

Module-5

25. Introduction to WEB Dev.

ASP.NET:

It is a web framework designed and developed by Microsoft. It is used to develop websites, web applications and web services. It provides fantastic integration of HTML, CSS and JavaScript. It was first released in January 2002. It is built on the Common Language Runtime (CLR) and allows programmers to write code using any supported .NET language.

ASP.NET provides three development styles for creating web applications:

1. Web Forms
2. ASP.NET MVC
3. ASP.NET Web Pages

Web Forms

It is an event driven development framework. It is used to develop application with powerful data access. It provides server side controls and events to create web application. It is part of the ASP.NET framework.

ASP.NET MVC

It gives us a MVC (Model View Controller), patterns-based way to build dynamic websites. It enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development. It also provides many features that enable fast development for creating outstanding applications.

ASP.NET Web Pages

It is used to create dynamic web pages. It provides fast and lightweight way to combine server code with HTML. It helps to add video, link to the social sites. It also provides other features like you can create beautiful sites that conform to the latest web standards.

All these are stable and well equipped frameworks. We can create web applications with any of them. These are also based on the .NET Framework and share core functionalities of .NET and ASP.NET.

We can use any development style to create application. The selection of style is depends on the skills and experience of the programmer.

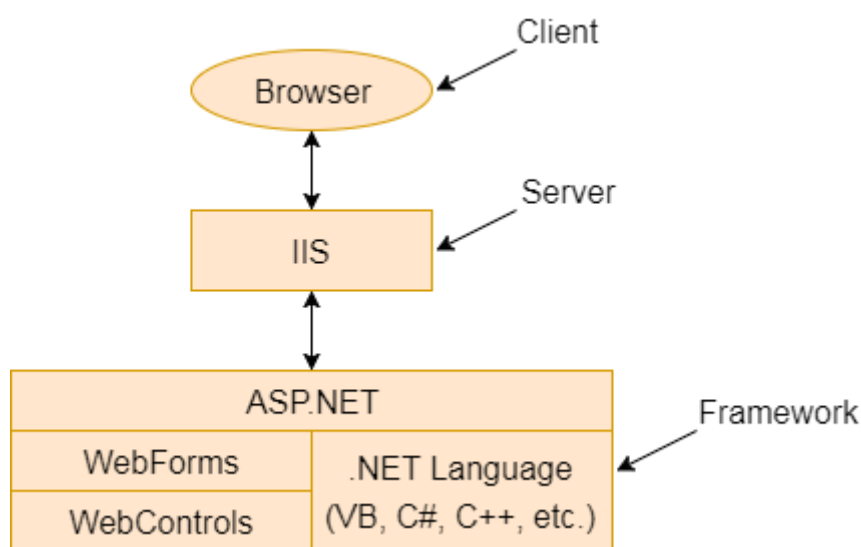
Although each framework is independent to other, we can combine and use any of that at any level of our application. For example, to develop client interaction module, we can use MVC and for data control, we can use Web Forms.

ASP.NET Web Form:

Web Forms are web pages built on the ASP.NET Technology. It executes on the server and generates output to the browser. It is compatible to any browser to any language supported by .NET common language runtime. It is flexible and allows us to create and add custom controls.

We can use Visual Studio to create ASP.NET Web Forms. It is an IDE (Integrated Development Environment) that allows us to drag and drop server controls to the web forms. It also allows us to set properties, events and methods for the controls. To write business logic, we can choose any .NET language like: Visual Basic or Visual C#.

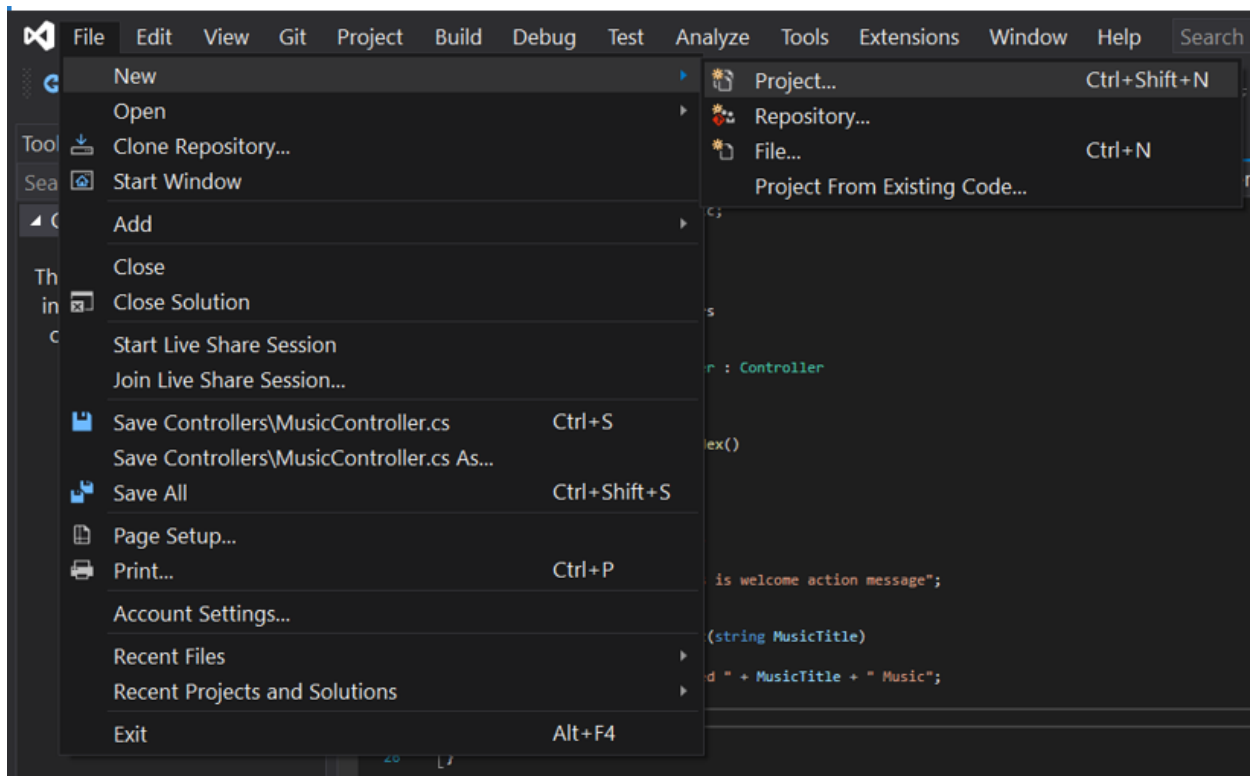
Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.



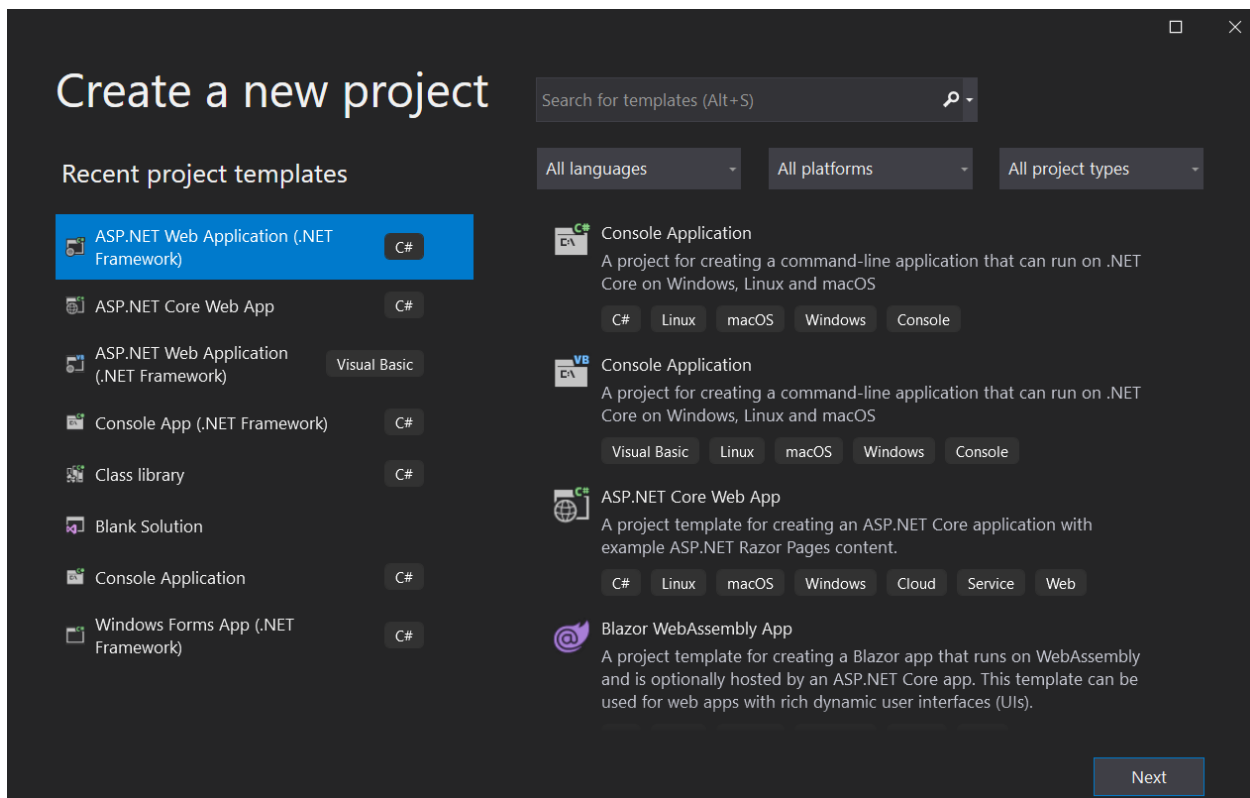
This diagram shows the components of the ASP.NET.

Web Form Project:

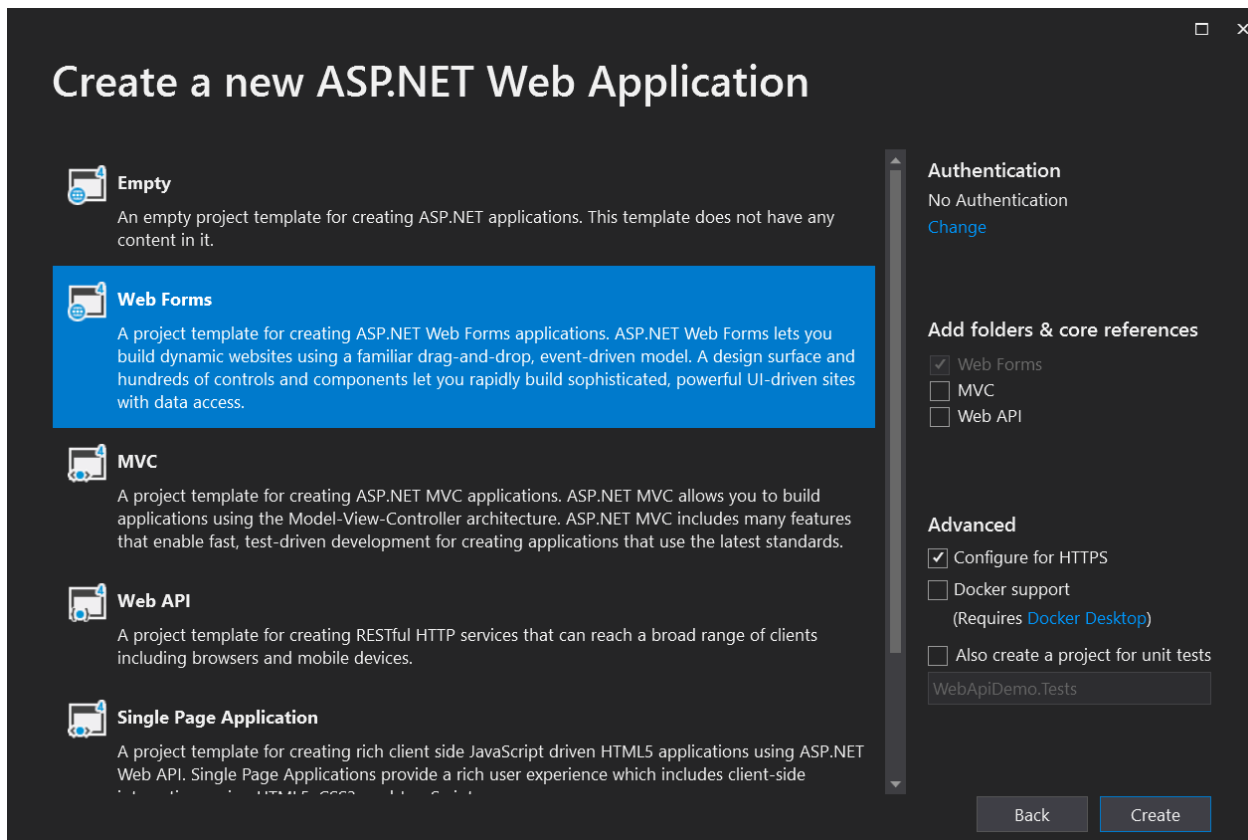
1. Creating a new project: Click on the file menu from the menu bar and select **new -> project**.



