

## Chapter -4

### 1. Setting up the Environment

#### ASP.NET Core Environment Setup

1. [How to set up a development machine for dot net core applications development](#)
2. [Install SDK and Editor](#)
3. [Verify the installation](#)

Tools and Software Requires for the development of .NET Core Applications.

**Machine:** (Windows,Mac, Linux)

**Editor:** Recommended Visual Studio, VS Code

**Dot Net Core SDK:** This is the software development KIT and this KIT is helpful for the development and running of the application in the system.

How to prepare a development environment for building .NET Core/ASP.NET Core applications?

The .NET Core can be installed in two ways:

1. **By Installing Visual Studio 2017/2019**
2. **By Installing .NET Core SDK**

The .NET Core SDK installer already contains ASP.NET Core libraries, so there is no separate installer for ASP.NET Core.

Install Visual Studio 2017/2019:

Currently, .NET Core 2.1 and .NET Core 3.1 is having Long Term Support (LTS). Visual Studio 2017 supports .NET Core 2.1, whereas Visual Studio 2019 supports both the versions.

You can use your favorite IDE, such as Visual Studio 2017/2019, Visual Studio Code, and Sublime Text, etc. to develop, restore, build and run .NET Core application. In my machine, I have installed Windows operating system. So, I am going to use Visual Studio as my editor, or you can say Integrated Development Environment (IDE) for developing ASP.NET Core application. But this is not mandatory. You can use any editor as per your choice.

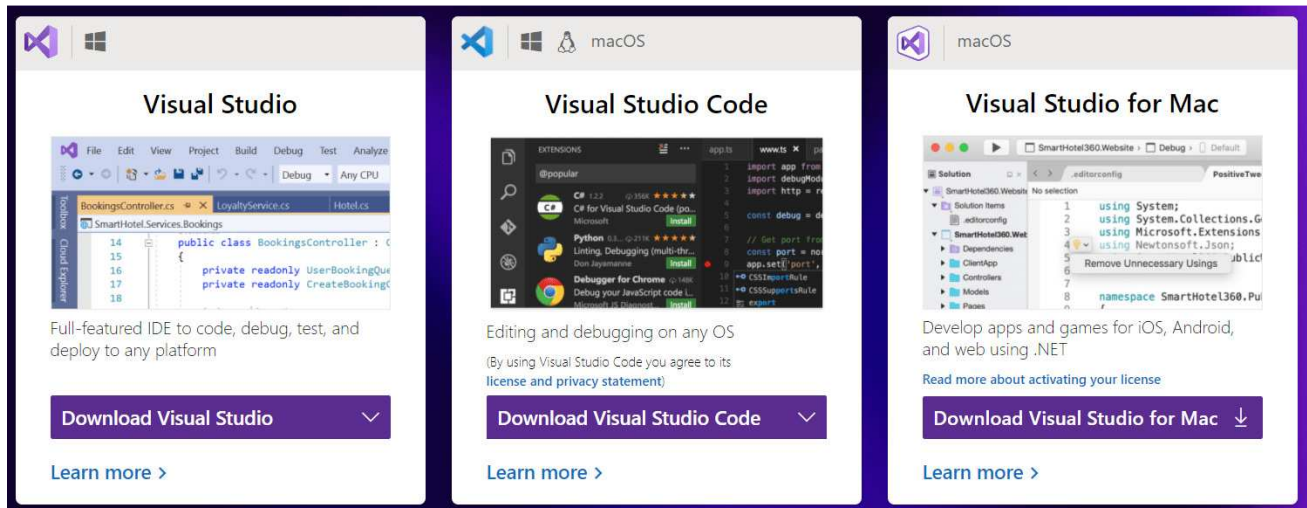
If you don't have Visual Studio on your development PC, then it is recommended to install the latest Visual Studio 2019. If you already have either Visual Studio 2017 or 2019, then you already have installed .NET Core SDK 2.1.

## Download and install Visual Studio

Download and install Visual Studio 2019 based on your OS. Select the appropriate edition as per your license. The Visual Studio community edition is free for students, open-source contributors, and individuals. As of this article, the latest version of Visual Studio is **Visual Studio 2019** and it can be downloaded from the following site

