:**

In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.

**29.MultiLevel A, B, C:**

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C

**30.Multiple Inheritence:**

In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that **C# does not support multiple inheritance** with classes. In C#, we can achieve multiple inheritance only through Interfaces.

**31.Hybrid Inheritence A, B, ABC, ABD**

It is a mix of two or more of the above types of inheritance. Since C# doesn’t support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In C#, we can achieve hybrid inheritance only through Interfaces.

**32.Hierarchical Inheritance - A, AB, AC**

 In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass.

**33.Method overriding vs Method overloading**

Method Overriding in C# is similar to the [**virtual function in C++**](https://www.geeksforgeeks.org/virtual-function-cpp/). Method Overriding is a technique that allows the invoking of functions from another class (base class) in the derived class. Creating a method in the derived class with the same signature as a method in the base class is called as method overriding.

**34.Base keyword**

This is used to access members of the base class from derived class. It basically used to access constructors and methods or functions of the base class. The base keyword cannot use within a static method. Base keyword specifies which constructor of the base class should be invoked while creating the instances of the derived class.

**35.Assemblies(Private, Public, Satellite – Resource):**

An assembly is **a file that is automatically generated by the compiler upon successful compilation of every .** **NET application**. It can be either a Dynamic Link Library or an executable file(dll or exe)

**36.Private Assembly**

A private assembly can be used by only a single application and not accessible for other applications. The dll or exe is generally stored in the application root folder.

**37.Public / Shared Assembly**

Public / Shared Assemblies are the assemblies that are accessible globally/shared across the machine to all the applications which are stored inside Global Assembly Cache (GAC), these can generally be found in the operating system folder like follows C:WindowsAssembly. For using the shared assemblies you need to register the assembly with a strong name in the GAC using gacutil.exe.

most common assemblies contained in the Global Assembly Cache,

**38.Satellite Assembly**

Satellite assemblies are useful while creating multilingual applications. Using atellite assemblies, localizable resources can be placed for different languages in different assemblies.

**Classes:  
39.Abstract class**

Abstraction in C# is the process to hide the internal details and show only the functionality. The abstract modifier indicates the incomplete implementation. The keyword abstract is used before the class or method to declare the class or method as abstract. Also, the abstract modifier can be used with indexers, events, and properties.

**40.Sealed class – Can’t inherit for security purpose**

Sealed classes are used to restrict the users from inheriting the class. A class can be sealed by using the **sealed** keyword. The keyword tells the compiler that the class is sealed, and therefore, cannot be extended. No class can be derived from a sealed class.

**41.Static class – This, can have static and non-static methods**

In C#, one is allowed to create a static class, by using *static*keyword. A static class can only contain static data members, static methods, and a static constructor.It is not allowed to create objects of the static class. Static classes are [**sealed**](https://www.geeksforgeeks.org/c-sealed-class/), means ***you cannot inherit a static class from another class***.

**42.Partial class – Logic can be splitted then can be added as a single class**

A partial class is a special feature of C#. It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a ***partial***keyword. This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.

**43.Abstraction**

Abstraction is hiding the implementation of the methods/class to the consumer and provide the needed information only to the consumer.

**44.Encapsulation**

Encapsulation is data hiding(information hiding) and it can be achieved with the help of access modifiers.

**45.Properties**

A property is like a combination of a variable and a method, and it has two methods: a get and a set method

The get method returns the value of the variable name.

The set method assigns a value to the name variable.

**46.Polymorphism.**

Polymorphism, in C#, is the ability of objects of different types to provide a unique interface for different implementations of methods.

Early binding is the compile time Polymorphism also it can be referred as method overloading. Late binding is the run time Polymorphism and it can be referred as method Overriding

**47.Interface:**

An interface has the only method signatures and definition (with empty bodies). Interfaces cannot be used to create objects. It’s used to achieve security - hide certain details and only show the important details of an object (interface).

**48.Constructors**

A constructor is a special method of the class which gets automatically invoked whenever an instance of the class is created. Like methods, a constructor also contains the collection of instructions that are executed at the time of Object creation. It is used to assign initial values to the data members of the same class. 

**49.Default Constructor**

A constructor with no parameters is called a default constructor. A default constructor has every instance of the class to be initialized to the same values. The default constructor initializes all numeric fields to zero and all string and object fields to null inside a class

**50.Parameterized Constructor**

A constructor having at least one parameter is called as parameterized constructor. It can initialize each instance of the class to different values.

**51.Copy Constructor**

This constructor creates an object by copying variables from another object. Its main use is to initialize a new instance to the values of an existing instance.

**52.Private Constructor**

If a constructor is created with private specifier is known as Private Constructor. It is not possible for other classes to derive from this class and also it’s not possible to create an instance of this class.

**53.Static Constructor**

Static Constructor has to be invoked only once in the class and it has been invoked during the creation of the first reference to a static member in the class. A static constructor is initialized static fields or data of the class and to be executed only once

**54.Model POCO**

A Plain Old CLR Objects (POCO) is a class, which doesn't depend on any framework-specific base class. It is like any other normal .NET class. Due to this, they are called Plain Old CLR Objects.

**55.Value type**

A data type is a value type if it holds a data value within its own memory space. It means the variables of these data types directly contain values.

**56.Reference type**

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

**57.Collections**

Collections standardize the way of which the objects are handled by your program. In other words, it contains a set of classes to contain elements in a generalized manner. With the help of collections, the user can perform several operations on objects like the store, update, delete, retrieve, search, sort etc

**58.Array**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

**58.Dictionary**

Dictionary is a generic collection which is generally used to store key/value pairs. The working of Dictionary is quite similar to the [non-generic hashtable](https://www.geeksforgeeks.org/c-sharp-hashtable-with-examples/).

The advantage of Dictionary is, it is generic type. Dictionary is defined under System.Collection.Generic namespace. It is dynamic in nature means the size of the dictionary is grows according to the need.

**59.List**

**List class** represents the list of objects which can be accessed by index. It comes under the **System.Collection.Generic** namespace. List class can be used to create a collection of different types like integers, strings etc. List<T> class also provides the methods to search, sort, and manipulate lists.

**60.SortedList**

SortedList class is a collection of **(key, value)** pairs which are sorted according to keys. Those pairs can be accessible by key and as well as by index(zero-based indexing). This comes under **System.Collections** namespace.

**61.Enums – To maintain the constant values  
Enumeration (or enum)** is a [value data type](https://www.geeksforgeeks.org/c-data-types-2/) in C#. It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain

**62.Files – Create, Write, Delete**

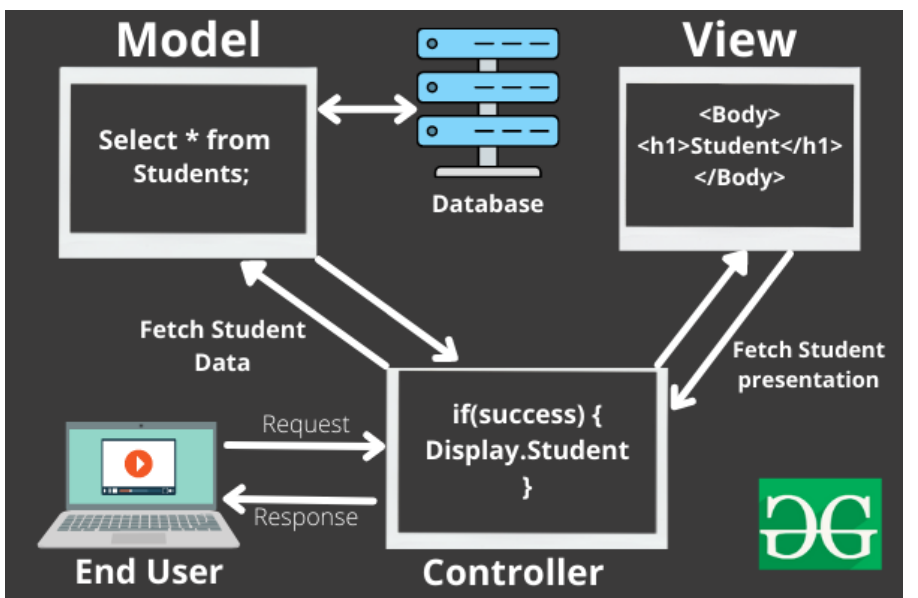
Generally, the file is used to store the data. The term File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc. There are two basic operation which is mostly used in file handling is reading and writing of the file. The file becomes stream when we open the file for writing and reading. A stream is a sequence of bytes which is used for communication

**63.Exceptions – throw new exception**

An exception is defined as an event that occurs during the execution of a program that is unexpected by the program code. The actions to be performed in case of occurrence of an exception is not known to the program. In such a case, we create an exception object and call the exception handler code. The execution of an exception handler so that the program code does not crash is called exception handling.

**MVC Framework Introduction**

The [**Model-View-Controller (MVC)**](https://www.geeksforgeeks.org/mvc-design-pattern/) framework is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**. Each architectural component is built to handle specific development aspects of an application



**Features of MVC :**

* It provides a **clear separation** of business logic, Ul logic, and input logic.
* It supports **Test Driven Development (TDD).**
* It offers full control over your HTML and URLs which makes it easy to design web application architecture.

**Disadvantages of MVC:**

* Increased complexity and Inefficiency of data
* It is not suitable for building small applications.

**MVC Fundamentals:**

// Add services to the container.

builder.Services.AddRazorPages();

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthorization();

app.MapDefaultControllerRoute();

app.MapRazorPages();

**Controller:**

The controller is the component that enables the interconnection between the views and the model so it acts as an intermediary.  It process all the business logic and incoming requests, manipulate data using the **Model**component and interact with the **View**to render the final output.

**View:**

The **View**component is used for all the UI logic of the application. It generates a user interface for the user. Views are created by the data which is collected by the model component but these data aren’t taken directly but through the controller.

**Model:**

The **Model**component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.

**Razer View:**

Razor View engine is a markup syntax which helps us to write HTML and server-side code in web pages using C# or VB.NET.

Razor is a templating engine and ASP.NET MVC has implemented a view engine which allows us to use Razor inside of an MVC application to produce HTML. However, Razor does not have any ties with ASP.NET MVC.

**Explanation about IIS Express**

@{

var price = 101;

}

@{

if(price == 100)

{

<p>It's hundred</p>

}

else

{

<p>It's not hundred</p>

}

}

**ASP.NET MVC Core Action Methods**

Every public method of the controller is ActionMethod except methods marked as [NonAction]. Action methods are similar to normal methods however ActionMethod has limitations as

* Action method must be public, it can not be private.
* Action method can not be static or extension method.
* Action method can not be getter or setter.

[ActionName("GetProductNameByID")]

public string GetProductName(int ProductID)

{

return "Product name is ABC.";

}

[ActionName("GetProductNameByProductCode")]

public string GetProductName(string code)

{

return "Product name is ABC.";

}

**ActionResult**

MVC framework includes various Result classes, which can be returned from an action method. The result classes represent different types of responses, such as HTML, file, string, JSON, javascript, etc. The following table lists all the result classes available in ASP.NET MVC.

| Result Class | Description |
| --- | --- |
| ViewResult | Represents HTML and markup. |
| EmptyResult | Represents No response. |
| ContentResult | Represents string literal. |
| FileContentResult/ FilePathResult/ FileStreamResult | Represents the content of a file. |
| JavaScriptResult | Represent a JavaScript script. |
| JsonResult | Represent JSON that can be used in AJAX. |
| RedirectResult | Represents a redirection to a new URL. |
| RedirectToRouteResult | Represent another action of same or other controller. |
| PartialViewResult | Returns HTML from Partial view. |
| HttpUnauthorizedResult | Returns HTTP 403 status. |

**Routing in ASP.NET Core MVC**

Routing is the process through which the application matches an incoming URL path and executes the corresponding action methods. ASP.NET Core MVC uses a routing middleware to match the URLs of incoming requests and map them to specific action methods.