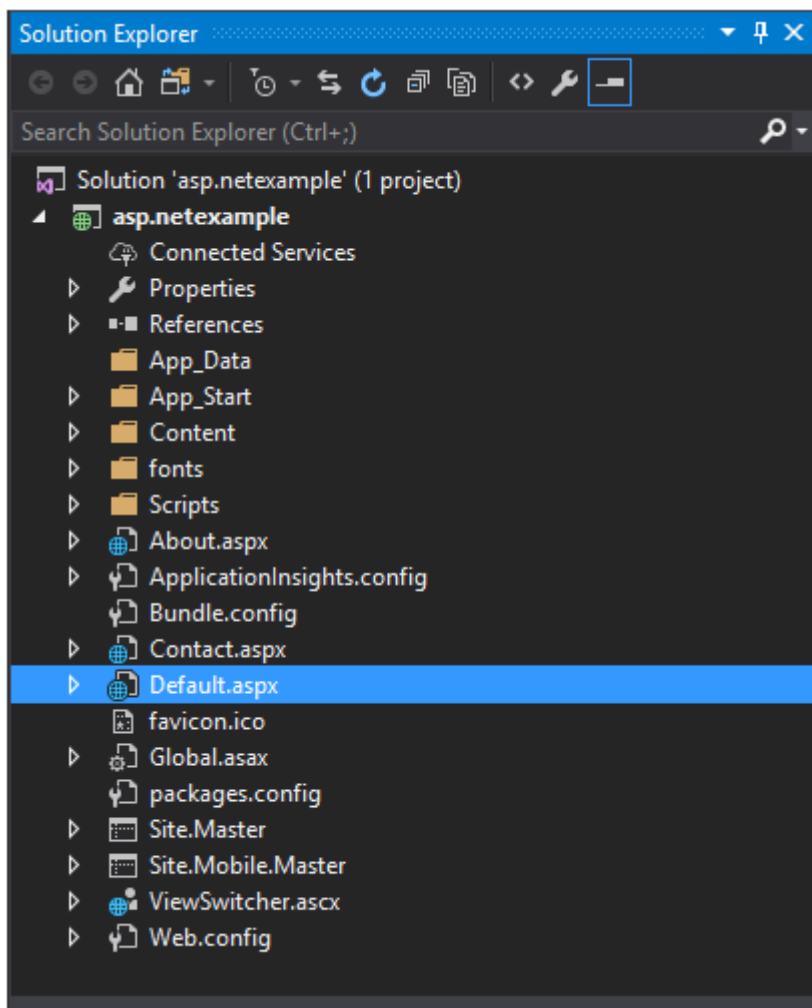
Select Project type: It provides couple of choices but we selecting ASP.NET Web Application.



Select Project Template: After selecting project types, now, it asks for the type of template that we want to implement in our application. Here, we are selecting Web Forms as because we are creating a Web Forms application.

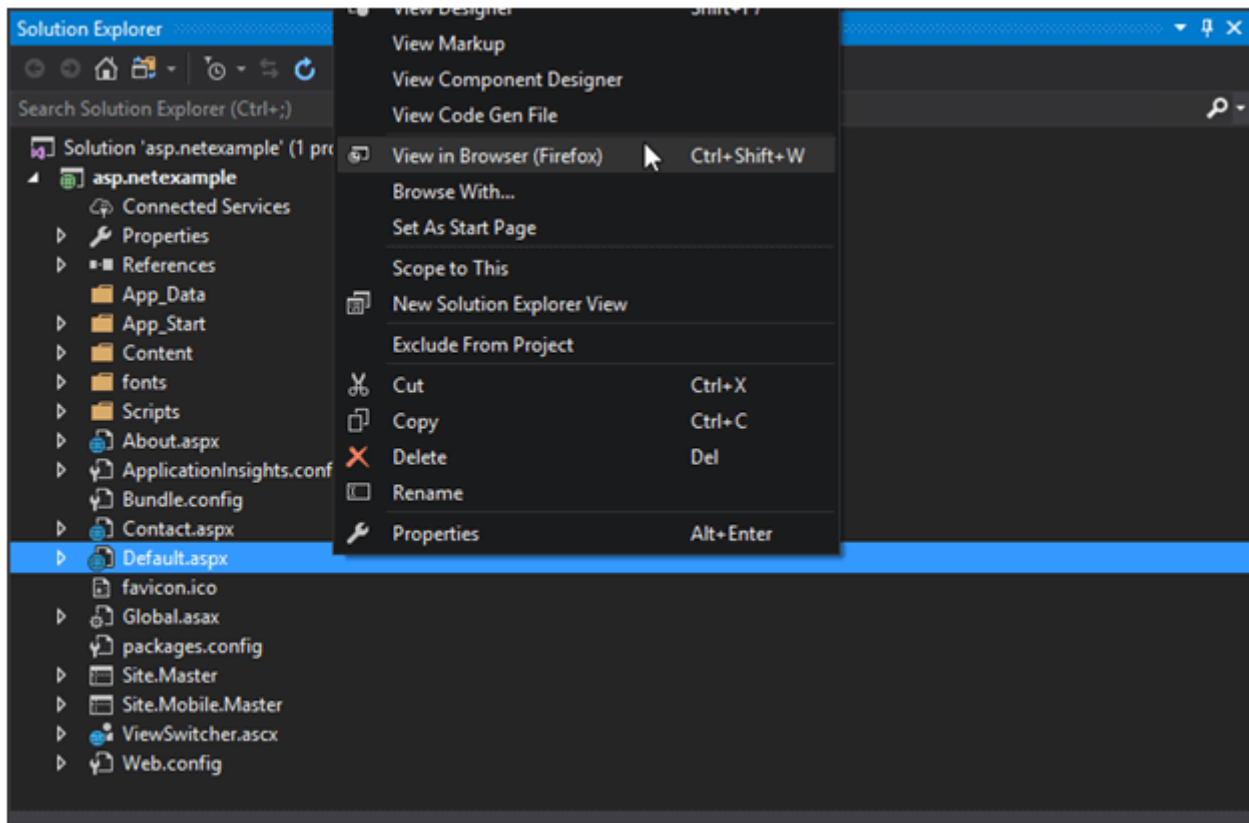


After clicking ok, it shows project in **solution explorer** window that looks like the below.

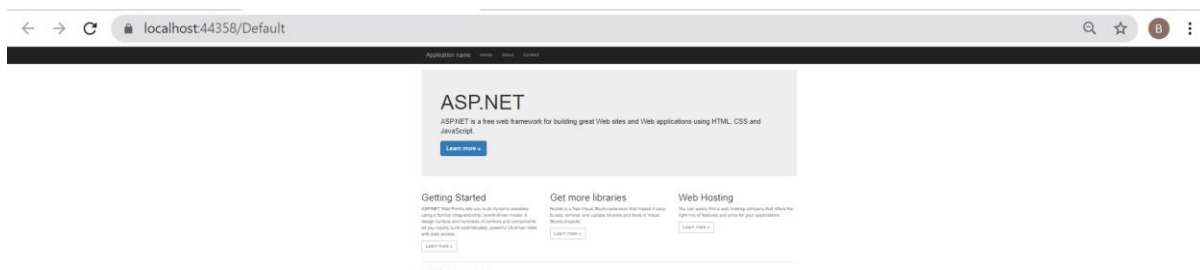


This project contains a **default.aspx** file which is a startup file. When we run the project this file executes first and display a home page of the site.

We can see its output on the browser by selecting **view in browser** option as we did below.

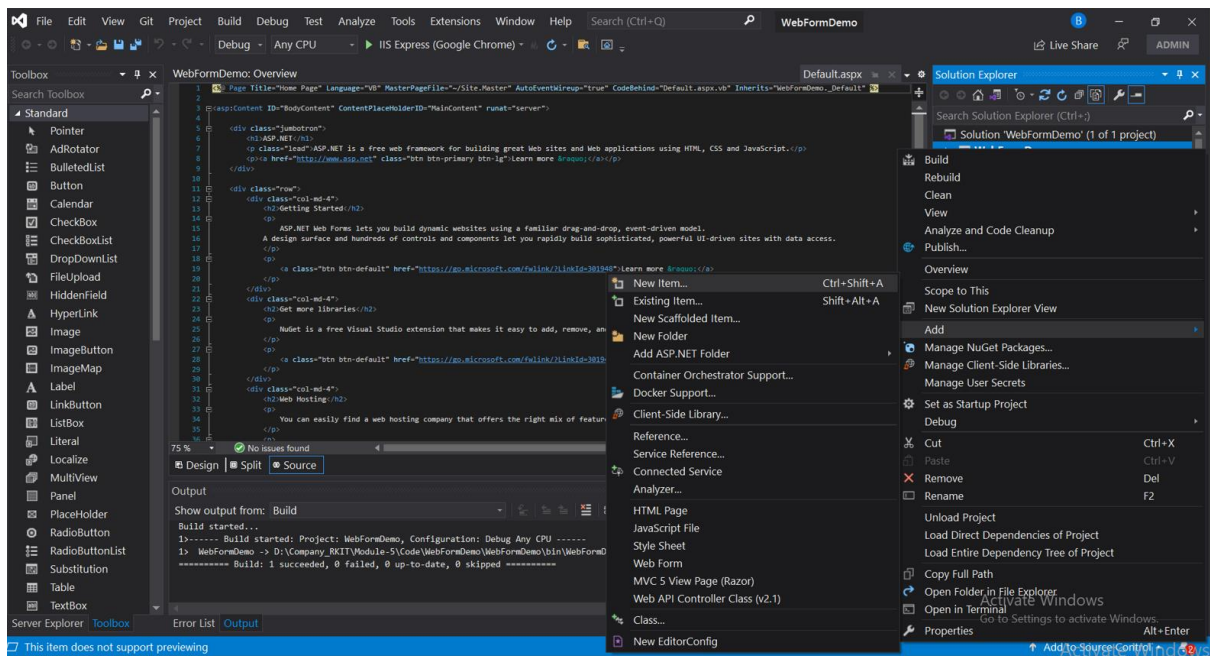


Finally, it shows output in the browser like this:

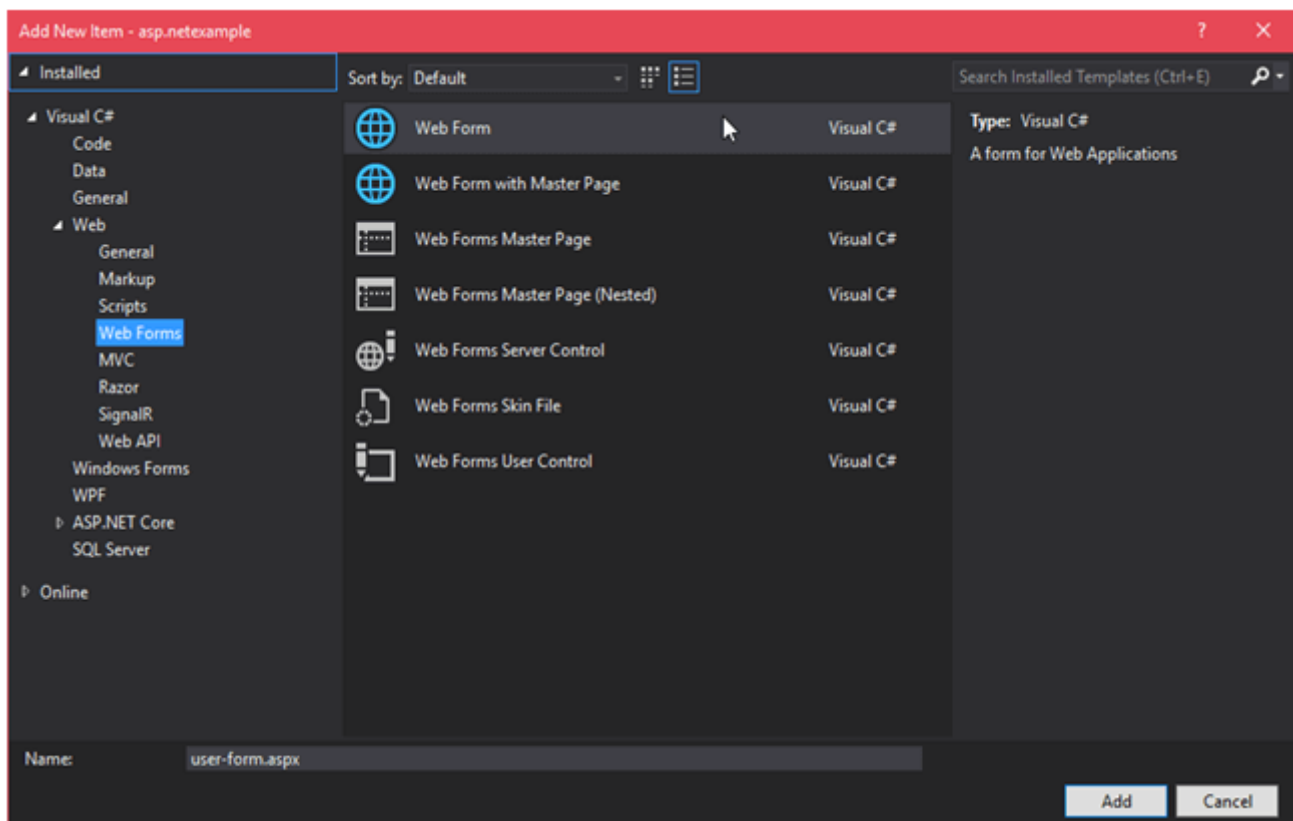


Create a New Web Form:

To add a new web form in our existing project, first select project then right click and add new item.