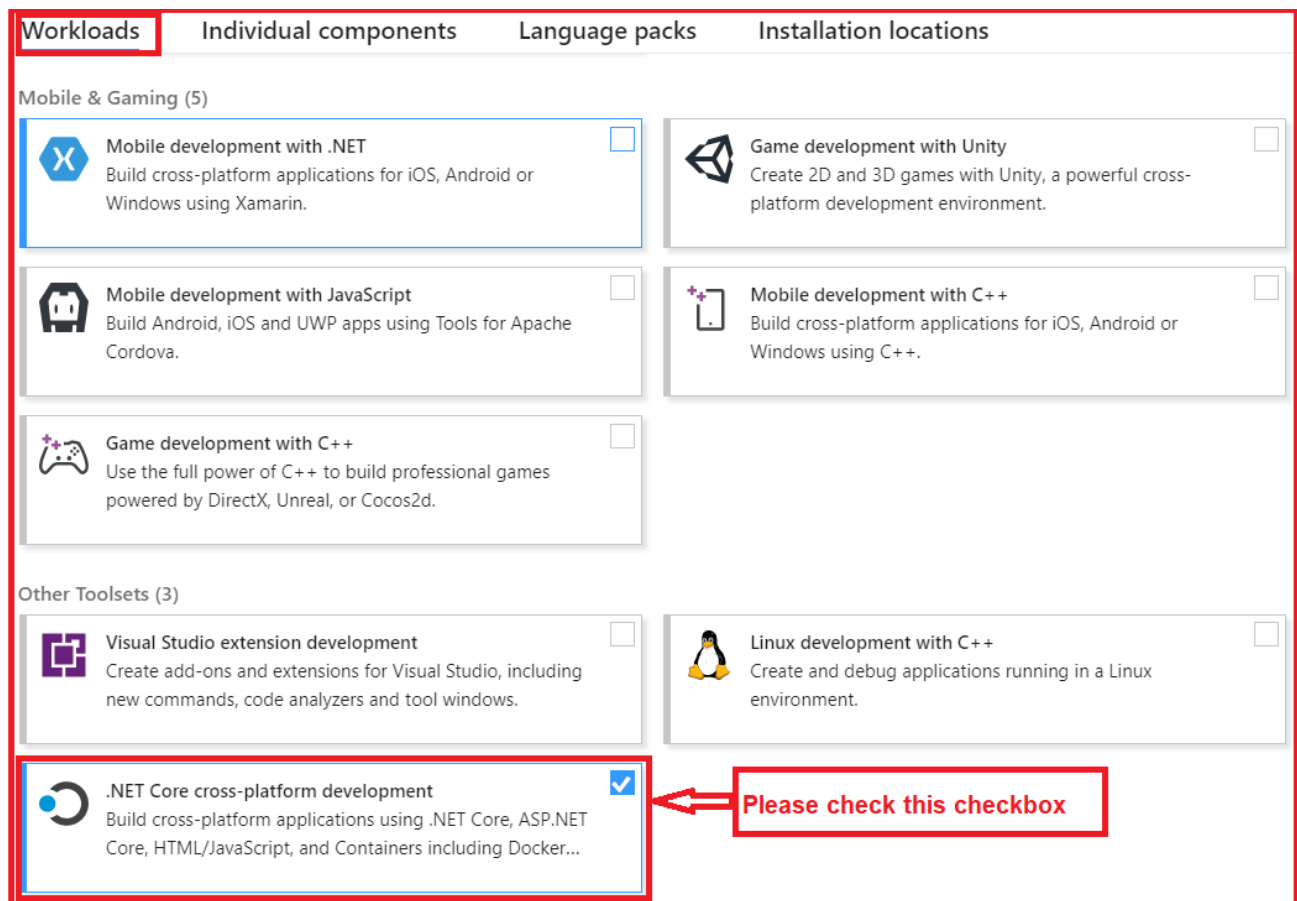
<https://visualstudio.microsoft.com/>

Once you visit the above website, you will see the following.



Based on your operating system download the required Visual Studio. In my machine, I have installed the Windows Operating System, so I choose Visual Studio and select the Community Edition and download it.

To develop .NET Core applications in Visual Studio 2019 we need .NET Core cross-platform development. So, while installing the Visual Studio 2019, please make sure to select the .NET Core cross-platform development workload checkbox as shown in the below image. This will install the .NET Core SDK 2.1. However, you need to install .NET Core 3.1 SDK separately if by default not available while installing Visual Studio 2019.



If you already have installed the Visual Studio 2019, then make sure the above .NET Core cross-platform development is installed. If not installed, then you just install this by using Visual Studio Installer.

Once you installed, then you can verify the same using command prompt (Windows OS) and Terminal (Linux OS). As in my Machine, I have installed Windows Operating System, So, I can verify the same using the “**dotnet --version**” command as shown in the below image.

```
Command Prompt
Microsoft Windows [Version 10.0.19041.329]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>dotnet --version
3.1.401
```

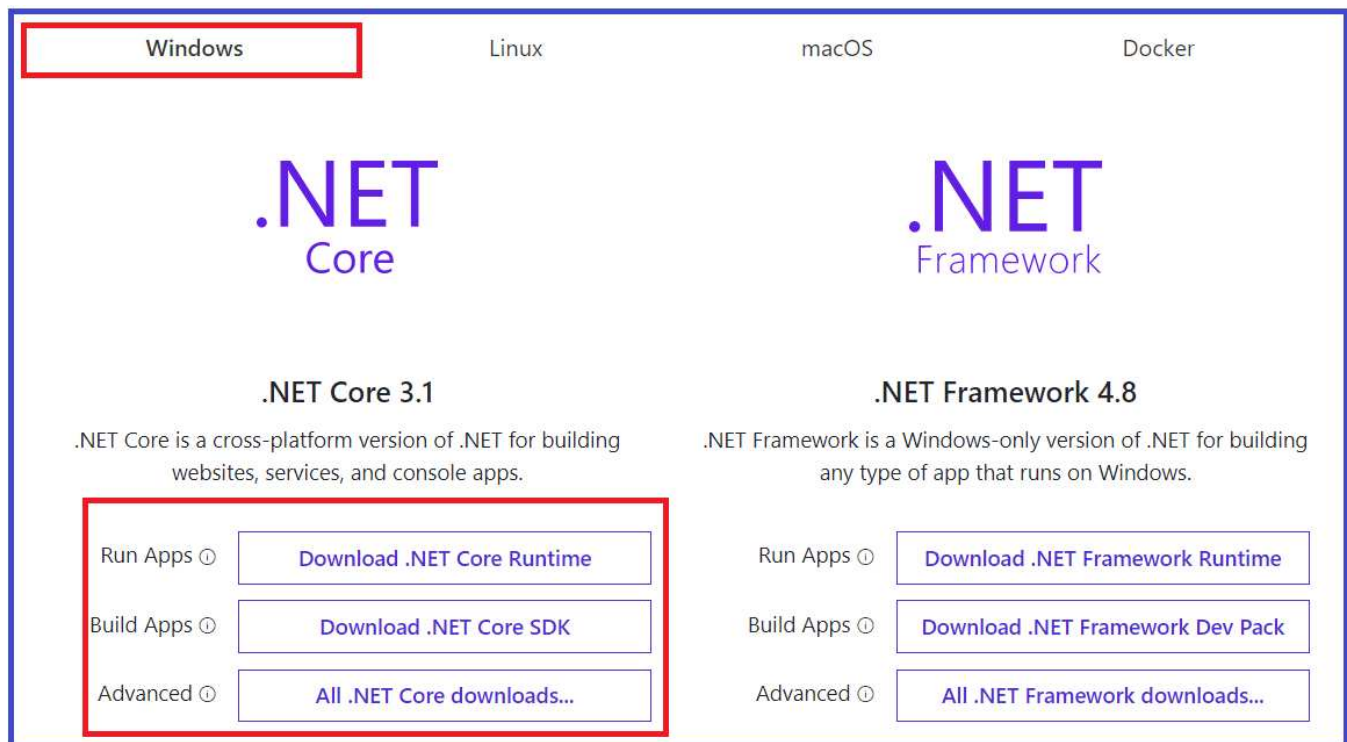
If the .NET Core Framework installed successfully, in your machine, they will see the version number as shown in the above image.

## Download and install .NET Core SDK

If Visual Studio 2019 installer doesn't include .NET Core 3.x then you need to install it separately. The ASP.NET Core is a part of .NET Core SDK, so you don't require to install it separately. To download the later version of .NET Core i.e. 3.1 as of today date, go to the following website and select the platform you are using.

<https://dotnet.microsoft.com/download>

Once you visit the above page, you will find the following. Here you will find two things as shown in the below image











As you can see, when we select the Windows option, then we have two options to develop .NET Core Application i.e. .NET Core 3.1 and .NET Framework 4.8. As you can see in the above image, .NET Core is a cross-platform of .NET for building websites, services, and console apps. On the other hand, .NET Framework is a Windows-only version of .NET for building any type of app that runs on Windows only. Here we are going with .NET Core 3.1 which is the latest version of .NET Core Framework as of this writing.

Further, in .NET Core there are three options are as follows:

**.NET Core Runtime:** The .NET Core Runtime is required only to run .NET Core applications. The .NET Core Runtime just contains the resources or libraries which are required to run existing .NET Core applications.

**.NET Core SDK:** If you want to develop and run the .NET Core application, then you need to download the .NET Core SDK. The .NET Core SDK contains the .NET Core Runtime. So, if you installed the .NET Core SDK, then there is no need to install .NET Core Runtime separately.