There are two types of routing for action methods:

* [Conventional Routing](https://code-maze.com/routing-asp-net-core-mvc/#conventionalrouting)
* [Attribute Routing](https://code-maze.com/routing-asp-net-core-mvc/#attributerouting)

**Conventional Routing**

Configure routing in Configure method in startup.cs class

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

endpoints.MapControllerRoute(

name : "employee",

pattern: "EmployeeDetails",

defaults: new { controller = "Employee", action = "Index" });

});

**Attribute Routing**

* By placing a route on the controller or the action method, we can make use of the Attribute Routing feature.
* Let’s modify the Configure() method in the startup.cs class and remove the default routes that we had defined earlier.

[Route("[controller]/[action]")]

[Route("[controller]")]

public class TestController : Controller

{

[Route("")] // Matches 'Test'

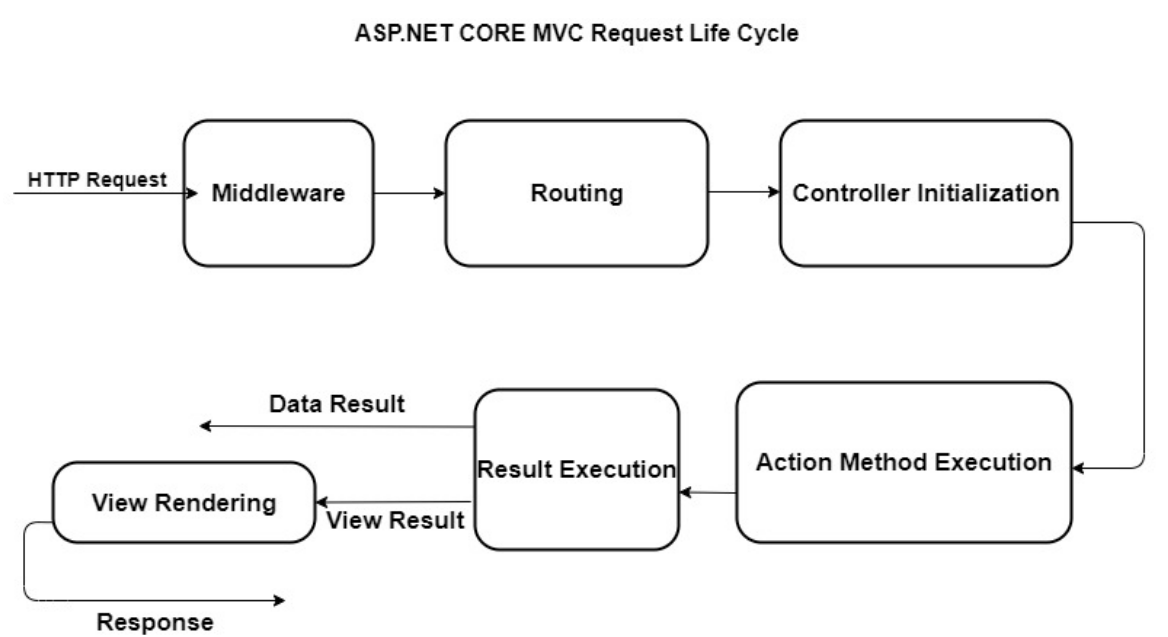
[Route("Index")] // Matches 'Test/Index'

public IActionResult Index()

}

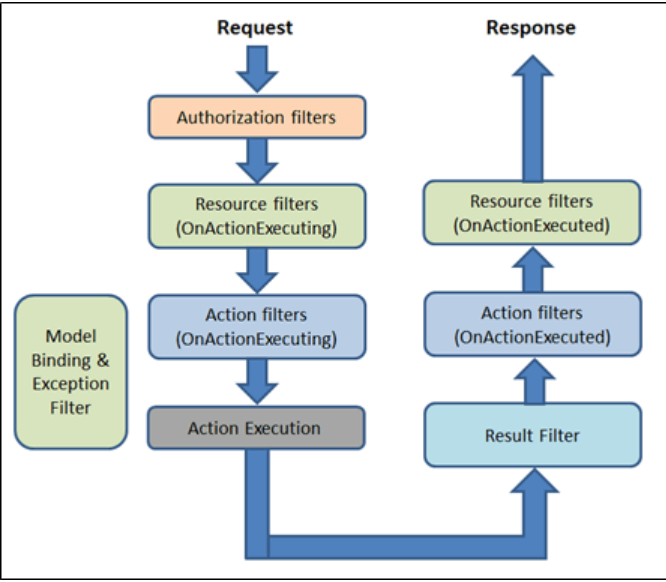
**ASP.NET Core MVC Request Life Cycle/Pipeline**

The ASP.NET Core MVC Request Life Cycle is a sequence of events, stages or components that interact with each other to process an HTTP request and generate a response that goes back to the client. In this article, we will discuss each and every stage of ASP.NET Core MVC Request Life Cycle in detail.



**Filters in MVC**

Every filter type is executed at a different stage in the filter pipeline. Following are the filter types.



***Authorization filters***  
The Authorization filters are executed first. This filter helps us to determine whether the user is authorized for the current request. It can short-circuit a pipeline if a user is unauthorized for the current request. We can also create custom authorization filter.

public class AuthorizeActionFilter : Attribute, IAuthorizationFilter

{

public void OnAuthorization(AuthorizationFilterContext context)

{

//Write you code here to authorize or unauthorize the user

}

}

***Resource filters***

The Resource filters handle the request after authorization. It can run the code before and after the rest of the filter is executed. This executes before the model binding happens. It can be used to implement caching.

public class CustomResourceFilterAttribute : Attribute, IResourceFilter

{

public void OnResourceExecuting(ResourceExecutingContext context)

{

context.Result = new ContentResult()

{

Content = "This is a Resource filter."

};

}

public void OnResourceExecuted(ResourceExecutedContext context)

{

}

}

***Action filters***  
The Action filters run the code immediately before and after the controller action method is called. It can be used to perform any action before or after execution of the controller action method. We can also manipulate the arguments passed into an action.

public class CustomActionFilter : IActionFilter

{

public void OnActionExecuting(ActionExecutingContext context)

{

// Executed before execution of an action method

}

public void OnActionExecuted(ActionExecutedContext context)

{

// Executed after execution of an action method

}

}

***Exception filters***  
The Exception filters are used to handle exception that occurred before anything written to the response body.

public class CustomExceptionFilter : Attribute, IExceptionFilter

{

public void OnException(ExceptionContext context)

{

context.Result = new ViewResult()

{

StatusCode = (int)HttpStatusCode.BadRequest,

ViewName = "Error"

};

context.ExceptionHandled = true;

}

}

***Result filters***  
The Result filters are used to run code before or after the execution of controller action results. They are executed only if the controller action method has been executed successfully.

public class CustomResultFilter : Attribute, IResultFilter

{

public void OnResultExecuting(ResultExecutingContext context)

{

context.Result = new ViewResult

{

ViewName = "Hello"

};

}

public void OnResultExecuted(ResultExecutedContext context)

{

}

}

**Data Annotation**

**Data Annotations** are nothing but certain validations that we put in our models to validate the input from the user. ASP.NET MVC provides a unique feature in which we can validate the models using the Data Annotation attribute. Import the following namespace to use data annotations in the application.

System.ComponentModel.DataAnnotations

It is very easy to use and the code becomes much cleaner as compared to normal ASP.NET validators.

Let us understand some of the validator attributes that we can use in MVC.

## **Types of Data Annotations in ASP.NET MVC**

### **Required**

This attribute specifies that the value is mandatory and cannot be skipped.

**Syntax**

[Required(ErrorMessage="Please enter name"),MaxLength(30)]

### **DataType**

This attribute is used to specify the datatype of the model.

**Syntax**

[DataType(DataType.Text)]

### **Range**

Using this attribute we can set a range between two numbers.

**Syntax**

[Range(100,500,ErrorMessage="Please enter correct value")]

### **StringLength**

Using this attribute we can specify maximum and minimum length of the property.

**Syntax**

[StringLength(30,ErrorMessage="Do not enter more than 30 characters")]

### **DisplayName**

Using this attribute we can specify property name to be displayed on view.

**Syntax**

[Display(Name="Student Name")]

### **MaxLength**

Using this attribute we can specify maximum length of property.

**Syntax**

[MaxLength(3)]

### **Bind**

This attribute specifies fields to include or exclude for model binding.

**Syntax**

[Bind(Exclude = "StudentID")]

### **DisplayFormat**

This attribute allows us to set date in the format specified as per the attribute.

**Syntax**

[DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}")]

### **RegularExpression**

We can set a regex pattern for the property. For ex: Email ID.

**Syntax**

[RegularExpression(@"^\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$", ErrorMessage = "Email is not valid.")]

**Ex:**

public class UserRegistration

{

[Required(ErrorMessage = "Please enter your first name")]

[MinLength(2, ErrorMessage = "Please enter atleast two characters")]

[MaxLength(50, ErrorMessage = "Please enter upto 50 characters")]

[Display(Name = "First Name")]

public string FirstName { get; set; }

[Required(ErrorMessage = "Please enter your last name")]

[MinLength(2, ErrorMessage = "Please enter atleast two characters")]

[MaxLength(50, ErrorMessage = "Please enter upto 50 characters")]

[Display(Name = "Last Name")]

public string LastName { get; set; }

[Required(ErrorMessage = "Please Enter your age")]

[DataType(DataType.PostalCode)]

[MaxLength(3, ErrorMessage = "Please enter valid age")]

[Display(Name = "Age")]

public int Age { get; set; }

[Required(ErrorMessage = "Please provide your email id")]

[MinLength(10, ErrorMessage = "Please enter atleast two characters")]

[MaxLength(70, ErrorMessage = "Please enter upto 50 characters")]

[DataType(DataType.EmailAddress)]

[Display(Name = "Email Address")]

public string EmailId { get; set; }

[Required(ErrorMessage = "Please provide your mobile number")]

[MaxLength(10, ErrorMessage = "Please enter upto 10 digits")]

[DataType(DataType.PhoneNumber)]

[Display(Name = "Mobile Number")]

public long MobileNumber { get; set; }

}

**Strongly typed view**

The view which binds to a specific type of ViewModel is called as **Strongly Typed View**. By specifying the model, the Visual studio provides the intellisense and compile time checking of type.

**Advantages of Strongly Typed View**

1. IntelliSense Help
2. Compile time error checking
3. You do not have to cast between types

*Since there is only one Model Property, you can have only one ViewModel per View.*

**Ex:**

@model UserManagement.Models.UserRegistration

@{

ViewData["Title"] = "UserRegistration";

}

<style>

.custom-width

{

width: 600px;

background-color : red;

}

</style>

<h4>UserRegistration</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="UserRegistration">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="FirstName" class="control-label"></**label**>

<**input** **asp-for**="FirstName" class="form-control custom-width" />

<**span** **asp-validation-for**="FirstName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="LastName" class="control-label"></**label**>

<**input** **asp-for**="LastName" class="form-control" />

<**span** **asp-validation-for**="LastName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Age" class="control-label"></**label**>

<**input** **asp-for**="Age" class="form-control" />

<**span** **asp-validation-for**="Age" class="text-danger"></**span**>

</div>

**ASP.NET MVC Scaffolding**

Scaffolding is used to define the code-generation framework used in web applications. It uses T4 templates to generate basic controllers and views for the models. It generates instances for the mapped domain model and code for all CRUD operations. It also reduces the amount of time for developing a standard data operation in the application.