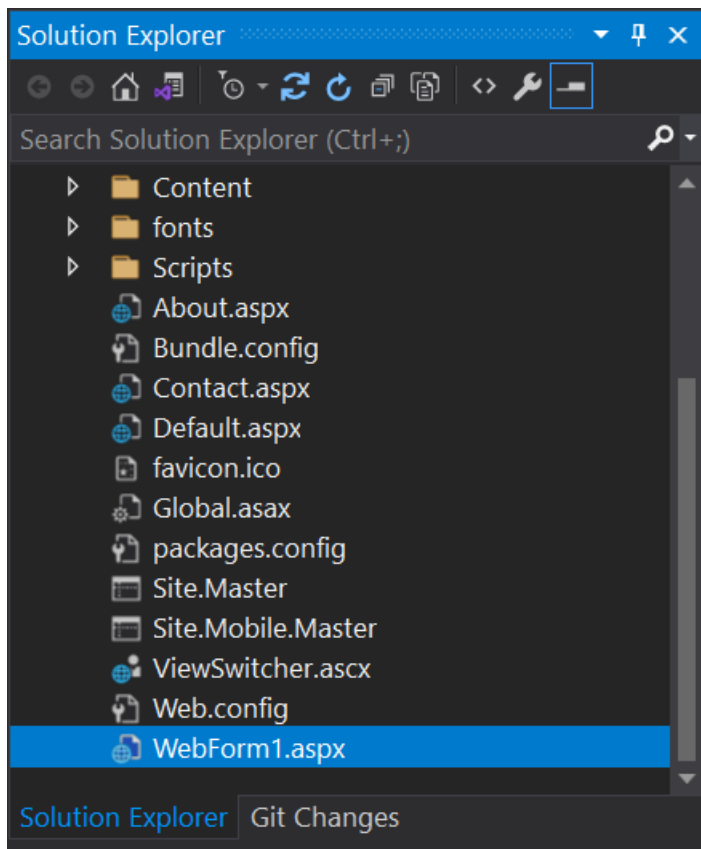


Select web forms option in left corner and then select web form and hit add button.

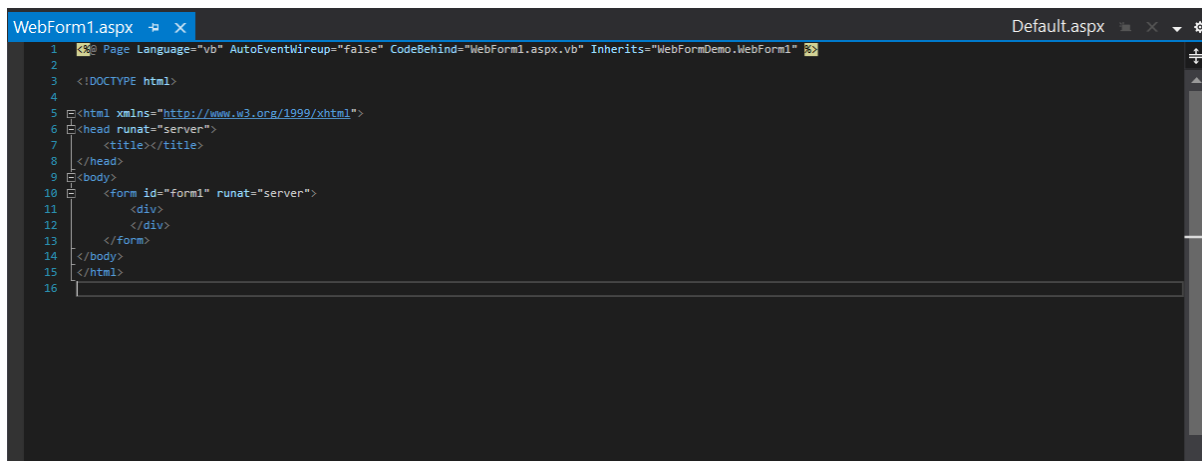


Now click on the add button and this form will add to our project.

After adding form, we can see that this is now in our project as we have shown in the below image.



Double click on this form and this will show some auto generated code like this:



If we run this file on the browser, it does not show any output. So, let's print some message by this form.

The modified code is as below.

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="WebForm1.aspx.vb" Inherits="WebFormDemo.WebForm1" %>
```

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
```

```
<title></title>
```

```
</head>
```

```
<body>
```

```
<form id="form1" runat="server">
```

```
<div>
```

```
<h2>Welcome to the Web Forms!</h2>
</div>
</form>
</body>
</html>
```

After running it on the browser it yields the following output.



ASP.NET MVC:

The MVC (Model-View-Controller) is an application development pattern or design pattern which separates an application into three main components:

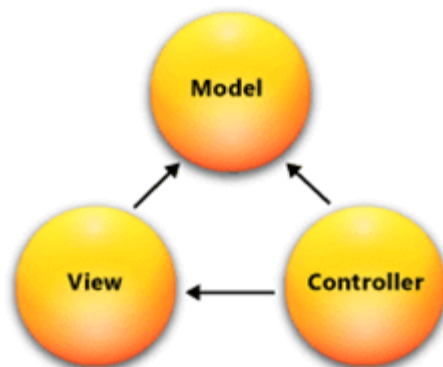
1. Model
2. View
3. Controller

Model: Model is a part of the application which implements the logic for the data domain of the application. It is used to retrieve and store model state in a database such as SQL Server database. It also used for business logic separation from the data in the application.

View: View is a component that forms the application's user interface. It is used to create web pages for the application. An example would be an edit view of a Products table that displays text boxes, drop-down lists and check boxes based on the current state of a Product object.

Controller: Controller is the component which handles user interaction. It works with the model and selects the view to render the web page. In an MVC application, the view only displays information whereas the controller handles and responds to the user input and requests.

The following image represents the ASP.NET MVC Design Pattern:

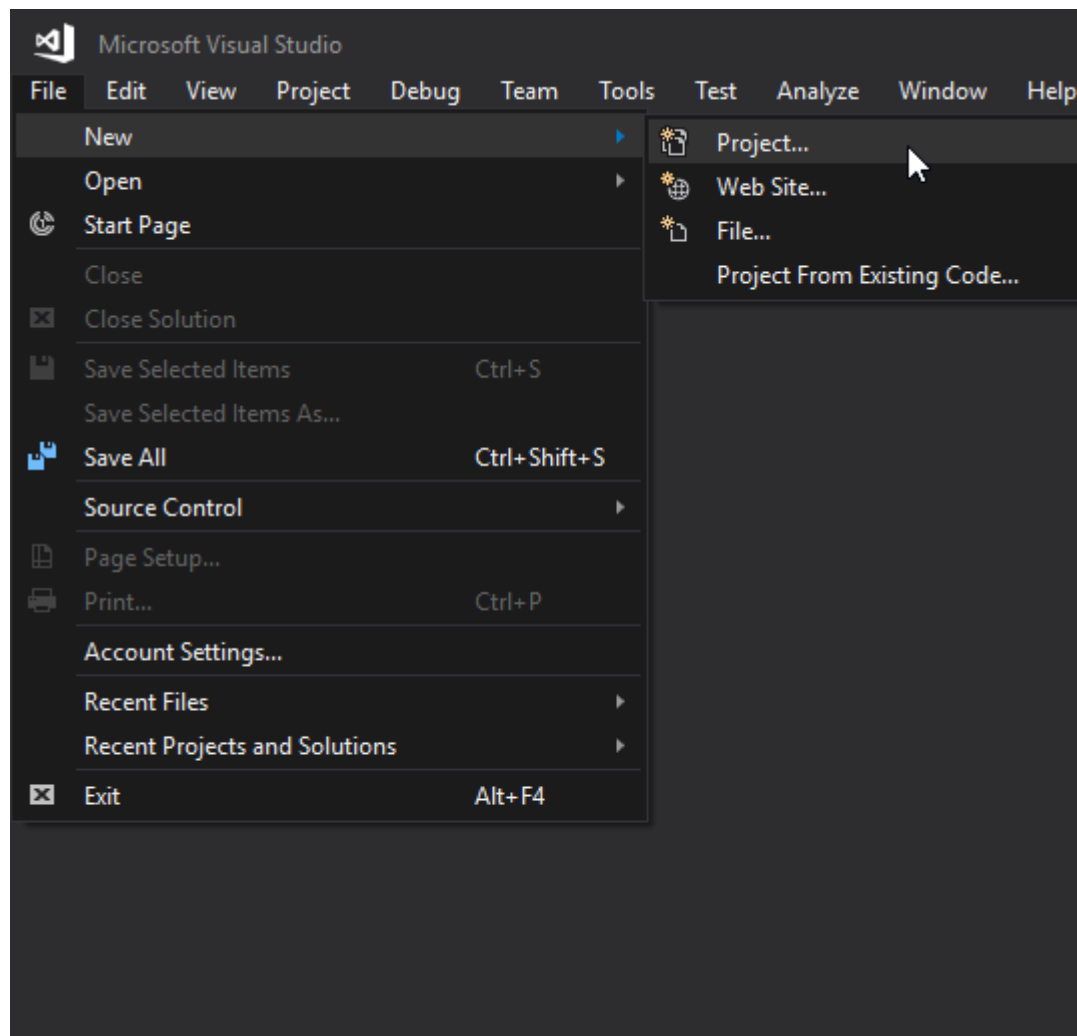


This design pattern is a lightweight framework which is integrated with various features such as master pages and membership based authentication. It is defined in the System.Web.Mvc assembly.

ASP.NET MVC Project:

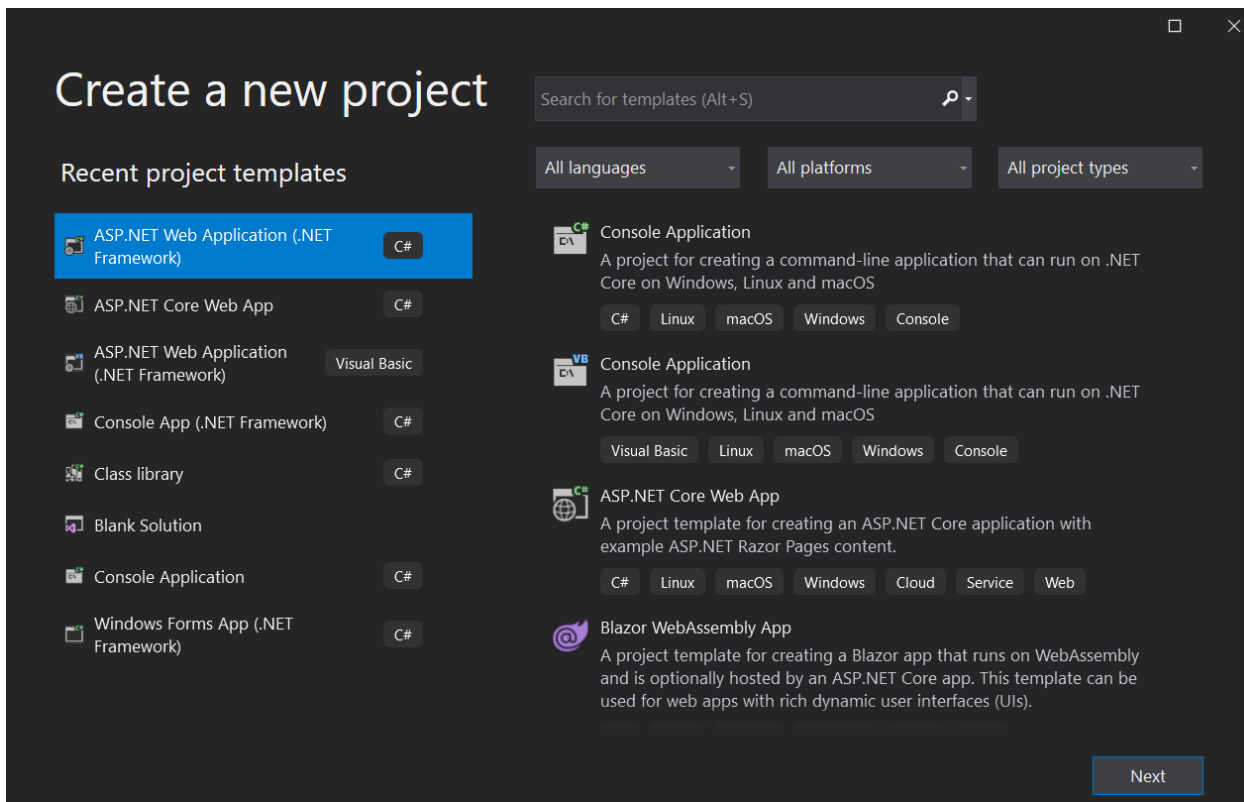
It includes the various steps that are given below. These following steps explain how to create MVC based web application.

1. **Create a Web Project:** Click on file menu from the menu bar and select new submenu to create a new project. The following image show that how to create a new project.



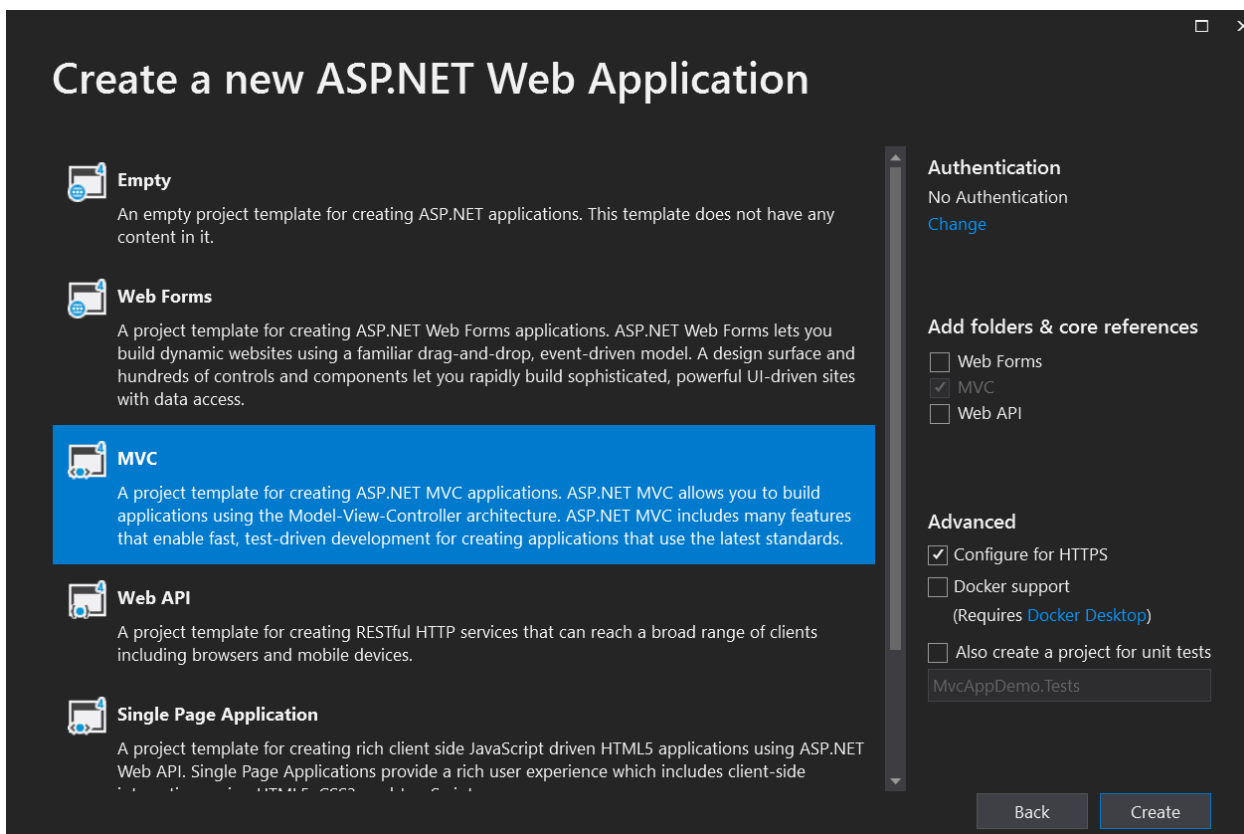
2. Select Project Type

Here, select type of project as a web project and provide name of the project.