**All .NET Core downloads:** If you want an older version of .NET Core, then you need to click on this link and it will navigate to another page, from where you can download the .NET core Framework version as per your choice as shown in the below image.

Version	Status	Latest release	Latest release date	End of support
<a href="#">.NET 5.0</a>	Preview 	5.0.0-preview.7	2020-07-21	
<a href="#">.NET Core 3.1</a> (recommended)	LTS 	3.1.7	2020-08-11	2022-12-03
<a href="#">.NET Core 3.0</a>	End of life 	3.0.3	2020-02-18	2020-03-03
<a href="#">.NET Core 2.2</a>	End of life 	2.2.8	2019-11-19	2019-12-23
<a href="#">.NET Core 2.1</a>	LTS 	2.1.21	2020-08-11	2021-08-21
<a href="#">.NET Core 2.0</a>	End of life 	2.0.9	2018-07-10	2018-10-01
<a href="#">.NET Core 1.1</a>	End of life 	1.1.13	2019-05-14	2019-06-27
<a href="#">.NET Core 1.0</a>	End of life 	1.0.16	2019-05-14	2019-06-27

As we are going to set up a development environment for developing and running .NET Core Applications, so, we need to install .NET Core SDK.

Click on the **Download .NET Core SDK** button to download the latest version of .NET Core SDK. It will download the .NET Core 3.1 SDK as of this writing. After downloading the installer, click on it to start the installation.

## Verify the installation

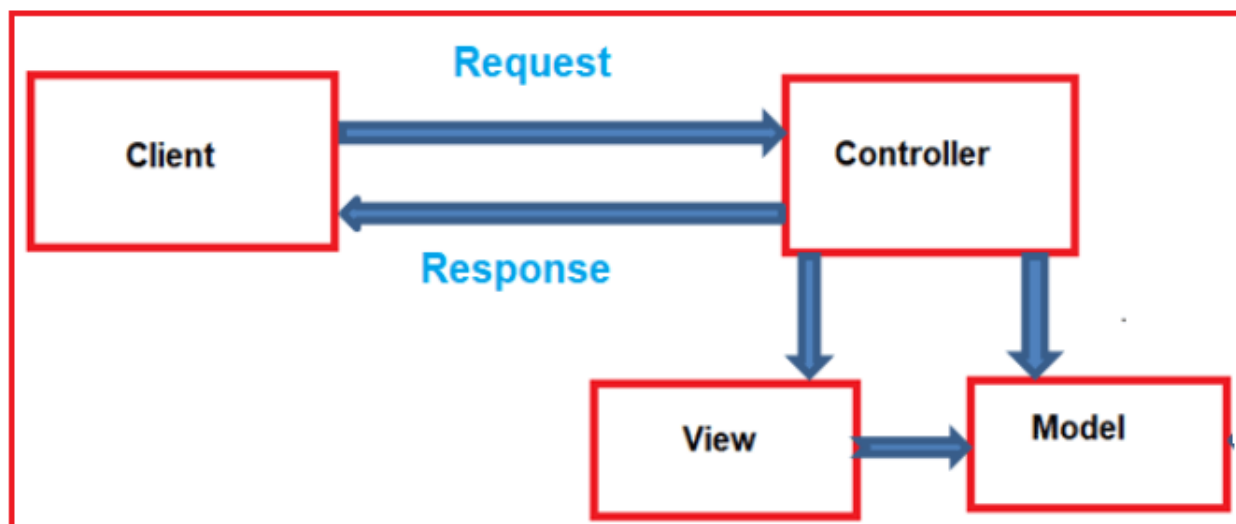
Once you install .NET Core SDK then again you can verify the same using the command prompt. Open Command Prompt and type **dotnet --version** command and press enter. If the .NET Core Framework installed successfully, in your machine, then you will see the version number as shown in the above image.

```
C:\> Command Prompt
Microsoft Windows [Version 10.0.19041.329]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\HP>dotnet --version
3.1.401
```

## 2. Controller and Action

In the MVC Design Pattern, the controller is the initial entry point and is responsible for selecting which model types to work with and which view to render (i.e., it controls how the app responds to a given HTTP request). Controllers are stored inside the **Controller Folder** in the root of the web application. They are basically C# classess whose Public methods are called as Action Methods. These Action Methods **handles the HTTP request** and prepares the response to be sent to the client.



When the client (browser) sends a request to the server, then that request first goes through the request processing pipeline. Once the request passes the request processing pipeline, it will hit the controller. Inside the controller, there are lots of methods (called action methods) actually handle that incoming HTTP Request. The action method inside the controller executes the business logic and prepared the response which is sent back to the client who initially made the request.

## Role of Controller in MVC:

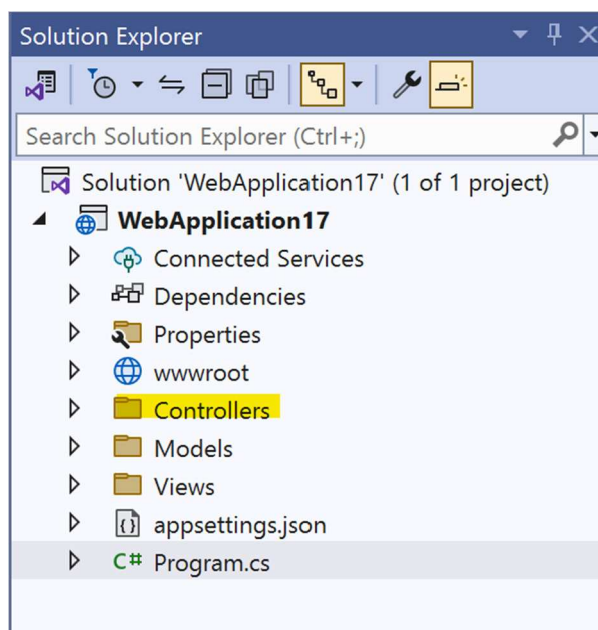
- **Group Related Actions:** A Controller groups related actions, i.e., Action Methods.
- **Request Handling:** Controllers receive HTTP requests and decide how to process them.
- **Features Applied:** Many important features, such as Caching, Routing, Security, etc., can be applied to the controller.
- **Model Interaction:** They often interact with models to retrieve data or save changes.
- **View Selection:** Controllers select a view and provide it with any necessary data for rendering.

## How to add Controllers in ASP.NET Core Application?

If you create the ASP.NET Core Application using the MVC Project Template, then by default it will create a controller called **HomeController** within the Controllers folder.