Basically, it is an automated code generation framework, it generates code for CRUD operations based on the provided domain model classes

**Authentication And Authorization In ASP.NET Core MVC**

Authentication is the process to validate an anonymous user based on some credentials and Authorization process happens just after that and grants resources to this validated user. So, we can say, it's two-step validating process before providing the access of the resources or data.

**CookieAuthenticationDefaults:**

CookieAuthenticationDefaults. **AuthenticationScheme** provides “Cookies” for the scheme. CookieAuthenticationOptions class is used to configure the authentication provider options.

Add below line in **ConfigureServices** method

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie();

1. app.UseAuthentication();

We can implement Authentication through Login feature. In most of the applications today, Authorization is decided internally based on your role.

**LogOut Method in AccountController.cs**

[HttpPost]

public IActionResult Logout()

{

var login = HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

return RedirectToAction("Login");

}

**View code for Logout:**

**Index.cshtml of account controller.cs**

<div class="container">

<div class="row">

<div class="col-md-12">

<h2><strong>Login Page </strong></h2><br /><br />

Hello @User.Identity.Name !, Role @User.FindFirst(claim=>claim.Type==System.Security.Claims.ClaimTypes.Role)?.Value

<**a** **asp-action**="logout" **asp-controller**="account">

Logout

</**a**>

<br />

<br />

<h4>Admin role user can only access this page!!</h4>

</div>

</div>

</div>

**Login Post Method in Account Controller.cs:**

The **[ClaimsIdentity](https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claimsidentity?view=net-6.0)** class is a concrete implementation of a claims-based identity; that is, an identity described by a collection of claims. A claim is a statement about an entity made by an issuer that describes a property, right, or some other quality of that entity. Such an entity is said to be the subject of the claim. A claim is represented by the [Claim](https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claim?view=net-6.0) class. The claims contained in a [ClaimsIdentity](https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claimsidentity?view=net-6.0) describe the entity that the corresponding identity represents, and can be used to make authorization and authentication decisions.

[HttpPost]

public IActionResult Login(UserLogin userInfo)

{

var userName = userInfo.UserName;

var password = userInfo.Password;

if (!string.IsNullOrEmpty(userName) && string.IsNullOrEmpty(password))

{

return RedirectToAction("Login");

}

//Check the user name and password

//Here can be implemented checking logic from the database

ClaimsIdentity identity = null;

bool isAuthenticated = false;

if (userName == "Admin" && password == "password")

{

//Create the identity for the user

identity = new ClaimsIdentity(new[] {

new Claim(ClaimTypes.Name, userName),

new Claim(ClaimTypes.Role, "Admin")

}, CookieAuthenticationDefaults.AuthenticationScheme);

isAuthenticated = true;

}

if (userName == "Reception" && password == "password")

{

//Create the identity for the user

identity = new ClaimsIdentity(new[] {

new Claim(ClaimTypes.Name, userName),

new Claim(ClaimTypes.Role, "User")

}, CookieAuthenticationDefaults.AuthenticationScheme);

isAuthenticated = true;

}

if (isAuthenticated)

{

var principal = new ClaimsPrincipal(identity);

var login = HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal);

return RedirectToAction("Index", "Home");

}

ViewBag.ErrorMessage = "Please enter valid user name and password";

return View();

}

**Create CustomAuthenticationFilter:**

public class CustomAuthenticationFilter : Attribute, IAuthorizationFilter

{

public void OnAuthorization(AuthorizationFilterContext context)

{

if (context.ActionDescriptor.EndpointMetadata.OfType<AllowAnonymousAttribute>().Any()) return;

var user = context.HttpContext.User;

var role = user.FindFirst(claim => claim.Type == System.Security.Claims.ClaimTypes.Role)?.Value;

if(role == "Admin")

{

if(context.ActionDescriptor.RouteValues["action"] == "Index" && context.ActionDescriptor.RouteValues["controller"] == "Home")

{

return;

}

else

{

context.Result = new RedirectToRouteResult(

new RouteValueDictionary

{

{ "controller", "Account" },

{ "action", "Login" }

});

}

}

else if (role == "User")

{

if (context.ActionDescriptor.RouteValues["action"] == "Privacy" && context.ActionDescriptor.RouteValues["controller"] == "Home")

{

return;

}

else

{

context.Result = new UnauthorizedResult();

}

}

else

{

context.Result = new RedirectToRouteResult(

new RouteValueDictionary

{

{ "controller", "Account" },

{ "action", "Login" }

});

}

}

}

**Use custom Authorization filter in Home controller:**

[CustomAuthenticationFilter]

public IActionResult Index()

{

return View();

}

[CustomAuthenticationFilter]

public IActionResult Privacy()

{

return View();

}

**AllowAnonymous:**

The AllowAnonymous attribute in MVC **is used to skip the authorization** which is enforced by Authorization Filter in MVC.

**Ex: [AllowAnonymous]**

**Register Filter Globally in startup.cs:**

Same way we can register all types of filters globally

services.AddMvc(options =>

{

options.Filters.Add(typeof(CustomAuthenticationFilter));

});

**Ref:** https://www.c-sharpcorner.com/article/authentication-and-authorization-in-asp-net-core-mvc-using-cookie/#:~:text=Authentication%20And%20Authorization%20In%20ASP.NET%20Core%20MVC%20Using%20Cookie,-Mukesh%20Kumar&text=Security%20is%20the%20main%20concern,think%20about%20Authentication%20and%20Authorization.

**State Management:**

State management means to preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless.

Cookie

Query string

Viewdata

Viewbag

Tempdata

Session

## **What is a cookie?**

A cookie is basically a physical, plain-text file stored by the client (usually a browser), tied to a specific website. The client will then allow this specific website to read the information stored in this file on subsequent requests, basically allowing the server (or even the client itself) to store information for later use.

// Set and Get cookies

CookieOptions cookieOptions = new CookieOptions();

cookieOptions.Expires = new DateTimeOffset(DateTime.Now.AddDays(7));

HttpContext.Response.Cookies.Append("first\_request", DateTime.Now.ToString(), cookieOptions);

var setCookieHeaders = HttpContext.Response.GetTypedHeaders().SetCookie;

// We assume only one cookie is found. You could loop over multiple matches instead.

// setCookieHeaders may be null if response doesn't contain set-cookie headers

var cookie = setCookieHeaders?.FirstOrDefault(x => x.Name == "first\_request");

## **ViewData:**

1. ViewData is a dictionary object that is derived from ViewDataDictionary class.
2. public ViewDataDictionary ViewData { get; set; }
3. ViewData is a property of ControllerBase class.
4. ViewData is used to pass data from controller to corresponding view.
5. It’s life lies only during the current request.
6. If redirection occurs then it’s value becomes null.
7. It’s required typecasting for getting data and check for null values to avoid error.

**ViewBag**

1. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
2. Basically it is a wrapper around the ViewData and also used to pass data from controller to corresponding view.
3. public Object ViewBag { get; }
4. ViewBag is a property of ControllerBase class.
5. It’s life also lies only during the current request.
6. If redirection occurs then it’s value becomes null.
7. It doesn’t required typecasting for getting data.

**TempData**

1. TempData is a dictionary object that is derived from TempDataDictionary class and stored in short lives session.
2. public TempDataDictionary TempData { get; set; }
3. TempData is a property of ControllerBase class.
4. TempData is used to pass data from current request to subsequent request (means redirecting from one page to another).
5. It’s life is very short and lies only till the target view is fully loaded.
6. It’s required typecasting for getting data and check for null values to avoid error.
7. It is used to store only one time messages like error messages, validation messages. To persist data with TempData refer this article: [Persisting Data with TempData](http://www.dotnettricks.com/learn/mvc/persisting-data-with-tempdata)

**Session**

**Packages Required**

We need to install the stable version of “Microsoft.AspNetCore.Session” from the NuGet Package Manager. Then only we can access the session state in ASP.NET Core. Click on the “Install” button.

1. HttpContext.Session.SetString(SessionName, "Jarvik");
2. HttpContext.Session.SetInt32(SessionAge, 24);
3. ViewBag.Name = HttpContext.Session.GetString(SessionName);
4. ViewBag.Age = HttpContext.Session.GetInt32(SessionAge);

**Transport data from controller to view**

@model Employee

<h2>Employee Detail:</h2>

<ul>

<li>Student Id: @Model.EmployeeId</li>

<li>Student Name: @Model. EmployeeName</li>

<li>Age: @Model.Age</li>

</ul>

**Middleware:**

A middleware is a piece of logic or code or a component, that can be injected in the request-response pipeline of a .net core application development services.

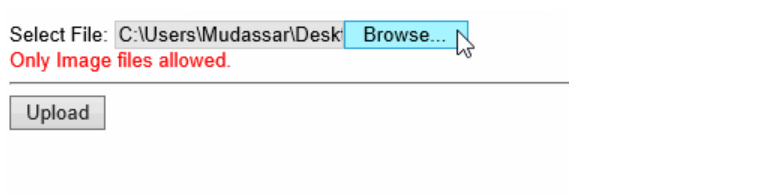
There will be multiple middleware in ASP.NET Core web application. It can be either framework provided middleware, added via NuGet or your own custom middleware.

You 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.

* **Static Files middleware** is to serve static files (js, CSS, images, etc) for a web app. **It is a terminal middleware**. So, any static files request would not go via subsequent middleware pipeline.
* **Routing middleware** is to add the endpoint information to the request object. This information is later used by other middleware.
* **CORS middleware** for applying CORS policies.
* **Authentication middleware** to check if the user is authenticated
* **Authorization middleware** to check if user is authorized to perform current operation.