3. Select MVC template

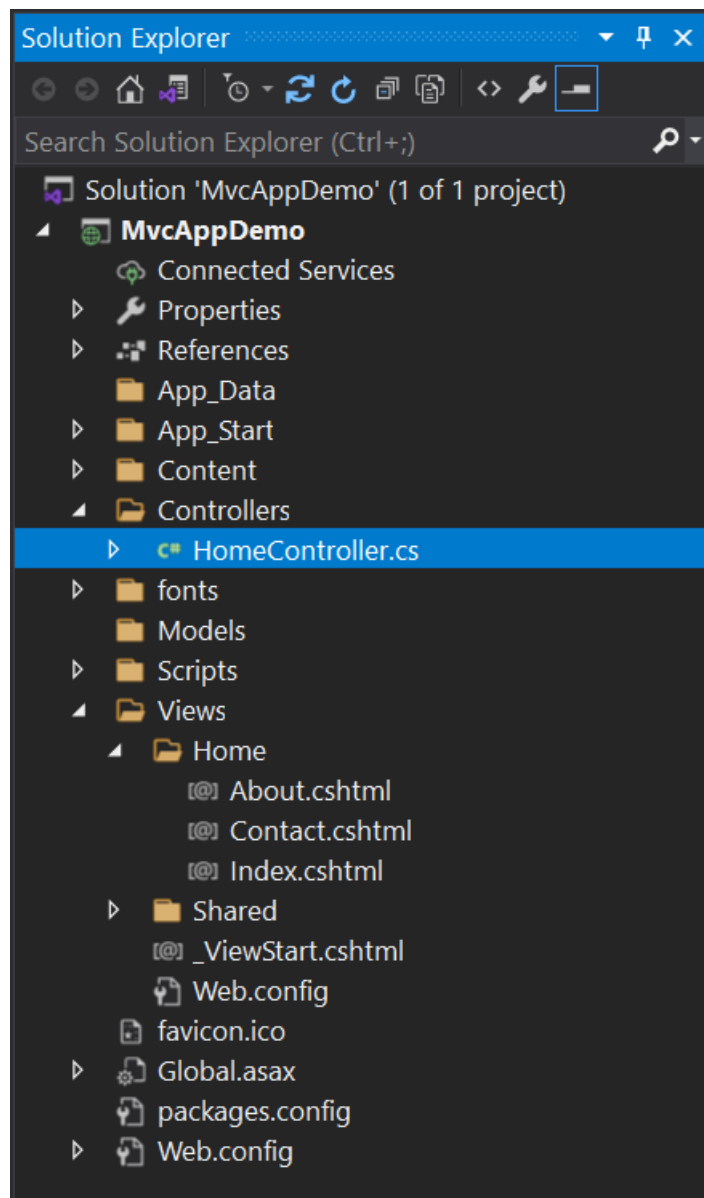
After selecting project type, now select the web template that we want to implement. Since we are working on MVC then select MVC template from the list of available template. At the same time, provide the authentication type to the application.



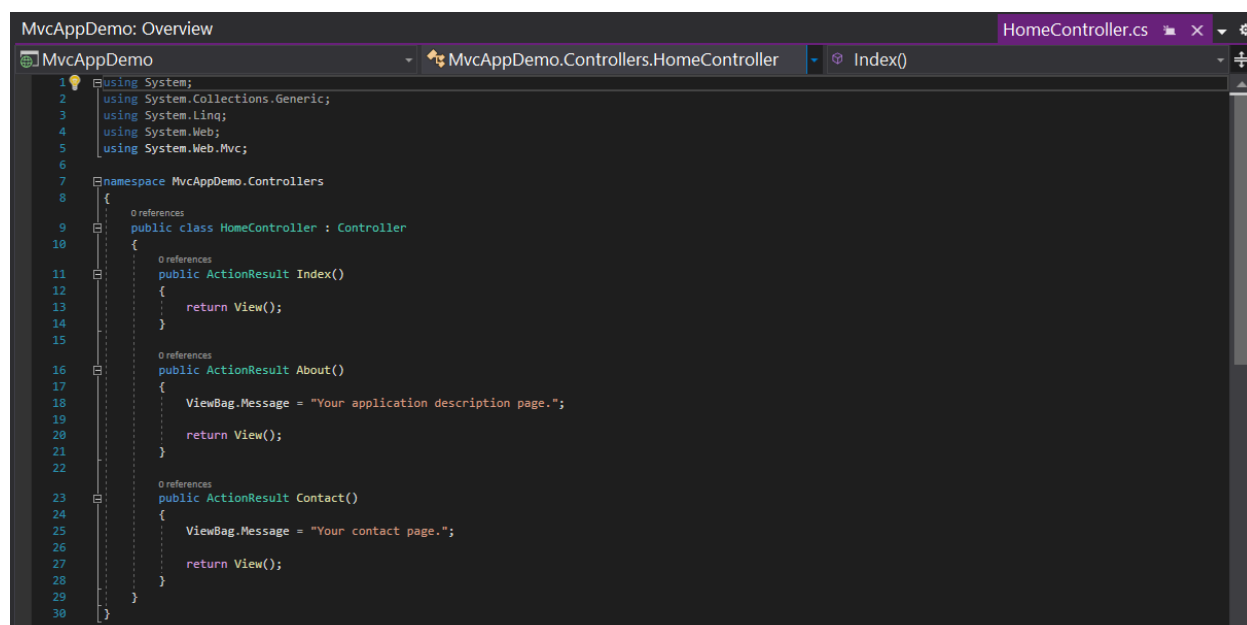
After clicking ok, it creates a project that has following structure:

4. MVC Web Application Project Structure

Following is the project structure that we just created.



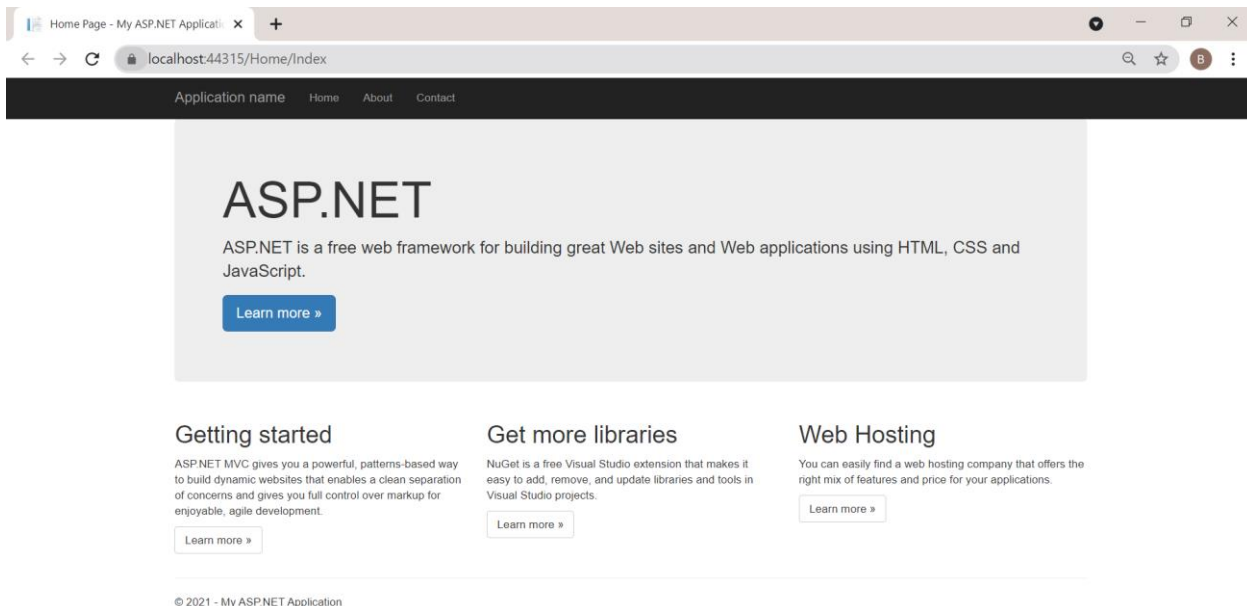
See, the project carefully, it contains three folders named **Model**, **View** and **Controller**. The **HomeController** is default controller for the application. This controller contains the following code:



Index file is default for the home controller in the view folder.

```
MvcAppDemo: Overview
Index.cshtml
1  ViewBag.Title = "Home Page";
2
3
4
5  <div class="jumbotron">
6      <h1>ASP.NET</h1>
7      <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>
8      <p><a href="https://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
9  </div>
10
11 <div class="row">
12     <div class="col-md-4">
13         <h2>Getting started</h2>
14         <p>
15             ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that
16             enables a clean separation of concerns and gives you full control over markup
17             for enjoyable, agile development.
18         </p>
19         <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301865">Learn more &raquo;</a></p>
20     </div>
21     <div class="col-md-4">
22         <h2>Get more libraries</h2>
23         <p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.</p>
24         <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301866">Learn more &raquo;</a></p>
25     </div>
26     <div class="col-md-4">
27         <h2>Web Hosting</h2>
28         <p>You can easily find a web hosting company that offers the right mix of features and price for your applications.</p>
29         <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301867">Learn more &raquo;</a></p>
30     </div>
31 </div>
```

Output:

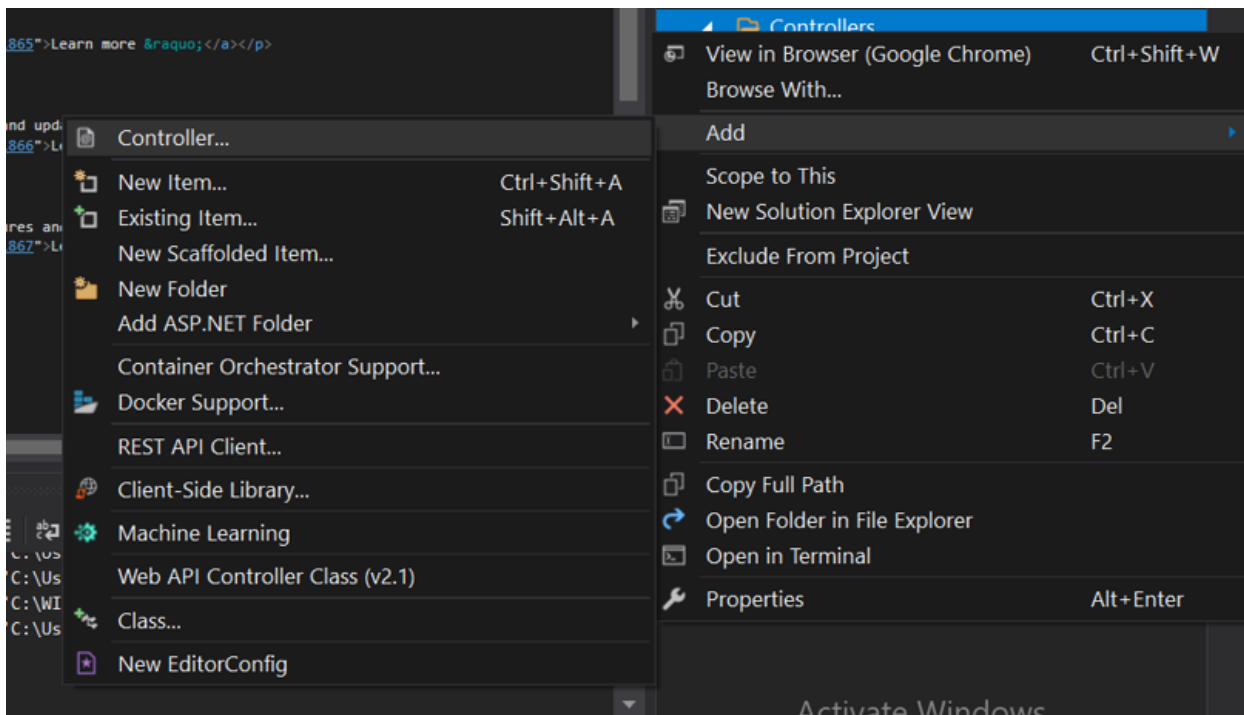


ASP.NET MVC Controller:

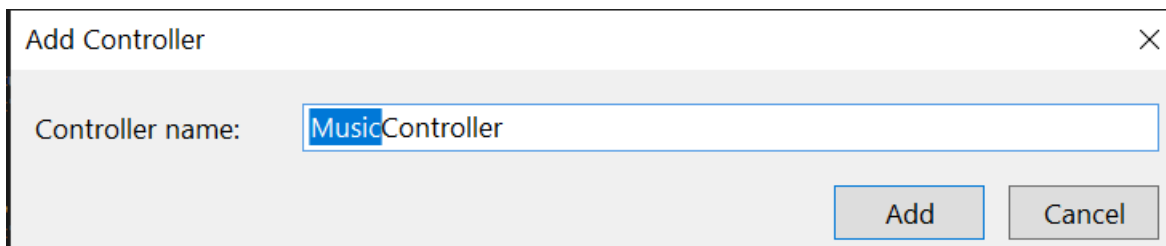
Controller is a class that handles user requests. It retrieves data from the Model and renders view as response.