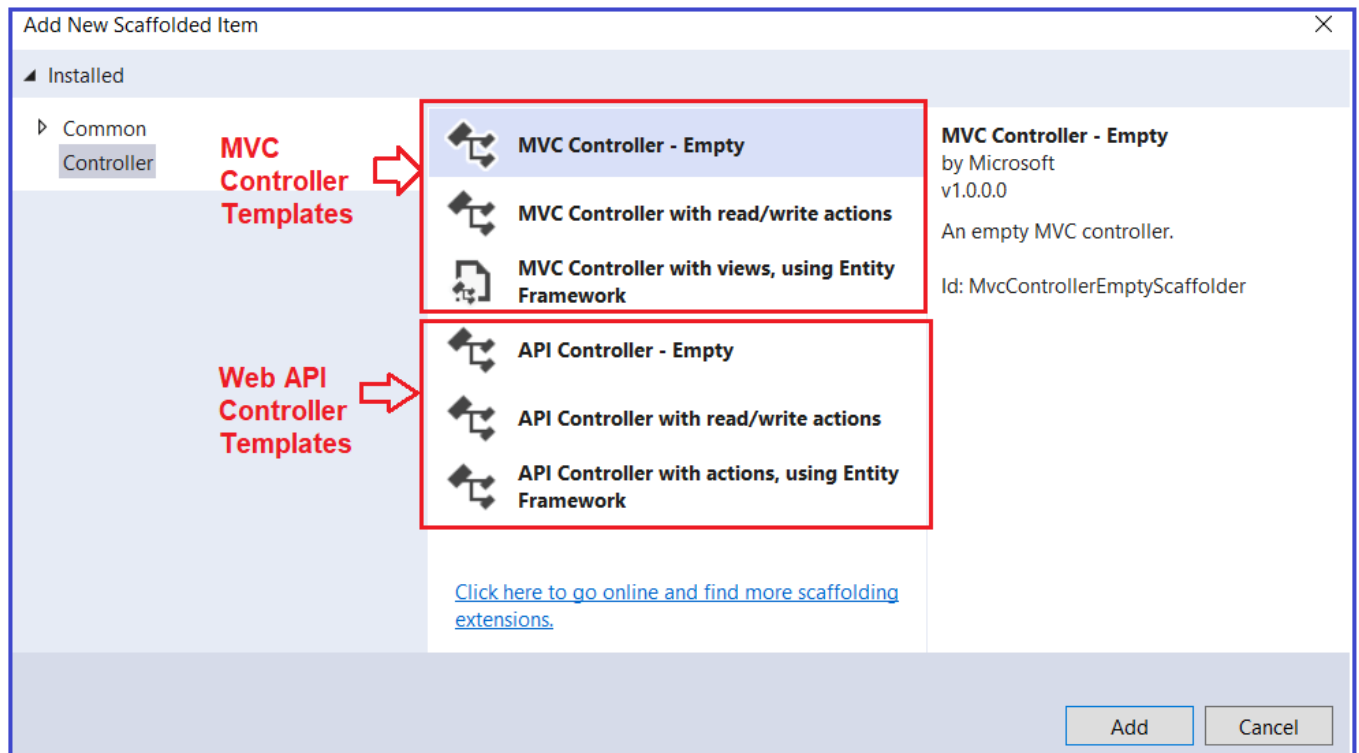
### Step1: Creating Empty ASP.NET Core Web Application

Create an ASP.NET Core Web Application with an mvc Project template. You can give any name to your application. I am giving the name for My Application as **FirstCoreMVCWebApplication**. The project will be created with the following folder structure and you can see there is folder called Controllers.



### Step3: Adding Controller

Once you create a new project place (StudentController) inside this Controllers folder. To do so, right-click on the Controller folder and select the Add => Controller option from the context menu which will open the Add Controller window as shown in the below image.



As you can see in the above image, we have three templates to create an MVC controller as well as three templates to create a Web API Controller. As we are interested MVC Application, so you can use any of the following three templates:

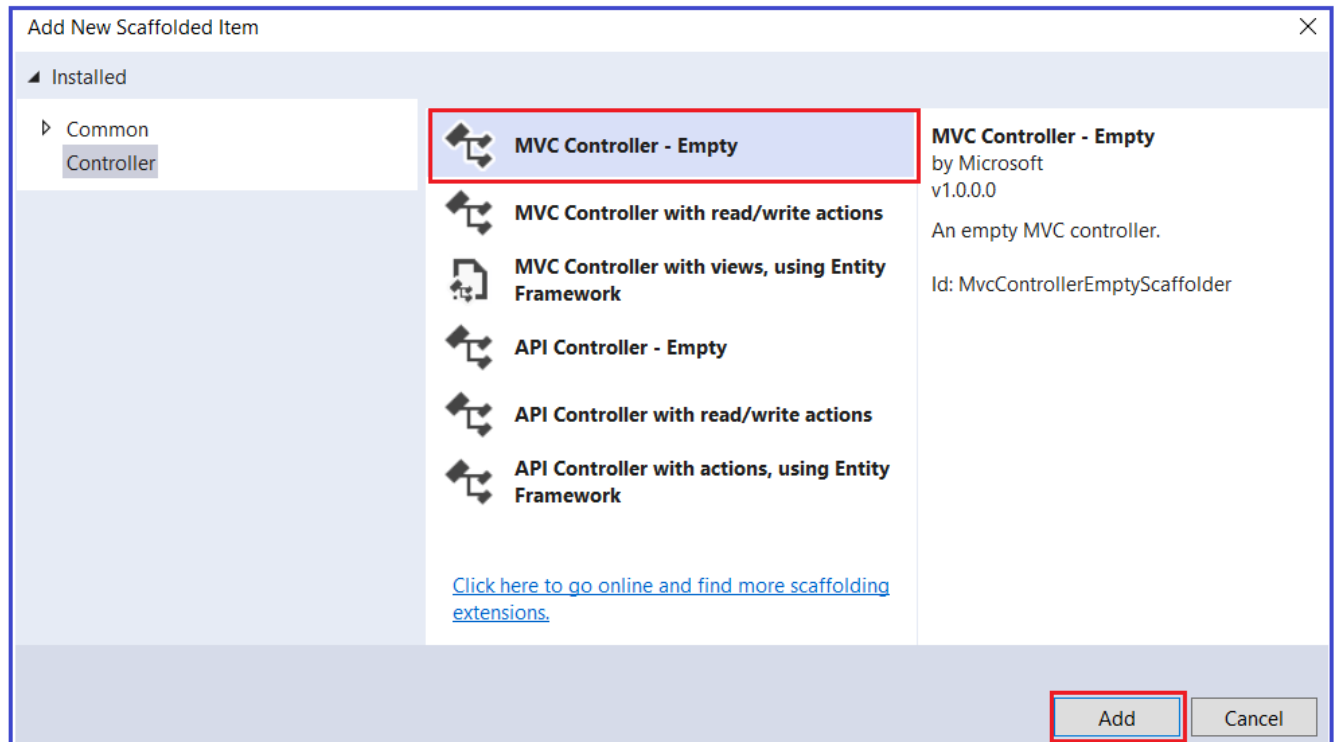
**MVC Controller – Empty:** It will create an Empty Controller.