**HTML Helpers:**

**File upload control in MVC**

****

**Action Methods:**

public IActionResult Index()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Index(List<IFormFile> files)

{

string path = @"D:\Aimore\Projects\MVCAppWithApi\MVCAppWithApi\";

long size = files.Sum(f => f.Length);

var filePaths = new List<string>();

foreach (var formFile in files)

{

if (formFile.Length > 0)

{

// full path to file in temp location

var filePath = path + formFile.FileName; //we are using Temp file name just for the example. Add your own file path.

filePaths.Add(filePath);

using (var stream = new FileStream(filePath, FileMode.Create))

{

await formFile.CopyToAsync(stream);

}

}

}

// process uploaded files

// Don't rely on or trust the FileName property without validation.

return Ok(new { count = files.Count, size, filePaths });

}

**View**

<**form** method="post" enctype="multipart/form-data" **asp-controller**="File" **asp-action**="Index">

<div class="form-group">

<div class="col-md-10">

<p>Upload one or more files using this form:</p>

<input type="file" name="files" multiple />

</div>

</div>

<div class="form-group">

<div class="col-md-10">

<input type="submit" value="Upload" />

</div>

</div>

</**form**>

**Radio Button & Drop down**

public class CreateUserVm

{

public string UserName { set; get; }

public IEnumerable<RoleVm> Roles { set; get; }

public int SelectedRole { set; get; }

}

public class RoleVm

{

public int Id { set; get; }

public string RoleName { set; get; }

}

**Action Method**

public IActionResult Index()

{

var vm = new CreateUserVm

{

Roles = new List<RoleVm>

{

new RoleVm {Id = 1, RoleName = "Admin"},

new RoleVm {Id = 2, RoleName = "Editor"},

new RoleVm {Id = 3, RoleName = "Reader"}

}

};

List<SelectListItem> cities = new()

{

new SelectListItem { Value = "1", Text = "Latur" },

new SelectListItem { Value = "2", Text = "Solapur" },

new SelectListItem { Value = "3", Text = "Nanded" },

new SelectListItem { Value = "4", Text = "Nashik" },

new SelectListItem { Value = "5", Text = "Nagpur" },

new SelectListItem { Value = "6", Text = "Kolhapur" },

new SelectListItem { Value = "7", Text = "Pune" },

new SelectListItem { Value = "8", Text = "Mumbai" },

new SelectListItem { Value = "9", Text = "Delhi" },

new SelectListItem { Value = "10", Text = "Noida" }

};

//assigning SelectListItem to view Bag

ViewBag.cities = cities;

return View(vm);

}

**View**

@model YourNamespaceHere.CreateUserVm

<form asp-action="Index" asp-controller="Home">

<label class="label">User name</label>

<div class="col-md-10">

<input type="text" asp-for="UserName" />

</div>

<label class="label">Select a Role</label>

<div class="col-md-10">

@foreach (var item in Model.Roles)

{

<input asp-for="SelectedRole" type="radio" value="@item.Id" /> @item.RoleName

}

</div>

<div class="form-group">

<label class="control-label"> Select City</label>

<select name="products" class="form-control" asp-items="@ViewBag.cities"></select>

</div>

<input type="submit" />

</form>

**Image:**

1. <img src="~/images/@employee.ProfilePicture"
2. **class**="rounded-circle"
3. height="40" width="40"
4. asp-append-version="true" />

**Partial View:**

The primary difference between the two methods is that Partial generates the HTML from the View and returns it to the View to be incorporated into the page. RenderPartial, on the other hand, doesn't return anything and, instead, adds its HTML directly to the Response object's output.

@Html.Partial("TestPartial.cshtml");

@{

Html.RenderPartial("TestPartial.cshtml");

}

Partial vs RenderPartial

RenderPartial should give you better performance

# Difference between RDBMS and DBMS

**Database Management System (DBMS)** is a software that is used to define, create and maintain a database and provides controlled access to the data.

**Relational Database Management System (RDBMS)** is an advanced version of a DBMS. 

| DBMS | RDBMS |
| --- | --- |
| DBMS stores data as file. | RDBMS stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |

**SQL server**

Microsoft SQL Server is a relational database management system (RDBMS) that **supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments**

**DML**

DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.

SELECT – Retrieves data from a table  
INSERT -  Inserts data into a table  
UPDATE – Updates existing data into a table  
DELETE – Deletes all records from a table

**DDL**

DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.

CREATE – Creates objects in the database  
ALTER – Alters objects of the database  
DROP – Deletes objects of the database  
TRUNCATE – Deletes all records from a table and resets table identity to initial value.

**DCL**

DCL is abbreviation of **Data Control Language**. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

GRANT – Gives user's access privileges to database  
REVOKE – Withdraws user's access privileges to database given with the GRANT command

**TCL**

TCL is abbreviation of **Transactional Control Language**. It is used to manage different transactions occurring within a database.

COMMIT – Saves work done in transactions  
ROLLBACK – Restores database to original state since the last COMMIT command in transactions  
SAVE TRANSACTION – Sets a savepoint within a transaction

**What Is Normalization in SQL?**

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

**1st Normal Form (1NF)**

* A table is referred to as being in its First Normal Form if atomicity of the table is 1.
* Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.
* The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Second Normal Form (2NF)**

The first condition for the table to be in Second Normal Form is that the table has to be in First Normal Form. The table should not possess partial dependency. The partial dependency here means the proper subset of the candidate key should give a non-prime attribute.

**Third Normal Form (3NF)**

* The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form.
* The second condition is that there should be no transitive dependency for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) should not depend on other non-prime attributes in a table. Therefore, a transitive dependency is a functional dependency in which A → C (A determines C) indirectly, because of A → B and B → C (where it is not the case that B → A).

## **SQL Server Data Types**

### **String Data Types**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **Description** | **Max size** | **Storage** |
| char(n) | Fixed width character string | 8,000 characters | Defined width |
| varchar(n) | Variable width character string | 8,000 characters | 2 bytes + number of chars |
| varchar(max) | Variable width character string | 1,073,741,824 characters | 2 bytes + number of chars |
| Text | Variable width character string | 2GB of text data | 4 bytes + number of chars |
| Nchar | Fixed width Unicode string | 4,000 characters | Defined width x 2 |
| nvarchar | Variable width Unicode string | 4,000 characters |  |
| nvarchar(max) | Variable width Unicode string | 536,870,912 characters |  |
| Ntext | Variable width Unicode string | 2GB of text data |  |
| binary(n) | Fixed width binary string | 8,000 bytes |  |
| varbinary | Variable width binary string | 8,000 bytes |  |
| varbinary(max) | Variable width binary string | 2GB |  |
| Image | Variable width binary string | 2GB |  |

### **Numeric Data Types**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| Bit | Integer that can be 0, 1, or NULL |  |
| Tinyint | Allows whole numbers from 0 to 255 | 1 byte |
| smallint | Allows whole numbers between -32,768 and 32,767 | 2 bytes |
| int | Allows whole numbers between -2,147,483,648 and 2,147,483,647 | 4 bytes |
| bigint | Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 | 8 bytes |
| decimal(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| numeric(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| smallmoney | Monetary data from -214,748.3648 to 214,748.3647 | 4 bytes |
| money | Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 | 8 bytes |
| float(n) | Floating precision number data from -1.79E + 308 to 1.79E + 308.  The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53. | 4 or 8 bytes |
| real | Floating precision number data from -3.40E + 38 to 3.40E + 38 | 4 bytes |

### **Date and Time Data Types**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| datetime | From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds | 8 bytes |
| datetime2 | From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds | 6-8 bytes |
| smalldatetime | From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute | 4 bytes |
| date | Store a date only. From January 1, 0001 to December 31, 9999 | 3 bytes |
| time | Store a time only to an accuracy of 100 nanoseconds | 3-5 bytes |
| datetimeoffset | The same as datetime2 with the addition of a time zone offset | 8-10 bytes |
| timestamp | Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable |  |

### **Other Data Types**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| sql\_variant | Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp |
| uniqueidentifier | Stores a globally unique identifier (GUID) |
| xml | Stores XML formatted data. Maximum 2GB |
| cursor | Stores a reference to a cursor used for database operations |
| table | Stores a result-set for later processing |

## **CREATE TABLE**

The CREATE TABLE command creates a new table in the database.

The following SQL creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

### **Example**

CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);

## **ALTER TABLE**

The ALTER TABLE command adds, deletes, or modifies columns in a table.

The ALTER TABLE command also adds and deletes various constraints in a table.

The following SQL adds an "Email" column to the "Customers" table:

### **Example**

ALTER TABLE Customers  
ADD Email varchar(255);

## **DROP TABLE**

The DROP TABLE command deletes a table in the database.

The following SQL deletes the table "Shippers":

### **Example**

DROP TABLE Shippers;

## **Schema in SQL server**

A **schema** is a collection of database objects like tables, triggers, stored procedures, etc. A schema is connected with a user which is known as the schema owner. Database may have one or more schema.

[SQL Server](https://www.geeksforgeeks.org/sql-tutorial/#sql-server) have some built-in schema, for example: dbo, guest, sys, and INFORMATION\_SCHEMA.

dbo is default schema for a new database, owned by dbo user. While creating a new user with CREATE USER command, user will take dbo as its default schema.

CREATE SCHEMA statement used to create a new schema in current database.

**Syntax :**

CREATE SCHEMA schemaname

## **Stored Procedures in SQL Server**

**Stored Procedures** are created to perform one or more DML operations on Database. It is nothing but the group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not returns a value.

The most important part is parameters. Parameters are used to pass values to the Procedure. There are 3 different types of parameters, they are as follows: 

1. **IN:**   
   This is the Default Parameter for the procedure. It always receives the values from calling program.
2. **OUT:**   
   This parameter always sends the values to the calling program.
3. **IN OUT:**   
   This parameter performs both the operations. It Receives value from as well as sends the values to the calling program.

## **Types of Function**

## **System Defined Function**

These functions are defined by [SQL Server](https://www.dotnettricks.com/learn/sqlserver/introduction-to-sql-server) for a different purposes. The functions that are defined by the system are known as "system defined functions". In other words, all the built-in functions supported by the SQL server are referred to as system defined functions. Usage of the built-in functions saves much development time while performing certain tasks. These types schemaof functions generally work with the SQL select statement to calculate the values and the manipulated data.

We have two types of system defined functions in SQL Server

## **Scalar Function**

Scalar functions operate on a single value and return a single value. Below is the list of some useful SQL Server Scalar functions.

System Scalar Function

Scalar Function

Description

abs(-10.67)

This returns an absolute number of the given number means 10.67.

rand(10)

This will generate a random number of 10 characters.

round(17.56719,3)

This will round off the given number to 3 places of decimal meaning 17.567

upper('dotnet')

This will return the upper case of the given string meaning 'DOTNET'

lower('DOTNET')

This will returns the lower case of the given string means 'dotnet'

ltrim(' dotnet')

This will remove the spaces from the left-hand side of the 'dotnet' string.

convert(int, 15.56)

This will convert the given float value to integer means 15.

## **Aggregate Function**

Aggregate functions operate on a collection of values and return a single value. Below is the list of some useful SQL Server Aggregate functions.

System Aggregate Function

Aggregate Function

Description

max()

This returns the maximum value from a collection of values.

min()

This returns the minimum value from a collection of values.

avg()

This returns an average of all values in a collection.

count()

This returns no of counts from a collection of values.

## **User-Defined Function**

Functions that are created by the user or developer in the system database or a user-defined database are generally known as "user-defined functions". The user-defined functions may accept required parameters, perform certain actions, and return the processed data. These custom functions help us to simplify the overall database development by encapsulating the complex business logic and making it available for reuse whenever any similar functionality is required. The user-defined functions hold the code that is needed to query data a lot easier to write. It also improves query readability, accessibility, and functionality, as well as allows other developers to replicate the same procedures accordingly.

These functions are created by the user in the system database or in a user-defined database. We have three types of user-defined functions.

## **Scalar Function**

The user-defined scalar function also returns a single value as a result of actions performed by the function. We return any datatype value from a function.

--Create function to get emp full name

Create function fnGetEmpFullName

(

@FirstName varchar(50),

@LastName varchar(50)

)

returns varchar(101)

As

Begin return (Select @FirstName + ' '+ @LastName);

end

## **Inline Table-Valued Function**

The user-defined inline table-valued function returns a table variable as a result of actions performed by the function. The value of the table variable should be derived from a single SELECT statement.

--Create function to get employees

Create function fnGetEmployee()

returns Table

As

return (Select \* from Employee)

## **Multi-Statement Table-Valued Function**

A user-defined multi-statement table-valued function returns a table variable as a result of actions performed by the function. In this, a table variable must be explicitly declared and defined whose value can be derived from multiple SQL statements.