Create a Controller:

We can create controller for the application by adding a new item into the controller folder. Just right click on the controller folder and click **add -> controller** as given below.

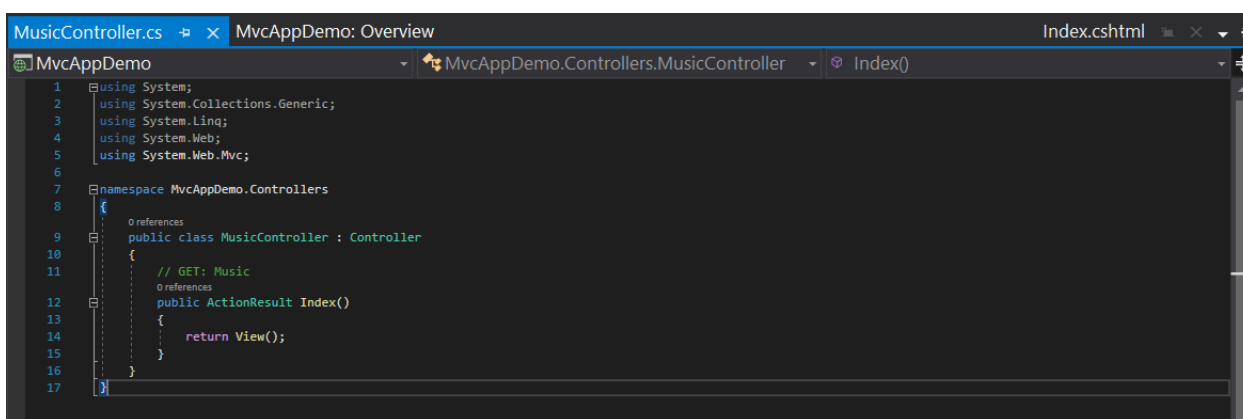


Providing controller name, then click Add.

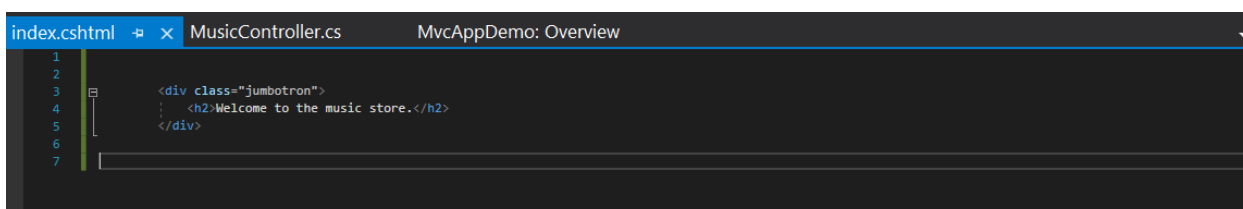


After adding this controller, as per the conventions project will create a folder with the same name as the controller name in the view folder to store the view files belongs to the controller.

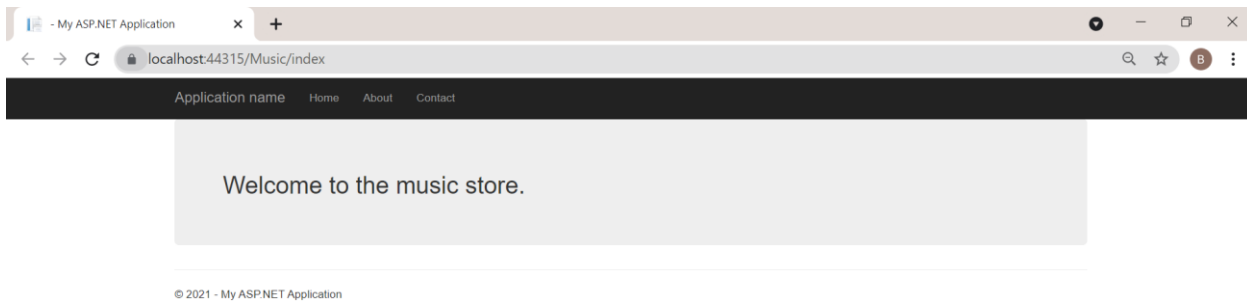
This controller contains the default code like below.



To access this controller to the browser, we are adding an index file to the Music folder inside the view folder. This index file contains the following code.

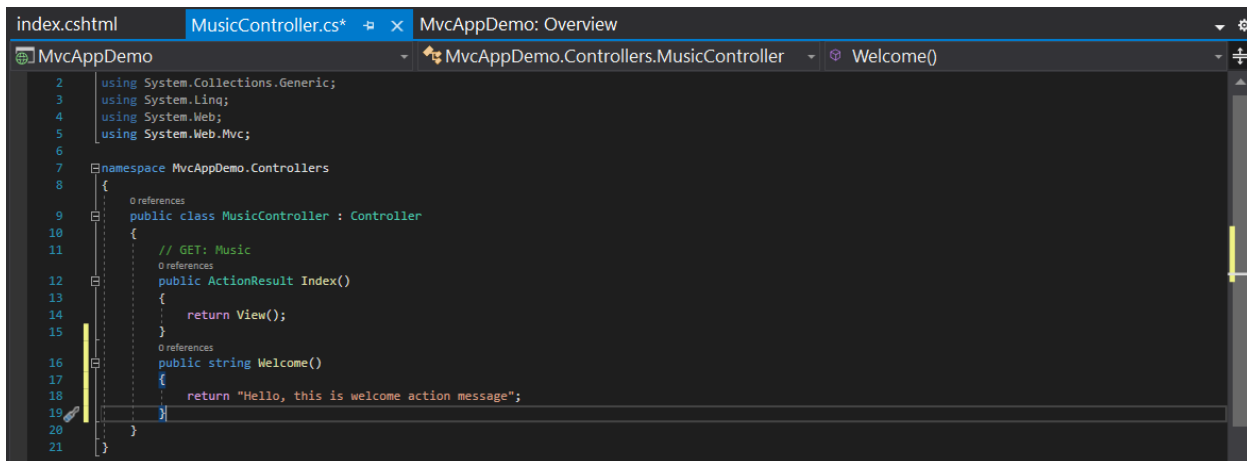


Output:



Adding an Action method:

To add action to the existing controller, we need to define a public method to our controller. Our **MusicController.cs** file is looks like the following after adding a welcome action method.



Output:

To access the welcome action method, execute the application then access it by using **Music/Welcome** URL. It will produce the following output.

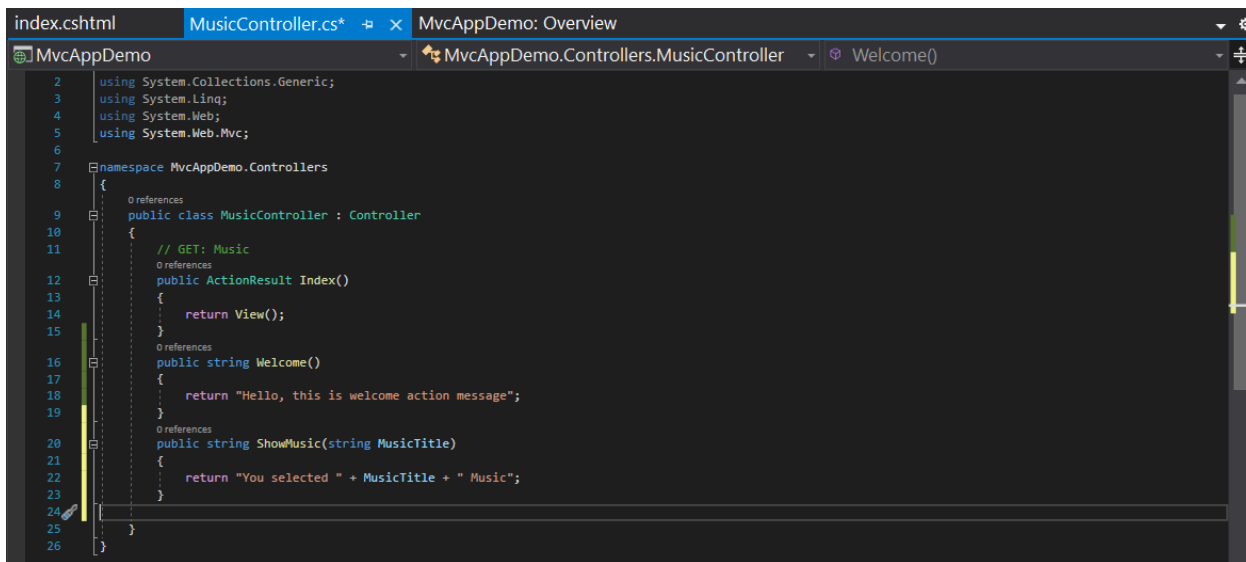


Action Method Parameters:

Action parameters are the variables that are used to retrieve user requested values from the URL.

The parameters are retrieved from the request's data collection. It includes name/value pairs for form data, query string value etc. the controller class locates for the parameters values based on the RouteData instance. If the value is present, it is passed to the parameter. Otherwise, exception is thrown.

Here, we are creating an action method in the controller. This action method has a parameter. The controller code looks like this:



```
1  index.cshtml | MusicController.cs* | MvcAppDemo: Overview
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6
7  namespace MvcAppDemo.Controllers
8  {
9      0 references
10     public class MusicController : Controller
11     {
12         // GET: Music
13         0 references
14         public ActionResult Index()
15         {
16             return View();
17         }
18         0 references
19         public string Welcome()
20         {
21             return "Hello, this is welcome action message";
22         }
23         0 references
24         public string ShowMusic(string MusicTitle)
25         {
26             return "You selected " + MusicTitle + " Music";
27         }
28     }
29 }
```

Output:

In URL, we have to pass the parameter value. So, we are doing it by this URL **localhost:port-no/Music/ShowMusic?MusicTitle=Classic**. It produces the following result.



Rest Web API:

API:

Before we understand what is Web API, let's see what is an API (Application Programming Interface).

As per Definition of API: In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building software and applications.

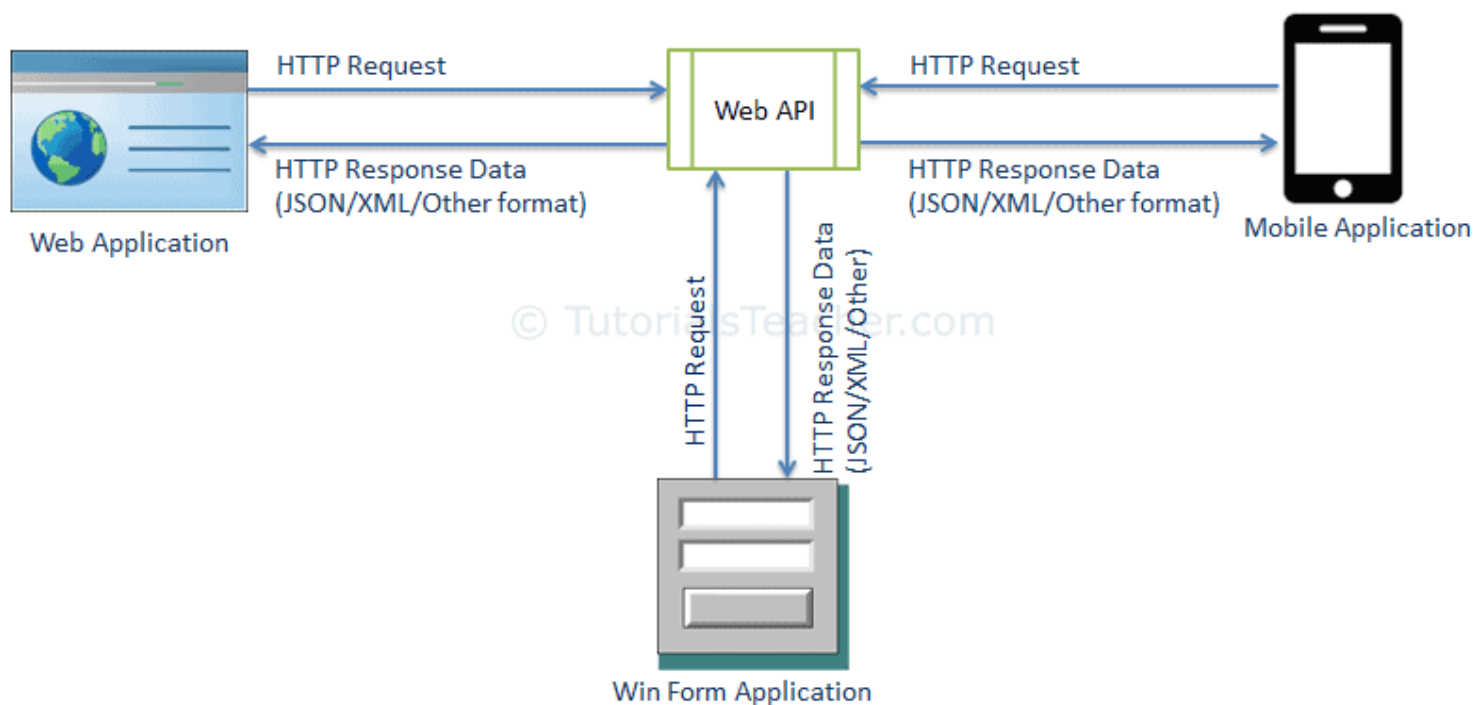
To put it in simple terms, API is some kind of interface which has a set of functions that allow programmers to access specific features or data of an application, operating system or other services.

Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol. It is a concept and not a technology. We can build Web API using different technologies such as Java, .NET etc.

ASP.NET Web API:

The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc. It works more or less the same way as ASP.NET MVC web application except that it sends data as a

response instead of html view. It is like a webservice or WCF service but the exception is that it only supports HTTP protocol.



Rest Web API:

An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST(REpresentational State Transfer) API is an API that conforms to the design principles of the REST, or representational state transfer architectural style. For this reason, REST APIs are sometimes referred to RESTful APIs.

How REST API Work?

REST APIs communicate via HTTP requests to perform standard database functions like creating, reading, updating, and deleting records (also known as CRUD) within a resource. For example, a REST API would use a GET request to retrieve a record, a POST request to create one, a PUT request to update a record, and a DELETE request to delete one. All HTTP methods can be used in API calls. A well-designed REST API is similar to a website running in a web browser with built-in HTTP functionality.

The state of a resource at any particular instant, or timestamp, is known as the resource representation. This information can be delivered to a client in virtually any format including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text. JSON is popular because it's readable by both humans and machines—and it is programming language-agnostic.

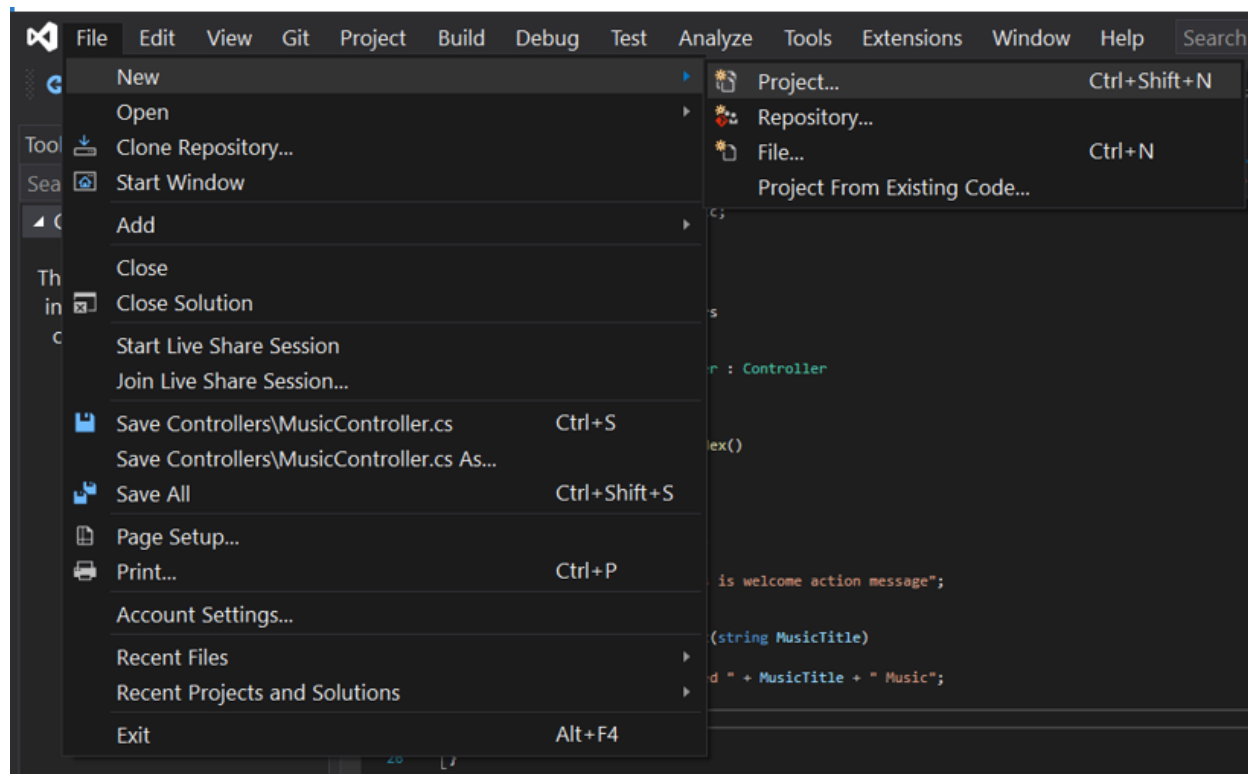
Request headers and parameters are also important in REST API calls because they include important identifier information such as metadata, authorizations, uniform resource identifiers

(URIs), caching, cookies and more. Request headers and response headers, along with conventional HTTP status codes, are used within well-designed REST APIs.

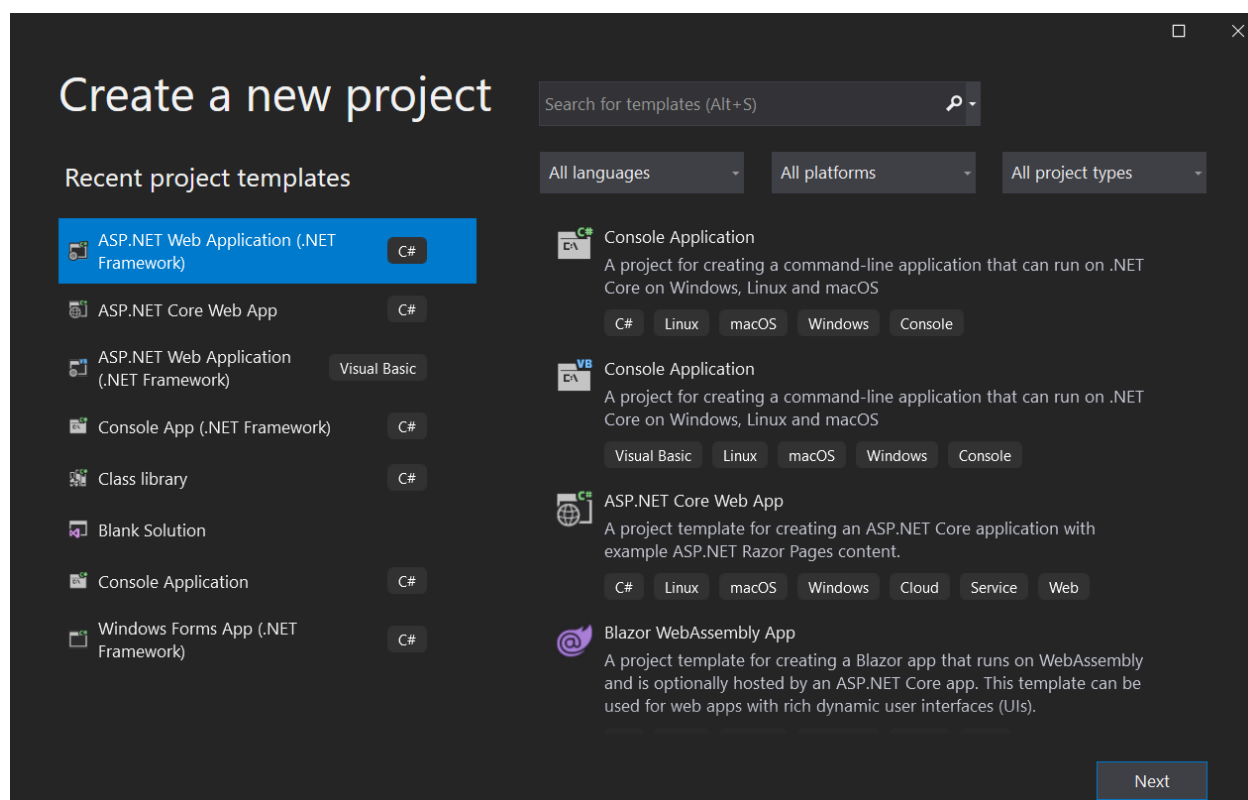
26. Start With Project.

- **Create New Web API Project:**

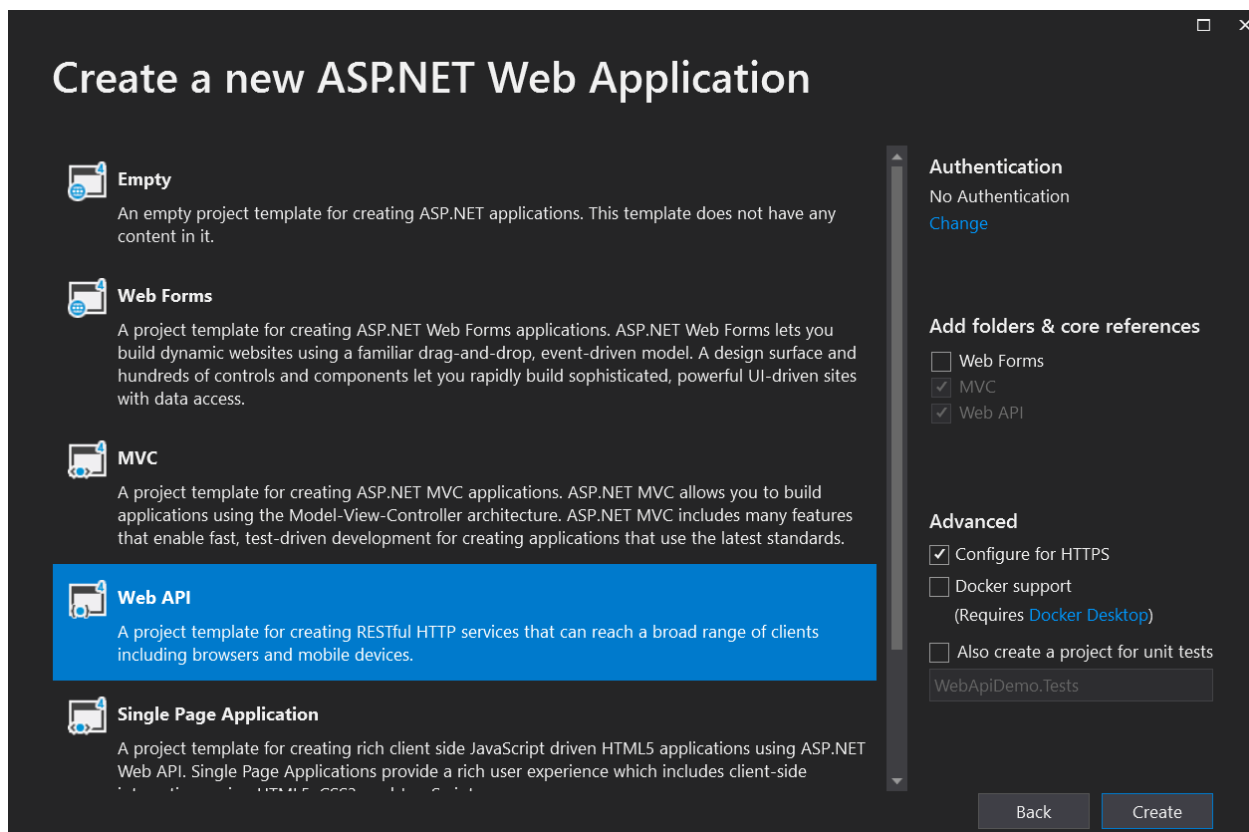
Creating a new project: Click on the file menu from the menu bar and select **new -> project**.



Select Project type: It provides couple of choices but we selecting ASP.NET Web Application.

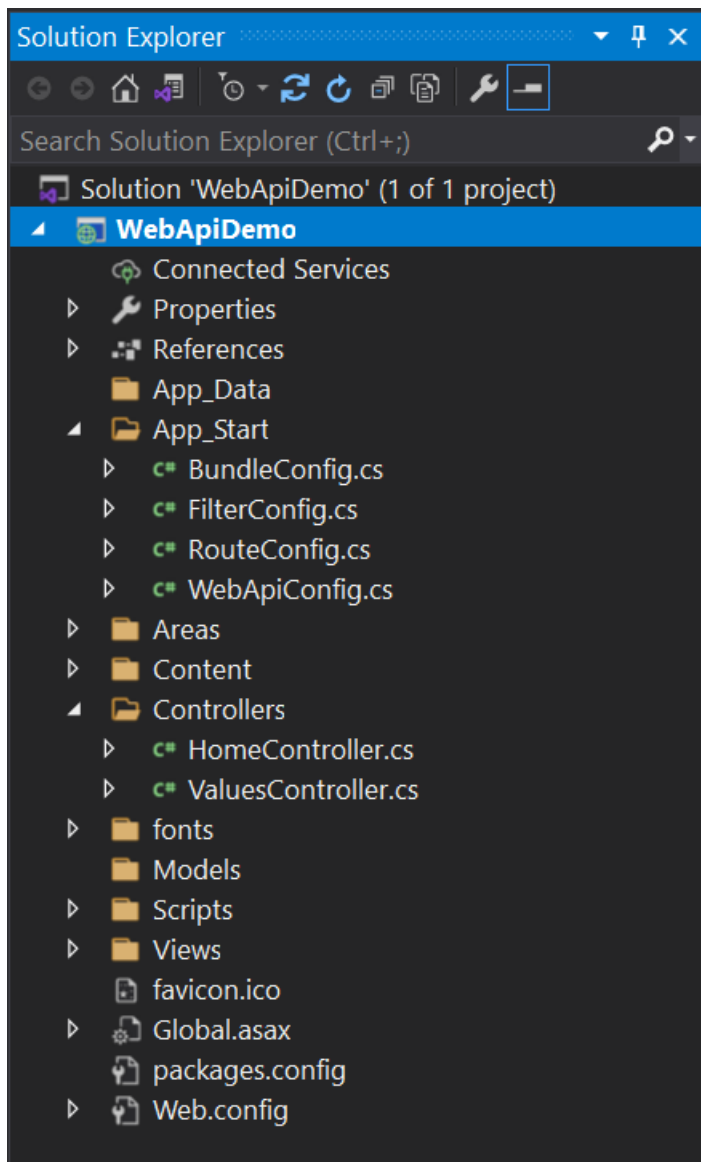


Select Project Template: After selecting project types, now, it asks for the type of template that we want to implement in our application. Here, we are selecting Web API as because we are creating a Web API application.