**MVC Controller with read/write actions:** This template will create the controller with five action methods to create, read, update, delete, and list entities.

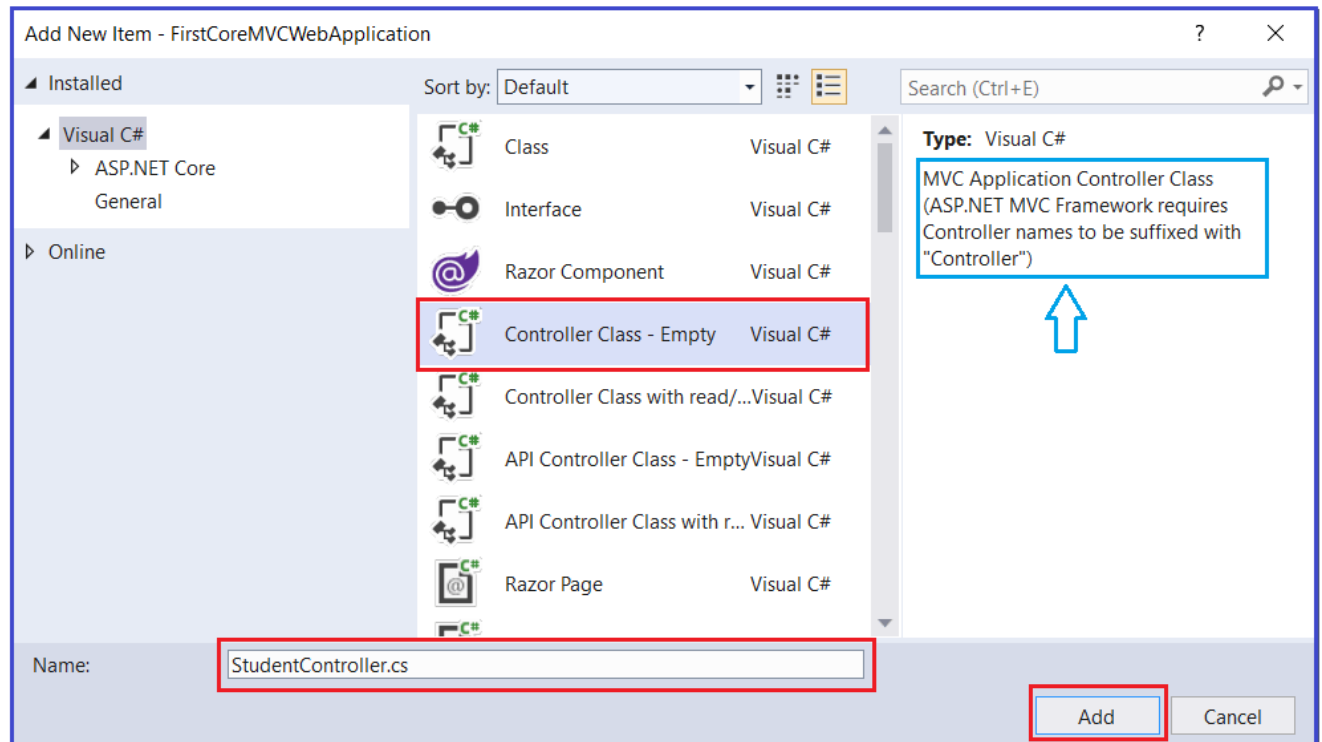
**MVC Controller with views, using Entity Framework:** This template will create an MVC Controller with actions and Razor views to create, read, update, delete, and list entities using Entity framework.

Here, we are going to create the MVC Controller with the Empty template. So, select the MVC Controller – Empty option and click on the Add button as shown in the below image.

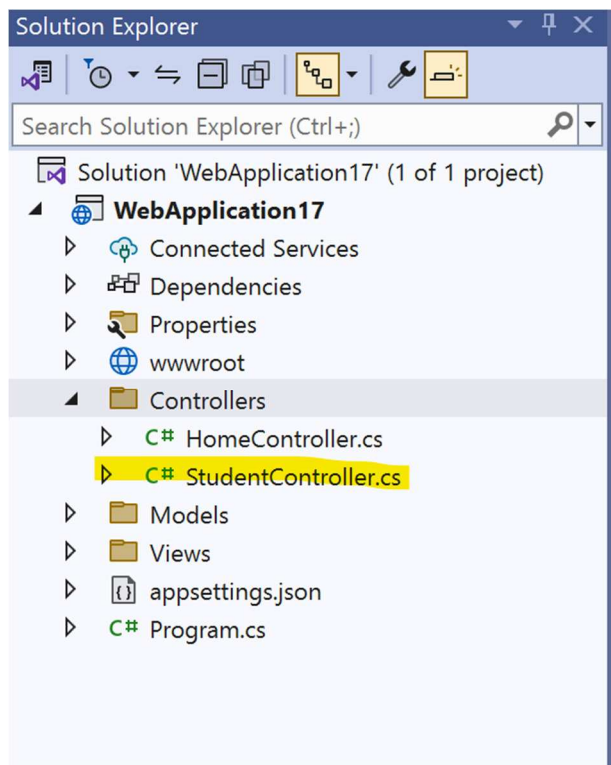




Once you click on the Add button, it will open the below window where you need to select the **Controller Class – Empty** option and give a meaningful name to your controller. Here, I am giving the name as **StudentController** and click on the Add button. Here, the point that you need to remember is the Controller names should be suffixed with the word Controller.



Once you click on the Add button then it will add StudentController within the Controllers folder as shown in the below image.