--Create function for EmpID,FirstName and Salary of Employee

Create function fnGetMulEmployee()

returns @Emp Table

(

EmpID int,

FirstName varchar(50),

Salary int

)

As

begin

Insert into @Emp Select e.EmpID,e.FirstName,e.Salary from Employee e;

--Now update salary of first employee

update @Emp set Salary=25000 where EmpID=1;

--It will update only in @Emp table not in Original Employee table

return

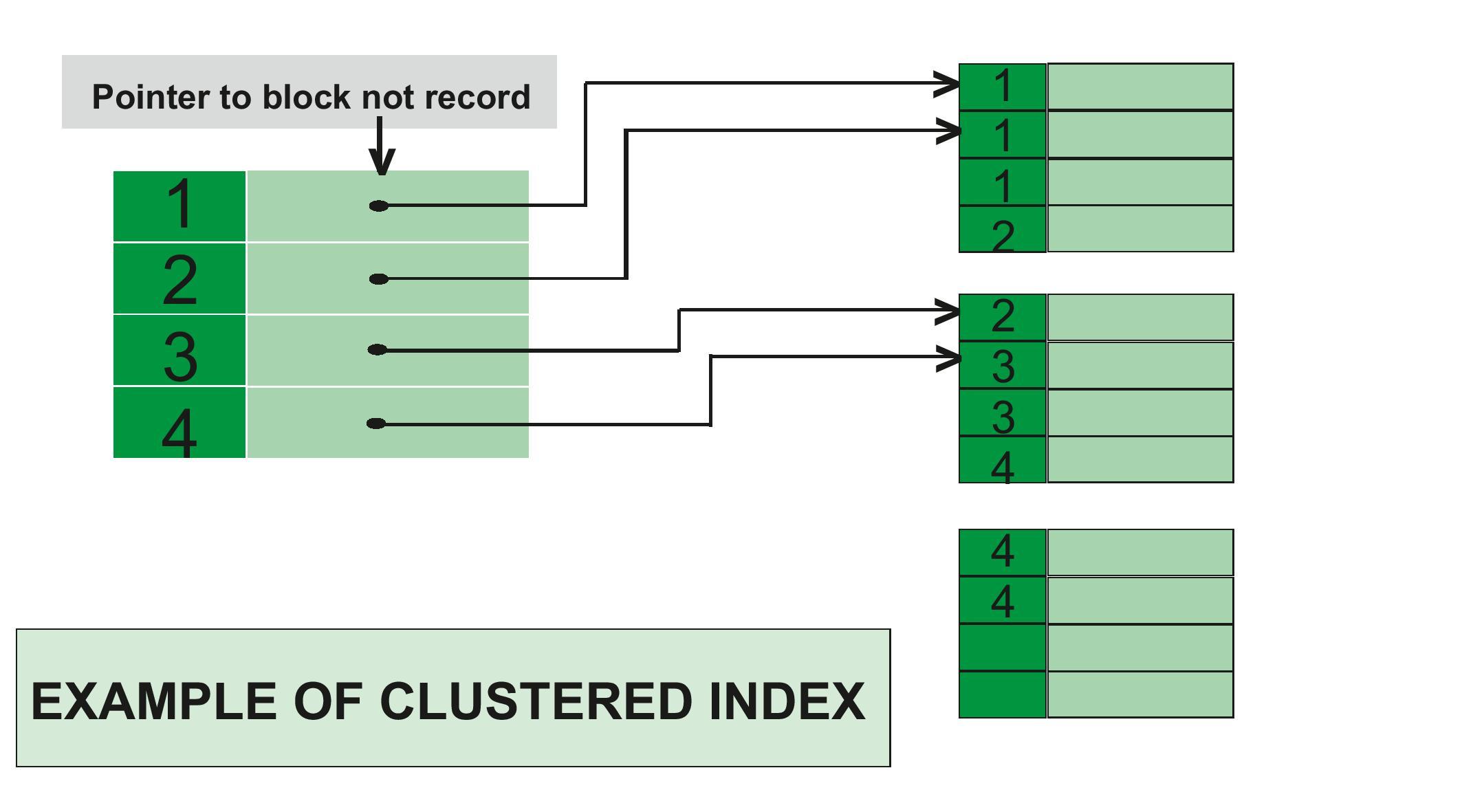
end

# Difference between Clustered and Non-clustered index

**1. Clustered Index :**   
Clustered index is created only when both the following conditions satisfy –

1. The data or file, that you are moving into secondary memory should be in sequential or sorted order.
2. There should be a key value, meaning it can not have repeated values.

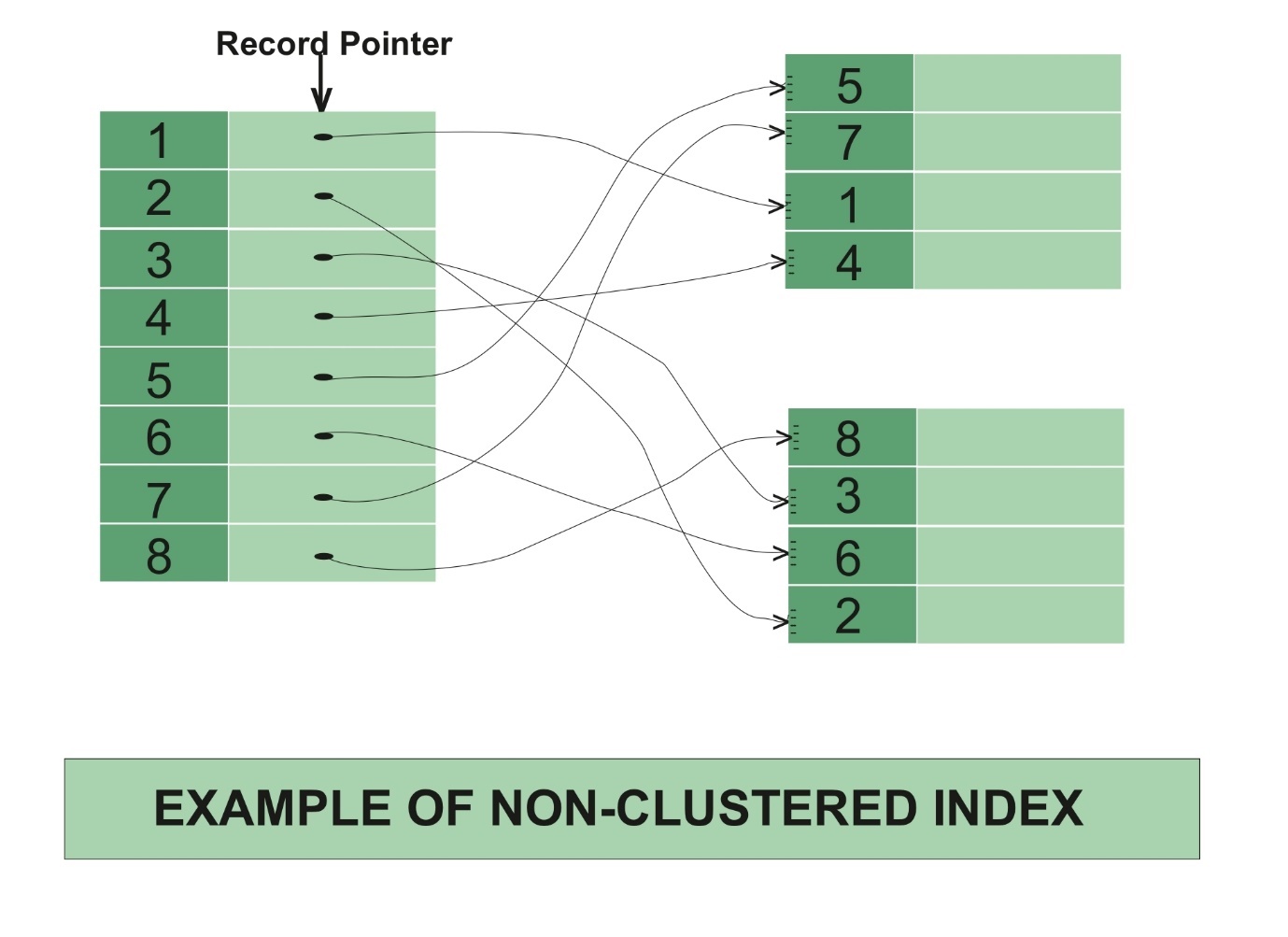
Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key. Clustered index is as same as dictionary where the data is arranged by alphabetical order.



**Ex: Primary Key**

**2. Non-clustered Index :**   
Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.



**Different between seek and scan**

An index scan or table scan is when SQL Server has to scan the data or index pages to find the appropriate records. **A scan is the opposite of a seek, where a seek uses the index to pinpoint the records that are needed to satisfy the query**.

**Difference between Clustered and Non-clustered index :**

| CLUSTERED INDEX | NON-CLUSTERED INDEX |
| --- | --- |
| Clustered index is faster. | Non-clustered index is slower. |
| Clustered index requires less memory for operations. | Non-Clustered index requires more memory for operations. |
| In clustered index, index is the main data. | In Non-Clustered index, index is the copy of data. |
| A table can have only one clustered index. | A table can have multiple non-clustered index. |
| Clustered index has inherent ability of storing data on the disk. | Non-Clustered index does not have inherent ability of storing data on the disk. |
| Clustered index store pointers to block not data. | Non-Clustered index store both value and a pointer to actual row that holds data. |
| In Clustered index leaf nodes are actual data itself. | In Non-Clustered index leaf nodes are not the actual data itself rather they only contains included columns. |
| In Clustered index, Clustered key defines order of data within table. | In Non-Clustered index, index key defines order of data within index. |
| A Clustered index is a type of index in which table records are physically reordered to match the index. | A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk. |
| The size of clustered index is large. | Size of non-clustered index is comparatively smaller. |
| Primary Keys of the table by default are clustered index. | Composite key when used with unique constraints of the table act as non-clustered index. |

# Difference between JOIN and UNION in SQL

**JOIN :** JOIN in SQL is used to combine data from many tables based on a matched condition between them. The data combined using JOIN statement results into new columns.

SELECT Boys.Name, Boys.Age, Girls.Address,

FROM Boys

INNER JOIN Girls

ON Boys.Rollno = Girls.Rollno;

* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

[**UNION**](https://www.geeksforgeeks.org/sql-union-clause/): UNION in SQL is used to combine the result-set of two or more SELECT statements. The data combined using UNION statement is into results into new distinct rows.

SELECT Name

FROM Boys

WHERE Rollno < 16

UNION

SELECT Name

FROM Girls

WHERE Rollno > 9

| JOIN | UNION |
| --- | --- |
| JOIN combines data from many tables based on a matched condition between them | SQL combines the result-set of two or more SELECT statements. |
| It combines data into new columns. | It combines data into new rows |
| Number of columns selected from each table may not be same. | Number of columns selected from each table should be same. |
| Datatypes of corresponding columns selected from each table can be different. | Datatypes of corresponding columns selected from each table should be same. |
| It may not return distinct columns. | It returns distinct rows. |

## **SQL Self Join**

A self join is a regular join, but the table is joined with itself.

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City  
FROM Customers A, Customers B  
WHERE A.CustomerID <> B.CustomerID  
AND A.City = B.City  
ORDER BY A.City;

## **The SQL GROUP BY Statement**

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;

## **The SQL HAVING Clause**

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

## **What is a SQL Server Trigger?**

A SQL Server trigger is a piece of procedural code, like a stored procedure which is only executed when a given event happens. There are different types of events that can fire a trigger. Just to name you a few, the insertion of rows in a table, a change in a table structure and even a user logging into a SQL Server instance.