## Understanding StudentController:

Now let us understand the StudentController class and the different components of this class. First of all, it is a class having a **.cs extension**. Open the **StudentController.cs** class and you should get the following default code in it.

```
using Microsoft.AspNetCore.Mvc;

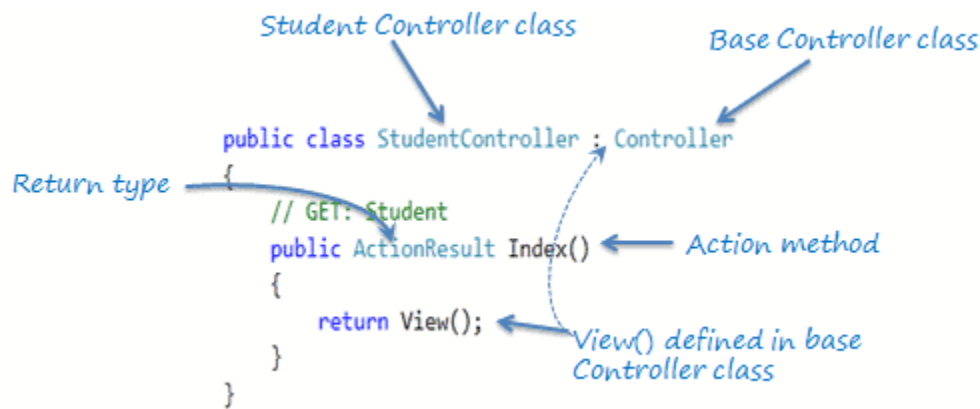
namespace FirstCoreMVCWebApplication.Controllers
{
    public class StudentController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

### 3. Action

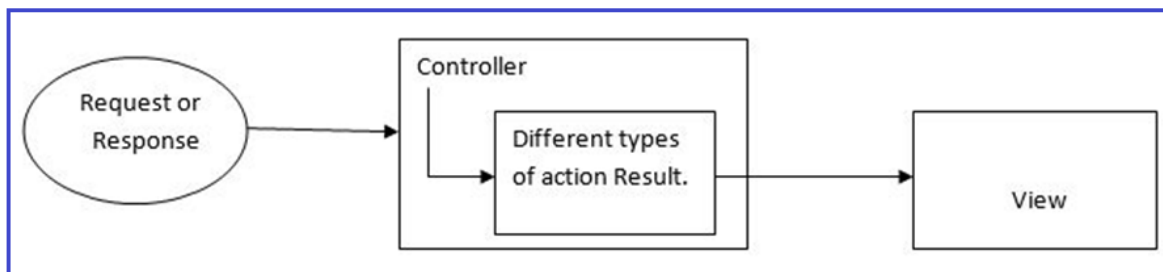
All the public methods of the **Controller** class are called **Action** methods. They are like any other normal methods with the following restrictions:

- Action method must be public. It cannot be private or protected
- Action method cannot be overloaded
- Action method cannot be a static .

The following illustrates the **Index()** action method in the **StudentController** class.



As you can see in the above figure, the **Index()** method is public, and it returns the **ActionResult** using the **View()** method. The **View()** method is defined in the Controller base class, which returns the appropriate **ActionResult**.



Action Method	Description
<b>ActionResult</b>	Defines a contract that represents the result of an action method.
<b>ActionResult</b>	A default implementation of IActionResult.
<b>ContentResult</b>	Represents a text result.
<b>EmptyResult</b>	Represents an ActionResult that when executed will do nothing.
<b>JsonResult</b>	An action result which formats the given object as JSON.
<b>PartialViewResult</b>	Represents an ActionResult that renders a partial view to the response.
<b>ViewResult</b>	Represents an ActionResult that renders a view to the response.

Example:

```
public IActionResult Index()
{
    return View();
}
public ActionResult About()
{
    return View();
}

public ContentResult ContentResult()
{ //localhost/Example/ContentResult
    return Content("I am ContentResult");
}

public JsonResult GetValue()
{
//Student/GetValue (Browser)
    var name = "Nawaraj Prajapati";
    return Json(new { data = name });
}
public PartialViewResult PartialViewResult()
{
    return PartialView("_PartialView");
}

public ViewResult ViewResult()
{
    return View("About", "Student");
}
```

## 4. Rendering HTML with Views

**Razor Syntax** is a powerful and easiest way to write server-based code directly into your view pages. Razor is a markup syntax that allows you to embed C# Programming directly into your view page. It is generally written in the **.cshtml** file.

- Razor code blocks are enclosed in `@{ ... }`
- Inline expressions (variables and functions) start with `@`
- Code statements end with semicolon
- Variables are declared with the `var` keyword
- Strings are enclosed with quotation marks
- C# code is case sensitive
- C# files have the extension `.cshtml`

**Example 1:** Use `@{ ... }` block to write C# code.

```
@{
    ViewData["Title"] = " net centric class ACHS ";
}

<div class="jumbotron">
    <h2 style="color:chocolate">Welcome to Computer Shop Management</h2>
</div>
<div>
    @{
        for (int i = 0; i < 5; i++)
        {
            <h3 style="color:deeppink">Hello world @i</h3>
        }
    }
</div>
```

**Example 2:** All the Inline Expressions like Variables and Functions starts with `@`.

```
@{
    ViewData["Title"] = "net centric class ACHS";
}

<div>
    @{
        int i = 10;
        <h2>@i</h2>

        <h3 style="color:deeppink"><i>Today Date is
@DateTime.Now.ToShortDateString()</i></h3>
```

```
}  
</div>
```

**Example 3:** Variables are declared with the **var** keyword and all the code statements end with a **semicolon(;)**.

```
@{  
    ViewData["Title"] = "net centric class ACHS";  
}  
  
<div>  
    @{  
        var a = 5;  
        var b = 10;  
        var result = a + b;  
        <h2>Addition of @a + @b = @result</h2>  
    }  
</div>
```

**Example 4:** Keep String value within **double quotes (" ")**

```
@{  
    ViewData["Title"] = "net centric class ACHS";  
}  
  
<div>  
    @{  
        var str = "Hello World";  
        <h3>@str</h3>  
    }  
</div>
```

**Example 5:** Use HTML Markup for adding HTML code inside Razor block.

```
@{  
    ViewData["Title"] = "net centric class ACHS";  
}  
  
<div>  
    @{  
        var str = "Hello World";  
        <h3>@str</h3>  
        <span>This is Text inside Razor.</span>  
    }  
</div>
```

**Example 6:** Use **Double @@**, if you want to print **single @**.

```
@{  
    ViewData["Title"] = "net centric class ACHS";  
}
```

```

<div>
    @{
        var price = 30;
        <h2>You can get 1GB Data @@ of @price</h2>
    }
</div>

```

### Example 7: Control Statement: if, else if and else

```

@{
    ViewData["Title"] = "net centric class ACHS";
}
<div>
    @{
        var price = 45;
        if (@price > 30)
        {
            <h3>Price is Higher than 30.</h3>
        }
        else if (@price == 30)
        {
            <h3>Price is Equal to 30.</h3>
        }
        else
        {
            <h3>Price is Lower than 30.</h3>
        }
    }
</div>

```

### Example 8: Switch Case

```

@{
    ViewData["Title"] = "net centric class ACHS";
}
<div>
    @{
        var Days = DateTime.Now.DayOfWeek.ToString();

        switch (Days)
        {
            case "Sunday": <strong>Its Holiday</strong>
                break;
            case "Monday": <strong>Oh no! It's Monday Again</strong>
                break;
            case "Tuesday": <strong>Go to Work. It's Tuesday</strong>
                break;
            case "Wednesday": <strong>Continue to Work. It's Wednesday</strong>
                break;
            case "Thursday": <strong>Continue to Work. It's Thursday</strong>
                break;
            case "Friday": <strong>Yeah! It's Friday.</strong>
                break;
            case "Saturday": <strong>Hurreyy... It's Saturday.</strong>
        }
    }
</div>

```



```

        break;
    default: <strong>Its Mysterious. I don't know the day name.</strong>
        break;
    }
}