There are three main characteristics that make triggers different than stored procedures:

* Triggers cannot be manually executed by the user.
* There is no chance for triggers to receive parameters.
* You cannot commit or rollback a transaction inside a trigger.

**CREATE TRIGGER production.trg\_product\_audit**

**ON production.products**

**AFTER INSERT, DELETE**

**AS**

**BEGIN**

**SET NOCOUNT ON;**

**INSERT INTO production.product\_audits(**

**product\_id,**

**product\_name,**

**brand\_id,**

**category\_id,**

**model\_year,**

**list\_price,**

**updated\_at,**

**operation**

**)**

**SELECT**

**i.product\_id,**

**product\_name,**

**brand\_id,**

**category\_id,**

**model\_year,**

**i.list\_price,**

**GETDATE(),**

**'INS'**

**FROM**

**inserted i**

**UNION ALL**

**SELECT**

**d.product\_id,**

**product\_name,**

**brand\_id,**

**category\_id,**

**model\_year,**

**d.list\_price,**

**GETDATE(),**

**'DEL'**

**FROM**

**deleted d;**

**END**

## **“Magic or Virtual” tables for triggers: INSERTED and DELETED**

SQL Server provides two virtual tables that are available specifically for triggers called INSERTED and DELETED tables. SQL Server uses these tables to capture the data of the modified row before and after the event occurs.

The following table shows the content of the INSERTED and DELETED tables before and after each event:

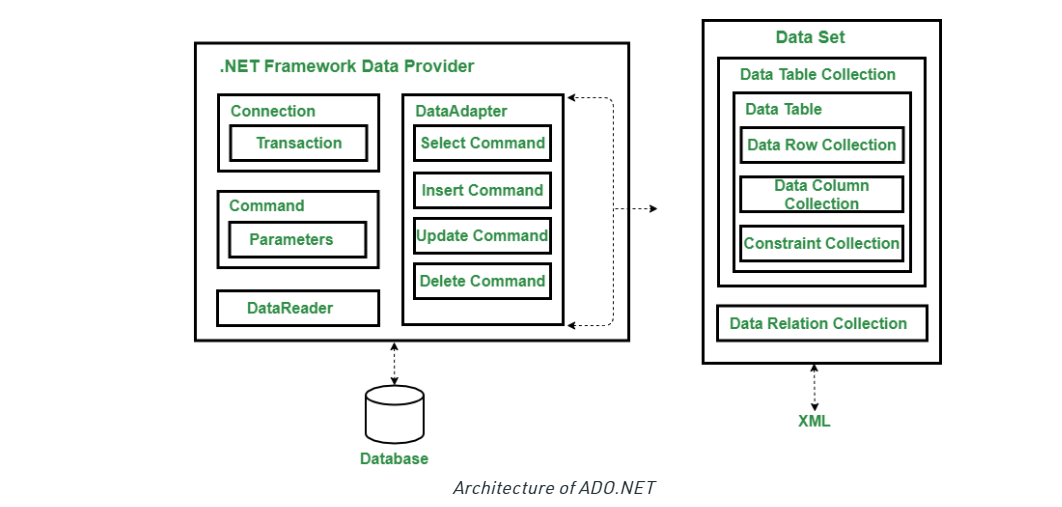
| **DML event** | **INSERTED table holds** | **DELETED table holds** |
| --- | --- | --- |
| INSERT | rows to be inserted | empty |
| UPDATE | new rows modified by the update | existing rows modified by the update |
| DELETE | empty | rows to be deleted |

# Introduction to ADO.NET

The .NET Framework includes its own data access technology i.e. **ADO.NET(ActiveX Data Objects).**ADO.NET is the latest implementation of Microsoft’s Universal Data Access strategy. ADO.NET consists of managed classes that allows .NET applications to connect to data sources such as Microsoft SQL Server, Microsoft Access, Oracle, XML, etc., execute commands and manage disconnected data. ADO.NET is **a set of classes that expose data access services for .** **NET Framework programmers**.

ADO.NET  was primarily developed to address two ways to work with data that we are getting from data sources. The two ways are as follows :

1. The first is to do with the user’s need to access data once and to iterate through a collection of data in a single instance.
2. The second way to work with data is disconnected architecture mode, in which we have to grab a collection of data and we use this data separately from the data store itself.

****

**Features of ADO.NET :**  
The following are the features of ADO.NET –

* **Interoperability-**  
  We know that XML documents are text-based formats. So, one can edit and edit XML documents using standard text-editing tools. ADO.NET uses XML in all data exchanges and for internal representation of data.
* **Maintainability –**  
  ADO.NET is built around the idea of separation of data logic and user interface. It means that we can create our application in independent layers.
* **Programmability (Typed Programming) –**  
  It is a programming style in which user words are used to construct statements or evaluate expressions. For example: If we want to select the “Marks” column from “Kawal” from the “Student” table, the following is the way to do so:

**DataSet.Student("Kawal").Marks;**

* **Performance –**  
  It uses disconnected data architecture which is easy to scale as it reduces the load on the database. Everything is handled on the client-side, so it improves performance.
* **Scalability –**  
  It means meeting the needs of the growing number of clients, which degrading performance. As it uses disconnected data access, applications do not retain database lock connections for a longer time. Thus, it accommodates scalability by encouraging programmers to conserve limited resources and allow users to access data simultaneously.

ADO.Net connection

Database connectivity and functionality are defined in the ADO.NET namespaces of the .NET Framework. ADO.NET comes with several data providers including SqlClient, OleDB, and ODBC. .NET framework also provides in-memory data access using LINQ. In this article, we will use the SqlClient data provider of ADO.NET.

SQL data provider of ADO.NET is defined in the System.Data.SqlClient namespace. We'll also use the System.Data and System.Data.Sql namespace that provide additional database functionality.

## **System.Data.SqlClient**

This assembly (namespace) of .NET Framework contains all of the classes required to connect to a SQL Server database and read, write, and update. The namespace provides classes to create a database connection, adapters, and SQL commands that provide the functionality to execute SQL queries.

We could have done it using the simple line to line code like:

1. SqlConnection conn = **new** SqlConnection();
2. conn.ConnectionString = "connection\_string";
3. conn.Open();
4. // use the connection here
5. conn.Close();
6. conn.Dipose();
7. // remember, there is no method to flush a connection.

## **Executing the Commands**

Once connected to the database, you can execute the set of commands that you're having and which would execute on the server (or the data provider) to execute the function you're trying to do, such as a query for data, insert the data, update records and so on and so forth.

SQL has the basic syntax for the commands and in my opinion, has the simple syntax of commands and nearly human-understandable commands in the programming world. In the namespace the class, SqlCommand does this job for us. An example of a command would be like:

1. SqlCommand command = **new** SqlCommand("SELECT \* FROM TableName", conn);

## **Sql Parameters**

Parameterizing the query is done using the SqlParametered into the command. For example, you might want to search for the records where the criteria match. You can denote that criteria, by ing the variable name into the query and then adding the value to it using the SqlParameter object. For instance, the following is your SqlCommandto be ed on to the server:

1. // Create the command
2. SqlCommand command = **new** SqlCommand("SELECT \* FROM TableName WHERE FirstColumn = @0", conn);
3. // Add the parameters.
4. command.Parameters.Add(new SqlParameter("0", 1));

**SQL Injection:**

SQL injections are one of the most common and dangerous security threats you can face, and no programming language or stack is immune to them. Yes, .NET SQL injection is a thing and you'd better learn how to prevent it.

That's what this post is about. We'll start with a brief explanation of SQL injections in general. You'll understand what this attack is, how it works, and why it's so dangerous.

After that, we'll get to the .NET-specific portions of the post, where we'll show you what SQL injections look like in .NET and how you can prevent them. Let's get started.

## .NET SQL Injection: What Is It? Why Care?

As promised, let's start with the basics.[What is SQL injection?](https://www.stackhawk.com/blog/what-is-sql-injection/) Why is it so important?

A SQL injection is a type of injection attack in which an ill-intended actor successfully injects—you've guessed it!—excerpts of SQL code into your application. They do that by exploring vulnerabilities that exist in portions of the app where it interacts with—and receives data from—the external world. In web applications, classical entry points for SQL injection attacks can be form fields and URL parameters.

How does a SQL injection attack work? You'll learn about the mechanics of the attack in more detail soon. For now, suffice it to say that, when successfully injecting SQL code, an attacker is able to append text to a legitimate query in the application. The query is sent to the database and executed, but its behavior is now different from what it's supposed to be.

A successful SQL attack can be devastating. It can enable the attacker to:

* Access/steal unauthorized data
* Include unauthorized data in the target database
* change or delete data from the database

To sum it up: SQL injections can be really bad for your organization. The consequences can be catastrophic, including not only financial but also legal and reputational ones.

## .NET SQL Injection: A Simple Example

I promised I would explain how a .NET SQL injection works in further detail. Time to deliver on that promise.

Let's say you've written a blog engine using .NET. A very common feature for blog engines is to include the title of posts in their URLs without spaces or special characters—we call that a "slug." That slug is then used to retrieve the relevant post from the database.

So, it's not unreasonable to think your app would have code like this:

var query = "SELECT Title, Body, Excerpt FROM Post WHERE Slug = '" + slug + "' ORDER BY Published DESC";  
  
// query execution, etc

So, what's the problem? Considering the **slug** variable comes from user input and it's simply concatenated as is in the query, this application is open to a SQL injection. The malicious actor could edit the URL so it looks like this:

<YOUR-DOMAIN-NAME>/posts/'; DROP TABLE Post;--

The string concatenation would result in the following query:

SELECT Title, Body, Excerpt FROM Post WHERE Slug = ''; DROP TABLE Post;--' ORDER BY Published DESC

As you can see, the single quote provided by the malicious actor matched the opening one already existing within the query. Then, the malicious query emits a DROP TABLE statement. Following that, it includes the "--" character, which makes everything else be considered as a comment.

## How to Prevent .NET SQL Injection

As you've seen, SQL injection attacks can deal a devastating blow to your application. How can you avoid that?

A lot of the advice to prevent SQL injections is related to "sanitizing your input." While distrusting all user input is a wise choice in general, sanitizing inputs won't suffice as a solution.[This article by Kevin Smith](https://kevinsmith.io/sanitize-your-inputs/) gives a nice explanation as to why. So, what are the valid solutions?

### Validating Input

Often, the methods in your application will accept only a small subset of all possible values as valid. For instance, if your ASP.NET MVC app has an action that gets an integer as a parameter...Well, there's really no reason to allow it to receive a string. Indicating the correct type in the action method will cause attempts of passing anything that's not an integer to fail.

### Using Allowlists

This is perhaps a continuation of the previous idea. In some situations, the set of valid values will be small enough that you'll be able to create a list with all of them. For instance, imagine a bug-tracking system in which you can filter tickets by their severity (e.g. low, medium, high, critical.) Since the set of possible values is so small, the application code can keep the whole list in memory and validate the URL parameters. If it's not on the list, it's a hard no.

### Using Parametrized Queries

The most excellent method when it comes to avoiding .NET SQL injections—or injections in any other tech stack—is to use parametrized queries.

Parametrized queries avoid SQL injection in a clever way: You simply don't include the user-provided values in the query at all. They are provided to the database separately, *never being part of the text of the SQL query itself.*

The query in our previous example would be written like this:

var query = "SELECT Title, Body, Excerpt FROM Post WHERE Slug = @slug ORDER BY Published DESC";

We avoid string concatenation by using a placeholder for the slug. Great, but how do we pass the actual value?

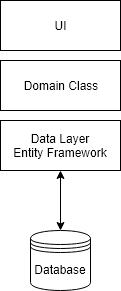
After creating a new **SqlCommand**, you would add a new parameter using the **AddWithValue** method:

var query = "SELECT Title, Body, Excerpt FROM Post WHERE Slug = @slug ORDER BY Published DESC";  
var command = new SqlCommand(query, connection);  
command.Parameters.AddWithValue("@slug", slug);

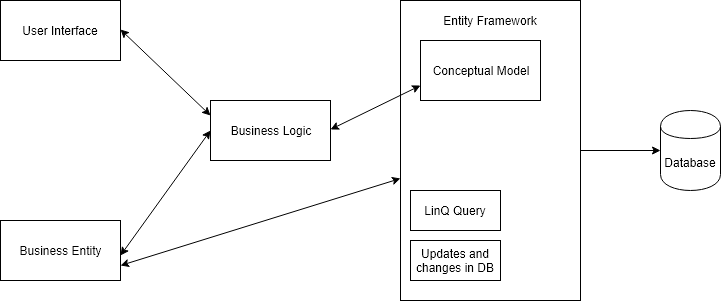
# Entity Framework in .NET

Entity Framework is an open-source object-relational mapper framework for [.NET](https://www.geeksforgeeks.org/introduction-to-net-framework/) applications supported by Microsoft. It increases the developer’s productivity as it enables developers to work with data using objects of domain-specific classes without focusing on the underlying database tables and columns where this data is stored. It eliminates the need for most of the data-access code which is used to interact with the database that developers usually need to write. It provides an abstract level to the developers to work with a relational table and columns by using the domain-specific object. It also reduces the code size of the data specific applications and also the readability of the code increases by using it. This is a new technology for accessing the data for Microsoft application. The latest version for Entity Framework is 6.0.

The following figure describes where the Entity Framework present in your application.

[](https://media.geeksforgeeks.org/wp-content/uploads/20190723185804/Entity1.png)

The above figure represents how an entity framework interacts with the domain class and database. It provides a connection between the business entity and data tables in the database.



### Features of Entity Framework

* It is platform independent.
* It uses LinQ queries to manipulate the data in the database instead of SQL queries.
* It keep the track of values that have been changed of the properties of entities.
* It also save changes which are done insert, delete or update operations.
* It also handle concurrency so the data override by a user and will reflect when another use fetches it.
* It also handles transaction management automatically and also provides the customize options for transaction management.
* It provides caching which means it stores the result of the frequently used queries.
* It also follow certain conventions for programming so it by default configure the EF Model.
* It also allows to configure the EF Model by a fluent API to override the default convention.

## Difference Between ADO.Net vs Entity framework

The ADO.Net vs Entity framework is defined as, the ADO.Net stands for ActiveX Data Object which is used to connect relational and non-relational systems, which provides an advantage to create everything from scratch and have full access control of a database in the application, by using it we can get data from ADO.Net in disconnected mode and, whereas, entity framework is an object-relational mapping framework for ADO.Net which is an open-source, that enables developers to work with data in the form of domain-specific objects and it reduces the time of development so that we can easily be focused on production instead of development.

|  |  |  |
| --- | --- | --- |
| **S.No** | **ADO.Net** | **EntityFframework** |
| 1. | It provides a bridge between relational and non-relational systems. | Whereas, it is an object-relational mapping framework for ADO.Net which is an enhancement to ADO.Net. |
| 2. | It allows communicating between application and database. | Where it handles the database operations. |
| 3. | It uses the wrapper of entity framework so that by using it the development is faster, hence it gives the fast performance. | Whereas, this is not faster in performance. |
| 4. | It is directly connected to the database. | Whereas, the LINQ queries translate into SQL at first and then it processes the query. |
| 5. | It requires more time and more effort to build a complete structure of the data access layer. | Thus it requires less time and fewer efforts to handle the database operations. |
| 6. | It gives complete control to the data layer so that developers can create classes and methods. | Whereas it automatically creates the model and database, context classes. |
| 7. | In ADO.Net debugging is embarrassing as we go from the application layer to the database layer as it has maintainable code. | Whereas it provides a clear model relationship between entities and dependent tiers. |
| 8. | In ADO.Net, it performs bulk data SQL operations and reporting. | On the other hand, the entity framework performs the CRUD operations in a project. |
| 9. | It is more flexible in terms of raw SQL queries and procedures because it has more control on the database. | Thus, this is not flexible, because it works on LINQ queries that return the entity type. |
| 10. | ADO.Net is easy to modify things than an entity framework. | Whereas, it is complicated to modify things according to our needs |

**Reference :** [**https://www.educba.com/ado-net-vs-entity-framework/**](https://www.educba.com/ado-net-vs-entity-framework/)

**Asp.Net Core Web API**

ASP.Net Web API is a lightweight framework used for building stateless and RESTful HTTP services. RESTful services are lightweight, stateless, client-server based, cacheable services that are based on the concept of resources. REST is an architectural style -- a set of constraints used to implement stateless services. It is an architectural paradigm that is used to create reusable, scalable services.

API stands for Application Programming Interface. API is actually some kind of interface which is having a set of functions. These set of functions will allow programmers to acquire some specific features or the data of an application.

Web API is an API as the name suggests, it can be accessed over the web using the HTTP protocol. It is a framework that helps you to create and develop HTTP based RESTFUL services. The web API can be developed by using different technologies such as java, ASP.NET, etc. Web API is used in either a web server or a web browser. Basically Web API is a web development concept. It is limited to Web Application’s client-side and also it does not include a web server or web browser details. If an application is to be used on a distributed system and to provide services on different devices like laptops, mobiles, etc then web API services are used. Web API is the enhanced form of the web application.

**ASP.NET Web API:** ASP.NET stands for Active Server Pages.NET. It is mostly used for creating web pages and web technologies. It is considered a very important tool for developers to build dynamic web pages using languages like C# and Visual Basic. ASP.NET Web API is a framework that helps you to build services by making it easy to reach a wide range of clients including browsers, mobiles, tablets, etc. With the help of ASP.NET, you can use the same framework and same patterns for creating web pages and services both.

**Where to use Web API?**

1. Web APIs are very useful in implementation of RESTFUL web services using .NET framework.
2. Web API helps in enabling the development of HTTP services to reach out to client entities like browser, devices or tablets.
3. ASP.NET Web API can be used with MVC for any type of application.
4. A web API can help you develop ASP.NET application via AJAX.
5. Hence, web API makes it easier for the developers to build an ASP.NET application that is compatible with any browser and almost any device.

**Why to Choose Web API?**

* A Web API services are preferable over other services to use with a native application that does not support SOAP but require web services.
* For creating resource-oriented services, the web API services are the best to choose. By using HTTP or restful service, these services are established.
* If you want good performance and fast development of services, the web API services are very helpful.
* For developing light weighted and maintainable web services, web API services are really helpful to develop that service. It supports any text pattern like JSON, XML etc.
* The devices that have tight bandwidth or having a limitation in bandwidth, then the Web API services are the best for those devices.

What is the use of HttpRequestMessage?

The HttpRequestMessage class contains headers, the HTTP verb, and potentially data. This class is commonly used by developers who need additional control over HTTP requests. Common examples include the following: **To examine the underlying SSL/TLS transport information**.

## **HttpResponseMessage**

A HttpResponseMessage allows us to work with the HTTP protocol (for example, with the headers property) and unifies our return type. In simple words an HttpResponseMessage is a way of returning a message/data from your action.

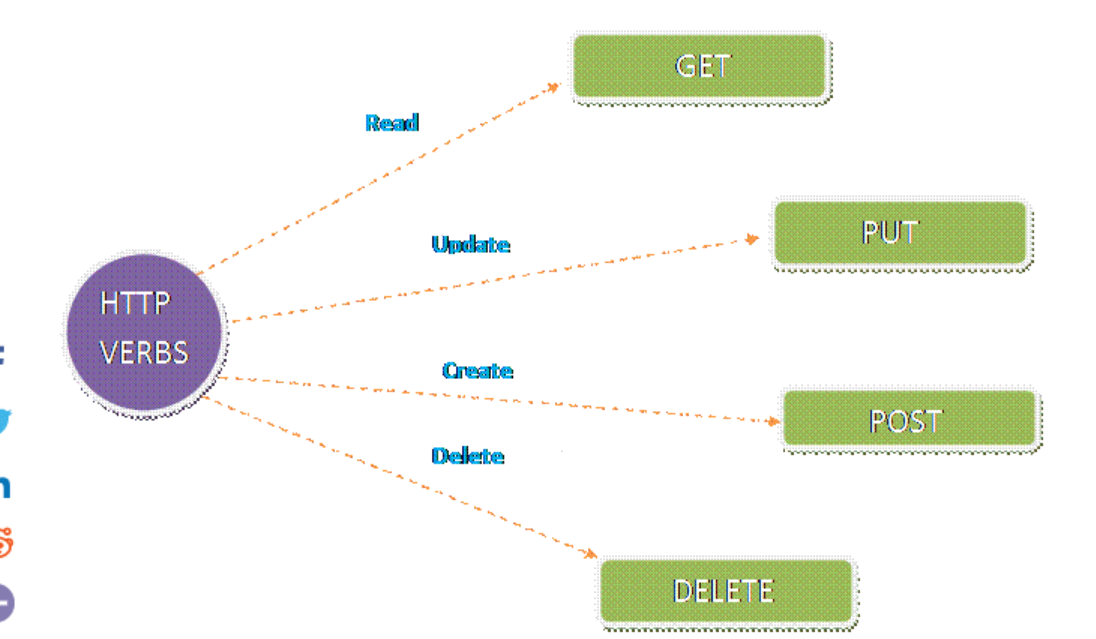
**IHttpActionResult**

The **IHttpActionResult** interface was introduced in Web API 2. Essentially, it defines an **HttpResponseMessage** factory. Here are some advantages of using the **IHttpActionResult** interface:

* Simplifies [unit testing](https://docs.microsoft.com/en-us/aspnet/web-api/overview/testing-and-debugging/unit-testing-controllers-in-web-api) your controllers.
* Moves common logic for creating HTTP responses into separate classes.
* Makes the intent of the controller action clearer, by hiding the low-level details of constructing the response.

**IHttpActionResult** contains a single method, **ExecuteAsync**, which asynchronously creates an **HttpResponseMessage** instance.

HTTP Operations GET, POST, PUT and DELETE From .NET Client

****

**GET:**

Get method returns what are the strings assigned in the languages variable. Get method contains the code, which looks, as shown below.

1. // GET api/values/5
2. **public** string Get(**int** id)
3. {
4. **return** languages[id];
5. }

**POST:**

If need to Add or Save data in the list arises, we can use Post method. For Post method, we use code given below.

1. // POST api/values
2. **public** **void** Post([FromBody]string value)
3. {
4. languages.Add(value);
5. }

**PUT AND DELETE:**

We can update and delete the data in the list, using the code given below.