</div>

```

### Example 9: Loop Statement: for, foreach, while, do while

```

@{
    ViewData["Title"] = "net centric class ACHS";
}
<div>
@{
    <h3>For Loop Example</h3>
    for (var i = 0; i < 5; i++)
    {
        <strong>@i </strong>
    }

    <h3>While Loop Example</h3>
    var j = 0;
    while (j < 5)
    {
        <strong>@j </strong>
        j++;
    }

    <h3>Do While Loop Example</h3>
    var k = 0;
    do
    {
        <strong>@k </strong>
        k++;
    }
    while (k < 5);
}

</div>

```

### Example 10: Foreach and Array

```

@{
    ViewData["Title"] = "net centric class ACHS";
}
<div>
@{
    int[] arr = { 1, 3, 5, 7, 11, 13, 17, 19 };
    foreach (int x in arr)
    {
        <span>@x, </span>
    }
}

```

```
</div>
```

### Example 11: Handling Errors with Try Catch

```
@{
    ViewData["Title"] = "net centric class ACHS";
}
<div>
    @{
        try
        {
            int a, b, result;
            a = 5;
            b = 0;
            result = a / b;
            <span>@result </span>
        }
        catch (Exception ex)
        {
            <span>@ex.ToString()</span>
        }
    }
</div>
```

### Example 12: @using : Add NameSpace

```
<div>
    @using System.IO;
    @using System.Data.SqlClient;
    @using System.Configuration;
    @{
        // Your Code Here.
    }
</div>
```

### Example 13: Comments in Razor

```
@{
    ViewData["Title"] = "net centric class ACHS";
}
@*
    @{
        /* C# comment */
        // Another C# comment
    }
    <!-- HTML comment -->
*@
```

#### Example 14: @model: Accessing Model data.

The **@model** command is used for adding and accessing model variables and data into the view page.

```
@model LoginViewModel
```

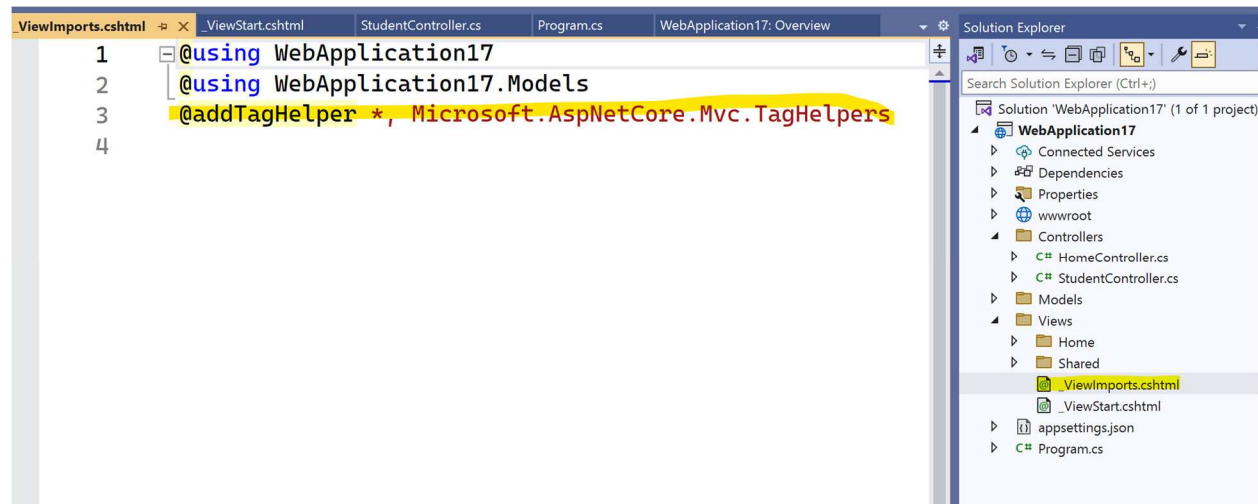
## 5. Tag Helper

Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files

### Advantage of Tag Helper:

- Tag Helpers use server-side binding without any server-side code.
- This helper object is very much use full when HTML developers do the UI designing who does not have any idea or concept about Razor syntax.
- It provides us an experience of working on basic HTML environments.
- It Supports rich IntelliSense environment support to create a markup between HTML and Razor
- To enable tag helper import `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`

In header section of .cshtml page or *you can find Inside view>Shared Folder with name \_viewImports.cshtml as shown in figure*



## Types of Tag Helpers in ASP.NET Core:

There are two types of Tag helpers in ASP.NET Core. They are as follows:

- **Built-In Tag Helpers:** They come in-built in the ASP.NET Core Framework and can perform common tasks like generating links, creating forms, loading assets, showing validation messages, etc.
- **Custom Tag Helper:** That can be created by us to perform our desired transformation on an HTML element.

## Inbuilt Tag Helper

### Form Tag Helper

- Form
- FormAction
- Input
- Label
- Option
- Select
- TextArea
- ValidationMessage
- ValidationSummary

```
<form asp-controller="Home" asp-action="Index" method="post">
    @* Validation Summary *@
    <div asp-validation-summary="All"></div>

    @* Label, Input, Validation(span) *@
    <input asp-for="Id" />

    <label asp-for="Firstname"></label>
    <input asp-for="Firstname" />
    <span asp-validation-for="Firstname"></span> <br />

    <label asp-for="BirthDate"></label>
```

```

<input asp-for="BirthDate" asp-format="{0:dd/MM/yyyy}" />
<span asp-validation-for="BirthDate"></span> <br />

@* Textarea *@
<label asp-for="Notes"></label>
<textarea asp-for="Notes"></textarea>
<span asp-validation-for="Notes"></span> <br />

@* Select (single) *@
<label asp-for="Title"></label>
<select asp-for="Title" asp-items="Model.GetTitles()">
    <option value="">--select--</option>
</select><br />

@* Select (multiple) *@
<label asp-for="Interests"></label>
<select asp-for="Interests" asp-items="Model.GetInterests()"></select><br />

@* Select (enum) *@
<label asp-for="Gender"></label>
<select asp-for="Gender" asp-items="Html.GetEnumSelectList<Gender>()">
    <option value="" selected>--select--</option>
</select><br />

<br />
<button type="submit">Save</button><br />
</form>

```

## Anchor Tag

```

<a asp-controller="Home" asp-action="Index" asp-route-id=@Model.Id>Index</a>

```

## 5.1 Model Binding and Validation

**Model Binding** is a process of ASP.NET Core framework to Extract Data from HTTP Requests and provide them to the arguments of Action Method

```

<form asp-action="Create">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

    <div class="form-group">
        <label asp-for="Fullname" class="control-label"></label>
        <input asp-for="Fullname" class="form-control" />
        <span asp-validation-for="Fullname" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Email" class="control-label"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Country" class="control-label"></label>

```