1. // PUT api/values/5
2. **public** **void** Put(**int** id, [FromBody]string value)
3. {
4. languages[id] = value;
5. }
7. // DELETE api/values/5
8. **public** **void** Delete(**int** id)
9. {
10. languages.RemoveAt(id);

## **CRUD operations via ADO.Net**

**GET:**

1. DataTable dt = **new** DataTable();
2. **string** strConString = @ "Data Source=WELCOME-PC\SQLSERVER2008;Initial Catalog=MyDB;Integrated Security=True";
3. **using**(SqlConnection con = **new** SqlConnection(strConString)) {
4. con.Open();
5. SqlCommand cmd = **new** SqlCommand("Select \* from tblStudent", con);
6. SqlDataAdapter da = **new** SqlDataAdapter(cmd);
7. da.Fill(dt);
8. }
9. **return** dt;
10. var myData = ds.Tables[0].AsEnumerable().Select(r => new Employee {
11. Name = r.Field<string>("Name"),
12. Age = r.Field<int>("Age")
13. });
14. var list = myData.ToList();

**Update:**

1. **string** strConString = @ "Data Source=WELCOME-PC\SQLSERVER2008;Initial Catalog=MyDB;Integrated Security=True";
3. **using**(SqlConnection con = **new** SqlConnection(strConString)) {
4. con.Open();
5. **string** query = "Update tblStudent SET student\_name=@studname, student\_age=@studage , student\_gender=@gender where student\_id=@studid";
6. SqlCommand cmd = **new** SqlCommand(query, con);
7. cmd.Parameters.AddWithValue("@studname", strStudentName);
8. cmd.Parameters.AddWithValue("@studage", intAge);
9. cmd.Parameters.AddWithValue("@gender", strGender);
10. cmd.Parameters.AddWithValue("@studid", intStudentID);
11. **return** cmd.ExecuteNonQuery();
12. }

**Insert:**

1. **string** strConString = @ "Data Source=WELCOME-PC\SQLSERVER2008;Initial Catalog=MyDB;Integrated Security=True";
3. **using**(SqlConnection con = **new** SqlConnection(strConString)) {
4. con.Open();
5. **string** query = "Insert into tblStudent (student\_name, student\_age,student\_gender) values(@studname, @studage , @gender)";
6. SqlCommand cmd = **new** SqlCommand(query, con);
7. cmd.Parameters.AddWithValue("@studname", strStudentName);
8. cmd.Parameters.AddWithValue("@studage", intAge);
9. cmd.Parameters.AddWithValue("@gender", strGender);
10. **return** cmd.ExecuteNonQuery();
11. }

**Delete:**

1. **string** strConString = @ "Data Source=WELCOME-PC\SQLSERVER2008;Initial Catalog=MyDB;Integrated Security=True";
3. **using**(SqlConnection con = **new** SqlConnection(strConString)) {
4. con.Open();
5. **string** query = "Delete from tblStudent where student\_id=@studid";
6. SqlCommand cmd = **new** SqlCommand(query, con);
7. cmd.Parameters.AddWithValue("@studid", intStudentID);
8. **return** cmd.ExecuteNonQuery();
9. }

**Web API Filters:**

Web API includes filters to add extra logic before or after action method executes. Filters can be used to provide cross-cutting features such as logging, exception handling, performance measurement, authentication and authorization.

| Filter Type | Interface | Class | Description |
| --- | --- | --- | --- |
| Simple Filter | IFilter | - | Defines the methods that are used in a filter |
| Action Filter | IActionFilter | ActionFilterAttribute | Used to add extra logic before or after action methods execute. |
| Authentication Filter | IAuthenticationFilter | - | Used to force users or clients to be authenticated before action methods execute. |
| Authorization Filter | IAuthorizationFilter | AuthorizationFilterAttribute | Used to restrict access to action methods to specific users or groups. |
| Exception Filter | IExceptionFilter | ExceptionFilterAttribute | Used to handle all unhandled exception in Web API. |
| Override Filter | IOverrideFilter | - | Used to customize the behaviour of other filter for individual action method. |

**Routing:**

**Overview**

Routing, an open debatable topic among many in the developer community, is an interesting feature in which to deep dive. Routing is a functionally based tag or Uri template used by APIs to match the desired action or methods expected to be executed. There are two types or rather two different types of Routing being used during development. Namely, ‘Convention-based Routing’ the elder son in the REST routing family followed by ‘Attribute Routing’ the most lovable son to date. As mentioned earlier it’s an open debatable topic over using which type of Routing mechanics during APIs development and designing phase as well.

In ‘Convention-based Routing’, route templates are defined by developers as per requirement, basically a set of strings of type text decorated with parameters. Once the request is received, it tries to match requested URI with this defined route templates. The only merit of using this routing type is, templates are defined at a single location in application solution structures, leveraging the template rules religiously across the controllers and actions.

Then, why is Attribute routing important? Yes, it is not only important but strongly recommended for API development by developers and architects across the communities. Though convention-based routing has its own Pros, while building a good API, there are few considerations, where this type of routing is not advisable. There are common URI patterns in REST APIs, which are tough to support by convention-based routing. Consider, a set of Response data or resources, are often clubbed with their hierarchical data or child resources. For eg. Departments have Employees, Songs have singers, Movies have actors and so on. URIs expected in such scenarios are,

*/movies/1/actors*

In the case of multiple controllers and huge resources this type of URI is difficult though achievable using convention-based routing, but at the cost of scaling and performance. This hits the key consideration area of designing scalable APIs. Here is where another routing type, Attribute Routing, plays a role.

**Attribute Routing**

**What is Attribute Routing?**

Technically, Attribute routing is all about attaching a route, as an attribute, to a specific controller or action method. Decorating Controller and its method with [Route] attribute to define routes is called Attribute Routing. In simpler terms, using [Route] attribute with controllers and method is Attribute Routing.

*[Route ("api/customers/{id}/orders")]*

It started from Web API 2 and now is the most recommended and adapted Routing type in RESTful APIs design and development.

**Why use Attribute Routing?**

As the name indicates, attribute routing uses attributes to define routes. Attribute routing gives you precise control over the URIs than convention-based routing in your APIs. Above described scenario of Hierarchical resources can be easily achieved by Attribute Routing, with making no compromise with the scalability of APIs.

Also, versioning APIs, overloading URI segments and multiple parameter type patterns can be achieved through attribute routing with ease.

**Working with Attribute Routing**

Any route attribute on the controller makes all actions in the controller attribute routing. Defining route attribute to the action or the controller takes precedence over conventional routing. Let’s be more precise to .NET Core APIs, it comes by default with Attribute routing. Attribute routing requires detailed input to specify a route. However, it allows more control of which route template applies to each action.

**Configuring**

When you create a WEB API with .NET Core framework, you can notice in its Startup.cs file,

1. **void** Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory) {
2. app.UseMvc();
3. }

This declaration of, ‘app.UseMvc()’ at configure section, enables Attribute Routing. This are by-default configurations of .NET Core applications. Explicit configurations thus are not required for enabling Attribute routing for .NET Core Web APIs

1. [Route("api/[controller]")]
2. publicclassBooksController: Controller {
3. [HttpGet("{id:int}/author/{authorid:int}")]
4. publicIEnumerable < string > Getdetails(**int** id, intauthorid) {
5. returnnewstring[] {
6. "V2.value1",
7. "V2.value2"
8. };
9. }
10. }

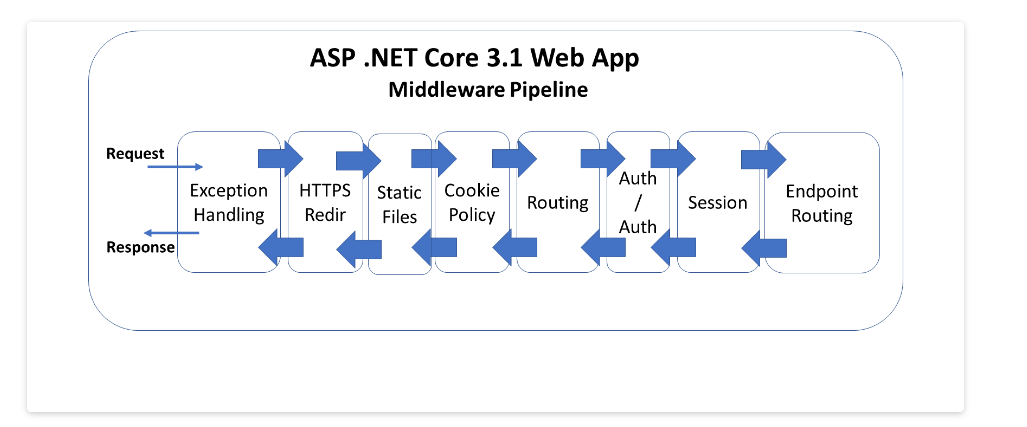
**Content negotiation:**

Content negotiation may be defined as the process of inspecting the structure of an incoming HTTP request to determine the best representation of a resource from amongst multiple available representations of the same resource. In essence, content negotiation is a concept that allows the same Url to serve the same content in various formats. You can take advantage of content negotiation to select the preferred media type.

The Web API framework comes with the following formatters by default.

* System.Net.Http.Formatting.JsonMediaTypeFormatter
* System.Net.Http.Formatting.XmlMediaTypeFormatter
* System.Net.Http.Formatting.FormUrlEncodedMediaTypeFormatter
* System.Web.Http.ModelBinding.JQueryMvcFormUrlEncodedFormatter

**Web API pipeline:**

****

**Install Swagger in Asp.Net Core Web API:**

NuGet Package Swashbuckle.AspNetCore

Add the following code to the **ConfigureServices** Method.

services.AddSwaggerGen();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo{

Version = "v1",

Title = "Implement Swagger UI",

Description = "A simple example to Implement Swagger UI",

});

});

Add the following code to the Configure Method of the Startup.cs file.

app.UseSwagger();

app.UseSwaggerUI(c =>{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Showing API V1");

});

**Entity Framework with MVC:**

* Microsoft.EntityFrameworkCore.Design 3.1.25
* Microsoft.EntityFrameworkCore 3.1.25
* Microsoft.EntityFrameworkCore.Tools 3.1.25
* Microsoft.EntityFrameworkCore.SqlServer 3.1.25
* **Microsoft.VisualStudio.Web.CodeGeneration.Design 3.1.5**

**Run below command in nuget package manager console:**

**Scaffold-DbContext "Data Source=MSP-LAPTOP;Initial Catalog=InstituteCmd;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models/DB -Table employees**

**Tools -> Nuget Package Manager -> Package Manager console**

**Add below connection string to appsettings.json**

**"ConnectionStrings": {**

**"DevConnection": "Data Source=MSP-LAPTOP;Initial Catalog=InstituteCmd;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"**

**},**

**Register the database connection context into Class starup.cs inside ConfigureServices**

**// Register SQL database configuration context as services.**

**services.AddDbContext<pubsContext>(options =>**

**{**

**options.UseSqlServer(Configuration.GetConnectionString("DevConnection"));**

**});**

**Dot Net Programs:**

**MVC Employee details**

**Add all the given numbers(823 – 13) => (823 – 8+2+3 = 13)**

**Convert number to string**

**Palindrome**

**Exercise:**

**Create a <table> that holds the employee information which should contain minimum 10 columns including salary, manager id(it should be another employee id)**

**Insert 1000 records**

**Primary key should start from 10001**

**Write a stored procedure to insert and select the records**

**Create a function to display employee full name**

**Create a trigger for a table that needs to records all insert, update and delete transactions**

**Write a query to get all the employees whoever is tagged under the particular manager**

**Update employee name from ADO.Net**

**Select employee records from ADO.Net**