```

        <input asp-for="Country" class="form-control" />
        <span asp-validation-for="Country" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
        <h2>@ViewBag.Message</h2>
    </div>

</form>

```

Q. Write Steps to Create Custom tag Helper in asp.net Core

### 1. Create a Class for the Tag Helper:

Start by creating a class that derives from **TagHelper**. This class will contain the logic for your

```

using Microsoft.AspNetCore.Razor.TagHelpers;

[HtmlTargetElement("customtag")]
public class CustomTagHelper : TagHelper
{
    public override void Process(TagHelperContext context, TagHelperOutput
output)
    {
        output.TagName = "p";
        output.Attributes.SetAttribute("class", "custom-class");
        output.Content.SetHtmlContent("This is a custom tag helper content.");
    }
}

```

In this example, the custom tag helper is named **CustomTagHelper**. It targets elements with the tag name **<customtag>**.

### 2.Register the Tag Helper:

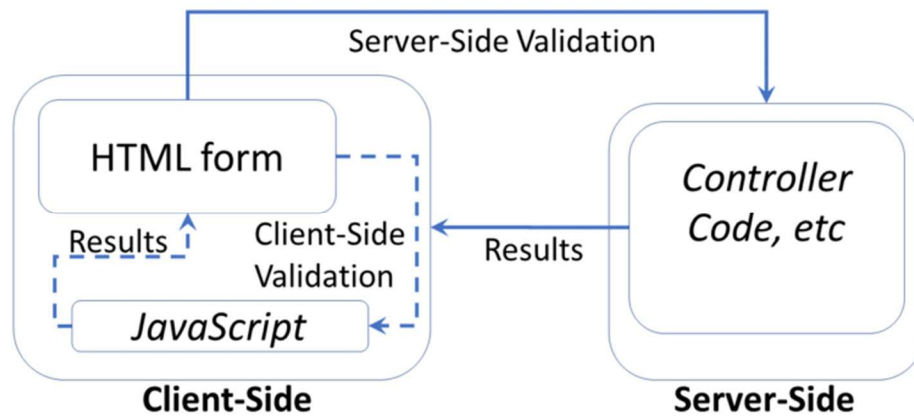
In your application, you need to register the custom Tag Helper. This is typically done in the **\_ViewImports.cshtml** file within the Views folder.

```
@addTagHelper *, WebApplication28
```

### 3. Use the Custom Tag Helper in Views:

```
<customtag></customtag>
```

## Validation in Asp.net Core: Two Types of validation:



### 1. Client-Side Validation

All the client-side validation in the world won't prevent a malicious user from sending a GET/POST request to your form's endpoint.

client-side validation helps to catch the problem before your server receives the request, while providing a better user experience.

```
<span asp-validation-for="Fullname" class="text-danger"></span>
```

#### Enabling Client Side Validation:

```
builder.Services.AddMvc().AddViewOptions(options =>
{
    options.HtmlHelperOptions.ClientValidationEnabled = true;
});
```

### 2. Server-Side Validation

Validation occurs before an MVC controller action (or equivalent handler method for Razor Pages) takes over. As a result, you should check to see if the validation has passed before continuing next steps.

```
[HttpPost]
```

```
public IActionResult Create(StudentViewModel stu)
{
    if (ModelState.IsValid)
    {
        ViewBag.Message = stu.Fullname + " : Record Submitted Successfully";
    }
    return View();
}
```

## URL Routing

ASP.NET Core controllers use the Routing middleware to match the URLs of incoming requests and map them to actions. Route templates:

- Are defined in startup code or attributes.
- Describe how URL paths are matched to **actions**.
- Are used to generate URLs for links. The generated links are typically returned in responses.

## 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
- Attribute Routing

## Conventional Routing

When we create a new ASP.NET Core MVC application using the default template, the application configures a default routing.



After creating a new project with the default ASP.NET Core MVC template, the **program.cs** class. We can see that the application has configured a default routing using

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

## WEB API

ASP.NET Core supports creating RESTful services, also known as web APIs, using C#. To handle requests, a web API uses controllers. *Controllers* in a web API are classes that derive from *ControllerBase*. This article shows how to use controllers for handling web API requests.

Due to the increasing number of different varieties of clients (mobile apps, browser-based SPAs, desktop apps, IOT apps, etc.), we need better ways for transferring data from servers to clients, independent of technology and server stacks.

REST APIs(Representational state transfer) solve this problem. REST stands for representational state transfer. REST APIs are HTTP-based and provide applications the ability to communicate using lightweight JSON format. They run on web servers.

**Resource:** Resources are uniquely identifiable entities (for example: data from a database, images, or any data).

**Endpoint:** A resource can be accessed through a **URL** identifier.

**HTTP method:** HTTP method is the type of request a client sends to a server. Operations we perform on the resource should follow this.

**HTTP header:** An HTTP header is a key-value pair used to share additional information between a client and server, such as:

- Type of data being sent to server (JSON, XML).
- Type of encryption supported by client.

- Authentication-related token.
- Customer data based on application need.

**Data format:** JSON is a common format to send and receive data through REST APIs.

## JWT Token

JWT stands for JSON Web Token. It is open standard and defines a better way for transferring data securely between two entities (client and server).

The data transmitted using JWT between parties are digitally signed so that it can be easily verified and trusted.

A JWT helps the resource server verify the token data using the same secret key, so that you can trust the data.

JWT consists of the following three parts:

**Header:** encoded data of token type and the algorithm used to sign the data.

**Payload:** encoded data of claims intended to share.

**Signature:** created by signing (encoded header + encoded payload) using a secret key.

## Dependency Injection and IOC containers

Dependency Injection (DI) is a **pattern** and IoC container is a **framework**.

Dependency Injection (often called just DI) is a software design pattern that helps us create **loosely coupled** applications. It is an **implementation of the Inversion of Control (IoC)** principle, and Dependency Inversion Principle (D in SOLID).

**IoC** is a **design principle** which recommends the inversion of different kinds of controls in object-oriented design to achieve **loose coupling** between application classes. In this case, control refers to any additional responsibilities a class has, other than its main responsibility,

such as control over the flow of an application, or control over the dependent object creation and binding (Remember SRP - Single Responsibility Principle). If you want to do **TDD** (Test Driven Development), then **you must use the IoC principle**, without which **TDD is not possible**

**Dependency Injection** (DI) is a design pattern which implements the IoC principle to invert the creation of dependent objects. Dependency Injection (DI) is a design pattern used to implement IoC. It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways.

The Dependency Injection pattern involves 3 types of classes.

- **Client Class:** The client class (dependent class) is a class which depends on the service class
- **Service Class:** The service class (dependency) is a class that provides service to the client class.
- **Injector Class:** The injector class injects the service class object into the client class.

The IoC container is a framework used to manage automatic dependency injection throughout the application, so that we as programmers do not need to put more time and effort into it.