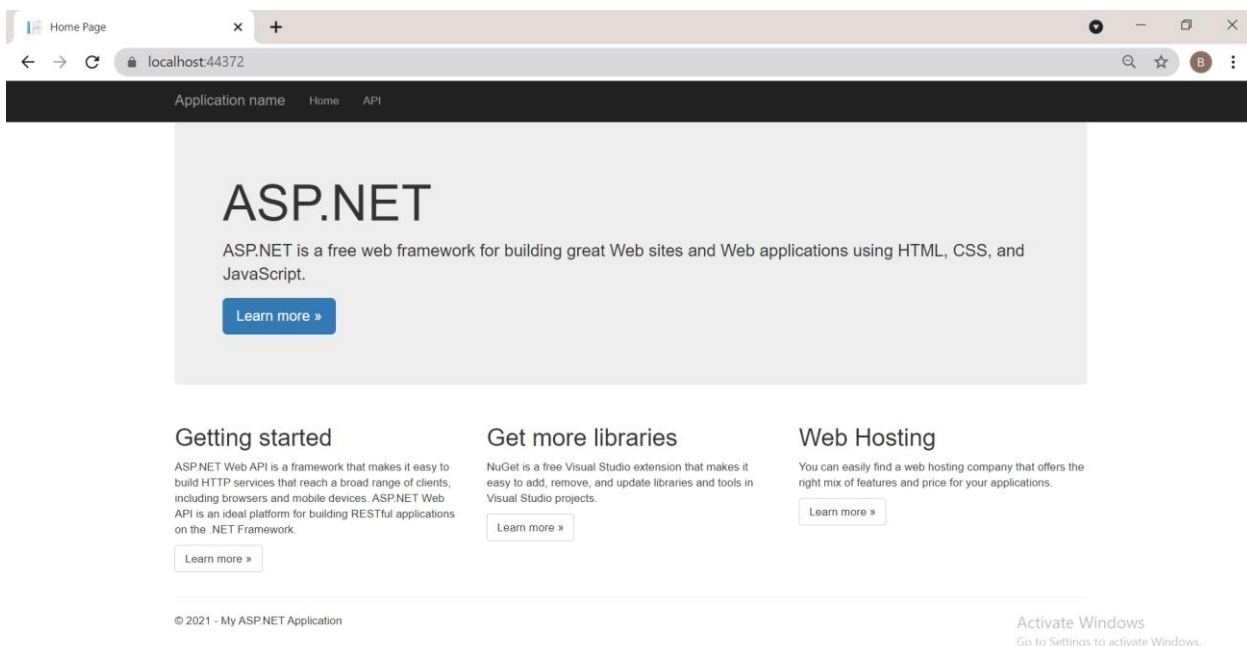


After clicking ok, it shows project in **solution explorer** window that looks like the below.

Two specific files for Web API, WebApiConfig.cs in **App_Start** folder and ValuesController.cs in **Controllers** folder as shown below.



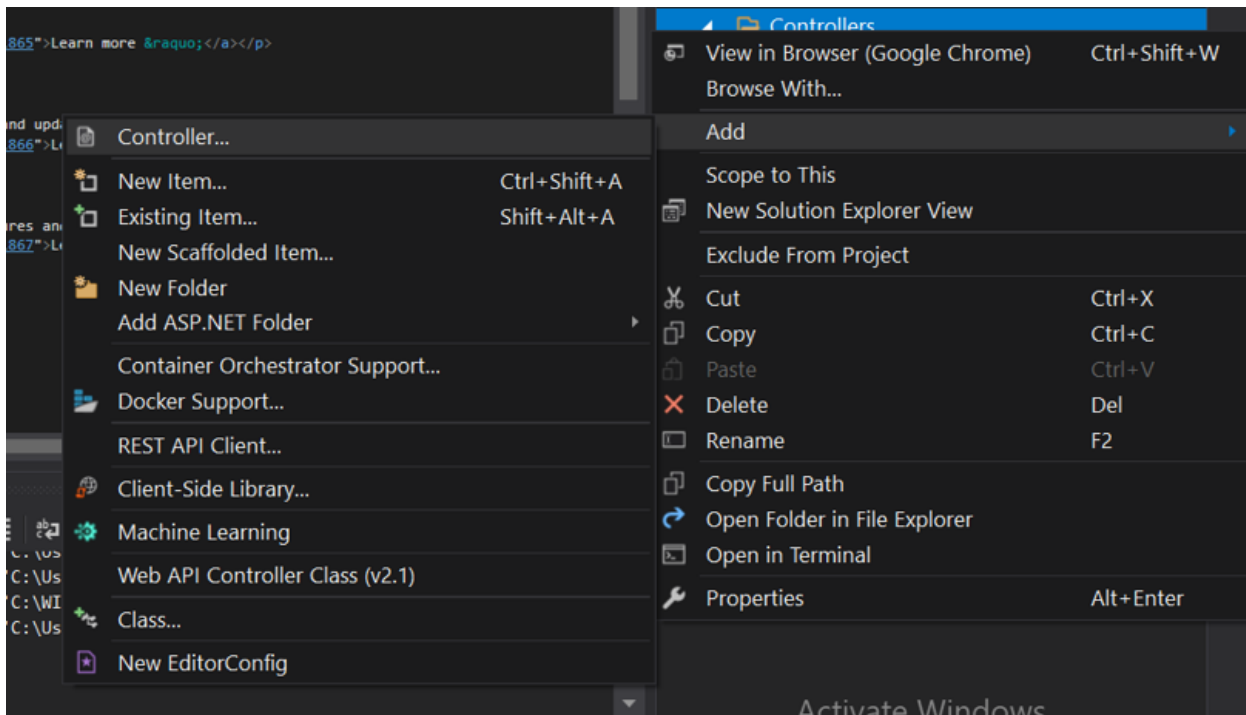
Output:



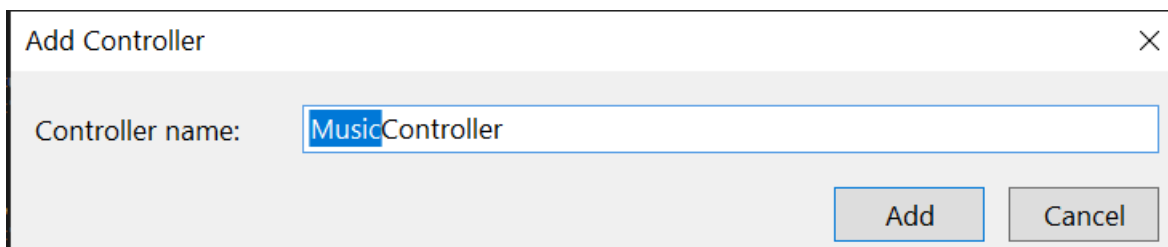
- **Create Controller, Model:**

Create a Controller:

We can create controller for the application by adding a new item into the controller folder. Just right click on the controller folder and click **add -> controller** as given below.

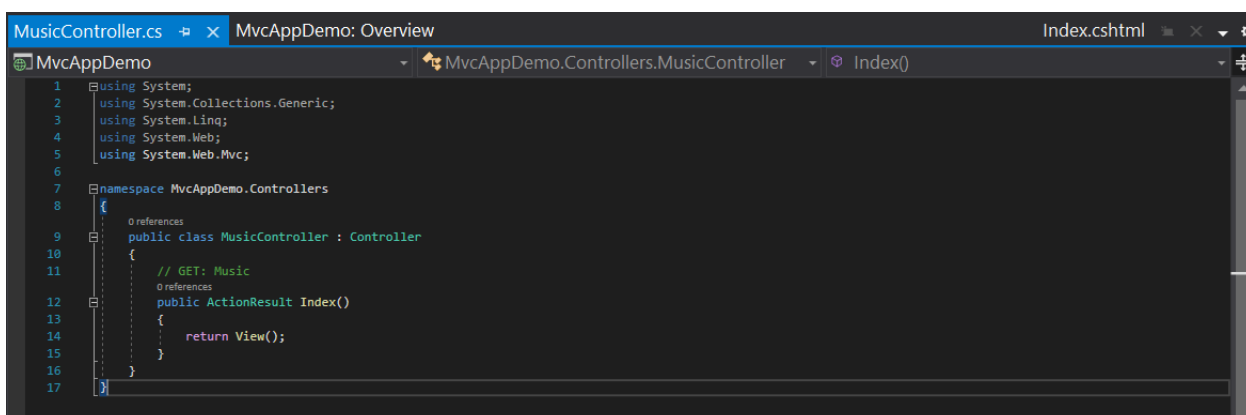


Providing controller name, then click Add.

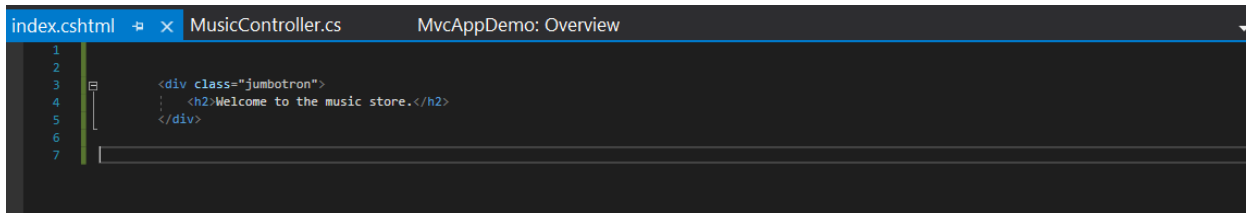


After adding this controller, as per the conventions project will create a folder with the same name as the controller name in the view folder to store the view files belongs to the controller.

This controller contains the default code like below.

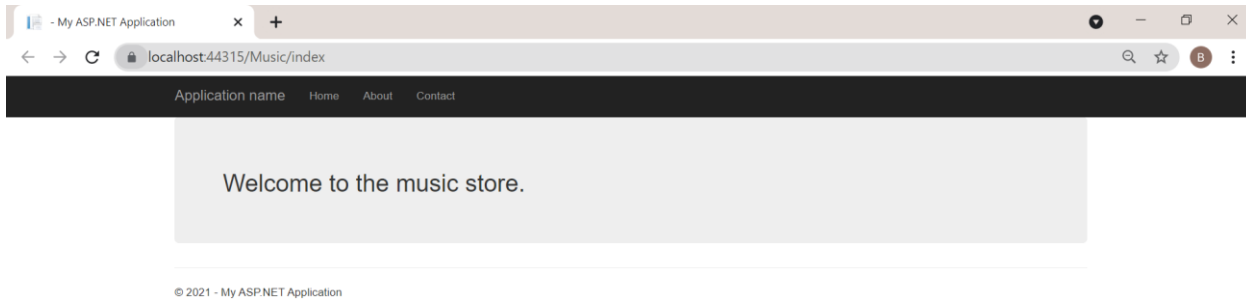


To access this controller to the browser, we are adding an index file to the Music folder inside the view folder. This index file contains the following code.



```
1
2
3 <div class="jumbotron">
4   <h2>Welcome to the music store.</h2>
5 </div>
6
7
```

Output:

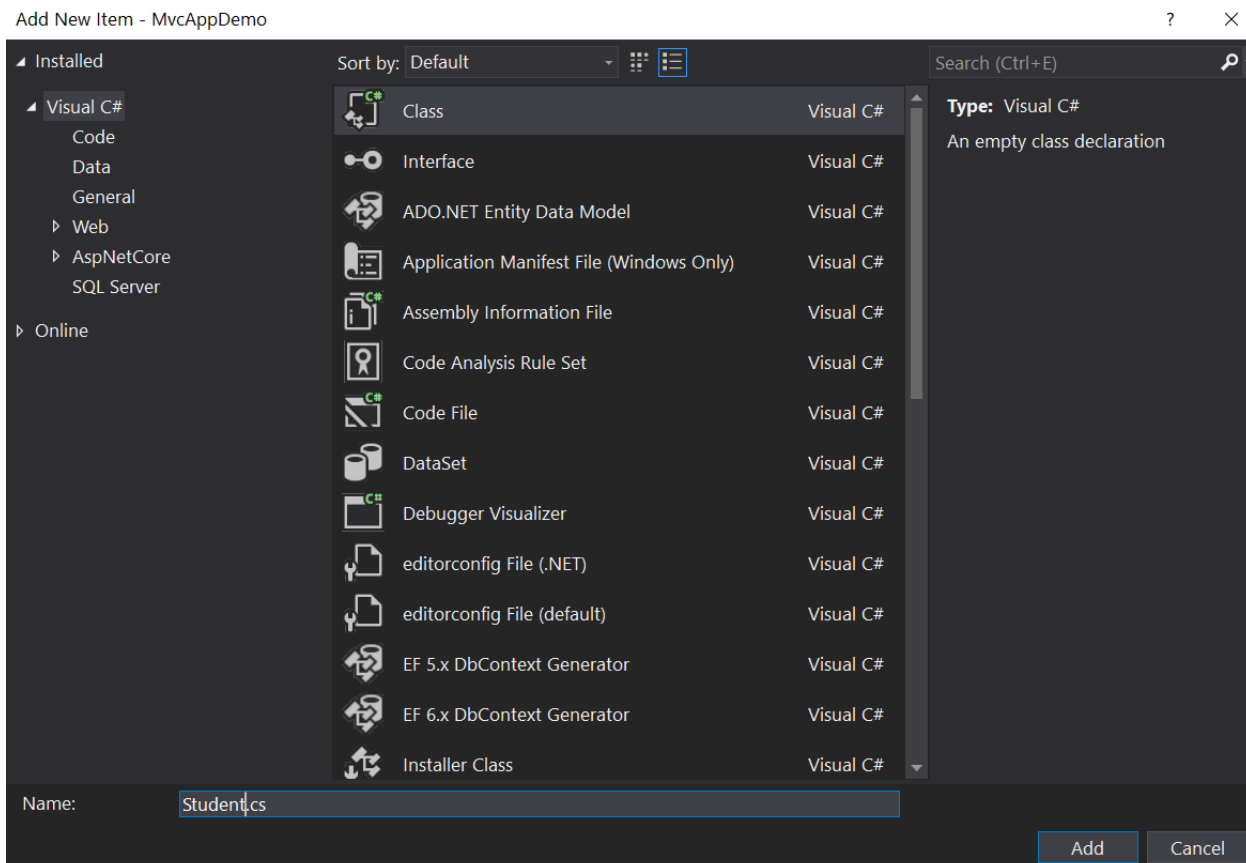


Create a Model:

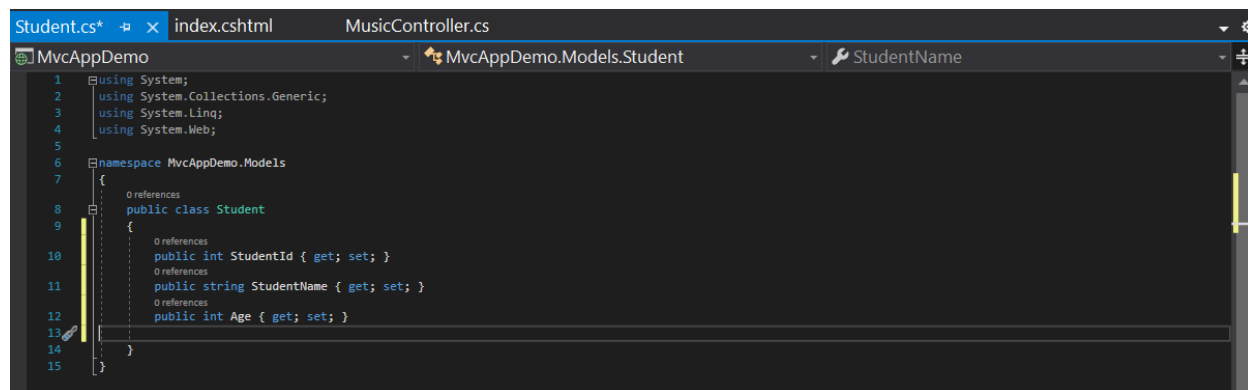
Let's create the model class that should have the required properties for the Student entity.

In the MVC application in Visual Studio, and right-click on the Model folder, select **Add** -> and click on **Class...** It will open the **Add New Item** dialog box.

In the Add New Item dialog box, enter the class name Student and click **Add**.



This will add a new Student class in model folder. We want this model class to store id, name, and age of the students. So, we will have to add public properties for Id, Name, and Age, as shown below.



The model class can be used in the view to populate the data, as well as sending data to the controller.

- **Parameters (From URI, From Body):**

Parameter binding means how Web API binds HTTP request data to the parameters of an action method.

Action methods in Web API controllers can have one or more parameters of different types. It can be either primitive type or complex type. Web API binds action method parameters with the URL's query string or with the request body depending on the parameter type.

By default, if the parameter type is of .NET primitive types such as int, bool, double, string, GUID, DateTime, decimal, or any other type that can be converted from string type, then it sets the value of a parameter from the query string. And if the parameter type is the complex type, then Web API tries to get the value from the request body by default.

Get Action Method with Primitive Parameter:

Consider the following example of the GET action method that includes a single primitive type parameter.

Example:

```
public class StudentController : ApiController
{
    public Student Get(int id)
    {
    }
}
```

As you can see, the above HTTP GET action method includes the id parameter of the int type. So, Web API will try to extract the value of id from the query string of the requested URL, convert it into int and assign it to the id parameter of the GET action method. For example, if an HTTP request is `http://localhost/api/student?id=1` then the value of the id parameter will be 1.

The followings are valid HTTP GET Requests for the above action method.

`http://localhost/api/student?id=1`

`http://localhost/api/student?ID=1`

Multiple Primitive Parameters:

Consider the following example of the GET action method with multiple primitive parameters.

Example:

```
public class StudentController : ApiController
{
    public Student Get(int id, string name)
    {
    }
}
```

As you can see above, an HTTP GET method includes multiple primitive type parameters. So, Web API will try to extract the values from the query string of the requested URL. For example, if an HTTP request is `http://localhost/api/student?id=1&name=steve`, then the value of the id parameter will be 1, and the name parameter will be "steve".

Followings are valid HTTP GET Requests for the above action method.

`http://localhost/api/student?id=1&name=steve`

`http://localhost/api/student?ID=1&NAME=steve`

`http://localhost/api/student?name=steve&id=1`

Post Method with Complex Type Parameter:

consider the following Post() method with the complex type parameter.

Example:

```
public class Student
{
```

```

    public int Id { get; set; }

    public string Name { get; set; }
}

public class StudentController : ApiController
{
    public Student Post(Student stud)
    {
    }
}

```

The above Post() method includes the Student type parameter. So, as a default rule, Web API will try to get the values of the stud parameter from the HTTP request body.

http://localhost/api/student is a valid HTTP POST request for the above action method.

[FromUri] and [FromBody]:

By default, ASP.NET Web API gets the value of a primitive parameter from the query string and a complex type parameter from the request body.

Use [FromUri] attribute to force Web API to get the value of complex type from the query string and [FromBody] attribute to get the value of primitive type from the request body, opposite to the default rules.

Example for FromUri Attribute:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace WebApiParameters.Controllers
{
    public class StudentController : ApiController
    {
        public IEnumerable<string> Get([FromUri] string id, [FromUri] string name)
        {
            return new string[]{
                $"The Id of the Student is {id}",
                $"The Name of the Student is {name}"
            };
        }
    }
}

```


In the above example, the Get() method includes a complex type parameter with the [FromUri] attribute. So, Web API will try to get the value of the Student type parameter from the query string. For example, if an HTTP GET request `http://localhost:xxxx/api/student?id=1&name=abc` then Web API will create an object of the Student type and set its id and name property values to the value of id and name query string parameter.

Output:



In the same way, apply the [FromBody] attribute to get the value of primitive data type from the request body instead of a query string, as shown below.

Example for FromBody Attribute:

Let us create a Student model which has the below properties.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebApiParameters.Models
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

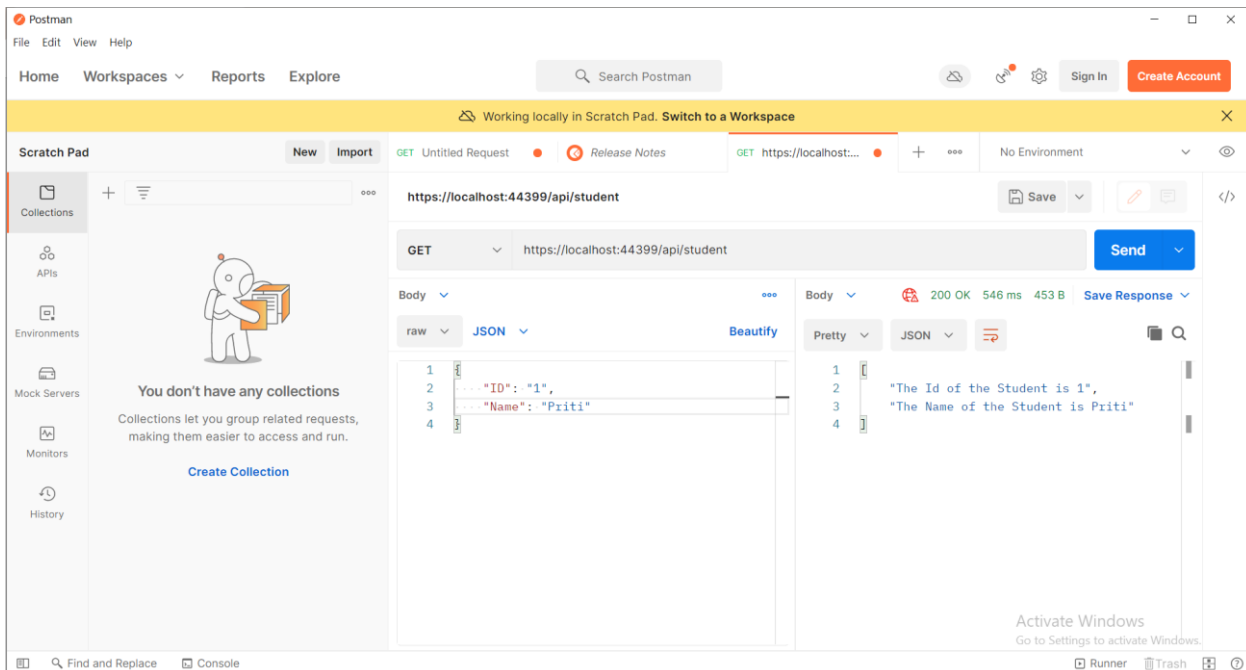
Controller Code :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using WebApiParameters.Models;

namespace WebApiParameters.Controllers
{
    public class StudentController : ApiController
    {
        public IEnumerable<string> Get([FromBody] Student student)
        {
            return new string[]{
                $"The Id of the Student is {student.Id}",
                $"The Name of the Student is {student.Name}"
            };
        }
    }
}
```

`http://localhost:xxxx/api/student` is a valid HTTP POST request for the above action method. For the above example the value for the student is passed in the request body and it is mapped to corresponding property of the Student object. Below is the request and response using Postman.

Output:



- **Serialization:**

Serialization is the process of converting the state of an object into a stream of data. The complement of serialization is deserialization, which converts a stream into an object. Together, these processes allow data to be stored and transferred.

Serialization in .NET is provided by the `System.Runtime.Serialization` namespace. This namespace contains an interface called `IFormatter` which in turn contains the methods `Serialize` and `De-serialize` that can be used to save and load data to and from a stream. In order to implement serialization in .NET, we basically require a stream and a formatter. While the stream acts as a container for the serialized object(s), the formatter is used to serialize these objects onto the stream.

Types of Serialization:

- **Binary serialization:** Binary serialization is a mechanism which writes the data to the output stream such that it can be used to re-construct the object automatically. The term binary in its name implies that the necessary information that is required to create an exact binary copy of the object is saved onto the storage media. A notable difference between Binary serialization and XML serialization is that Binary serialization preserves instance identity while XML serialization does not. In other words, in Binary serialization the entire object state is saved while in XML serialization only some of the object data is saved. Binary serialization can handle graphs with multiple references to the same object; XML serialization will turn each reference into a reference to a unique object.

Steps for serializing an object:

1. Create a stream object.
2. Create a Binary Formatter Object.
3. Call the BinaryFormatter.Serialize() method.

Steps for deserializing an object:

1. Create a stream object to read the serialized output.
2. Create a Binary Formatter Object.
3. Call the BinaryFormatter.Deserialize() method to deserialize the object and cast it to the correct type.

Code:

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace BinaryDemo
{
    [Serializable]
    class Customer
    {
        public int CustomerID { get; set; }
        public string CustomerName { get; set; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // step-1
            Customer objcustomer = new Customer();
            objcustomer.CustomerID = 1;
            objcustomer.CustomerName = "abc";

            //step-2
            IFormatter formatter = new BinaryFormatter();
            Stream stream = new FileStream(@"D:\Company_RKIT\Module-5\test.txt", FileMode.Create, FileAccess.Write);

            //step-3 serialize
            formatter.Serialize(stream, objcustomer);
            stream.Close();

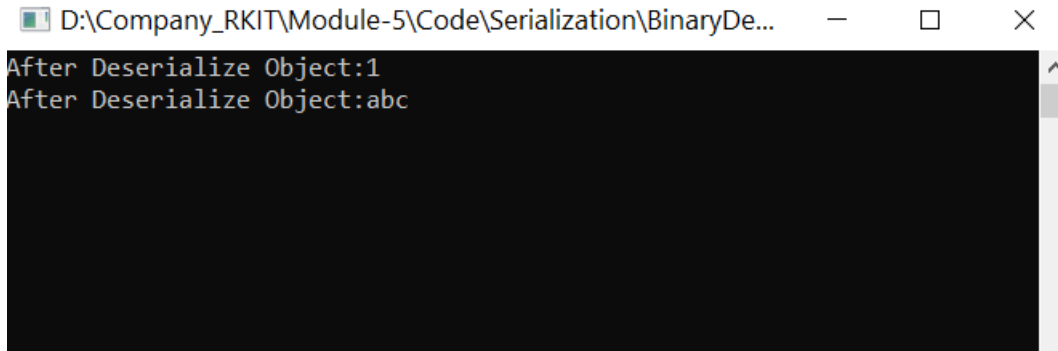
            //step-4 deserialize
            stream = new FileStream(@"D:\Company_RKIT\Module-5\test.txt", FileMode.Open, FileAccess.Read);
            Customer objcustomernew = (Customer)formatter.Deserialize(stream);

            Console.WriteLine("After Deserialize Object:" + objcustomernew.CustomerID);
            Console.WriteLine("After Deserialize Object:" + objcustomernew.CustomerName);

            Console.ReadLine();
        }
    }
}
```

```
}
```

Output:



```
D:\Company_RKIT\Module-5\Code\Serialization\BinaryDe...
After Deserialize Object:1
After Deserialize Object:abc
```

- **XML serialization:** It serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data. Because XML is an open standard, it is an attractive choice for sharing data across the Web.

Some common attributes that are available during Serialization are:

XmlAttribute: This member will be serialized as an XML attribute.

XmlElement: The field will be serialized as an XML element.

XmlIgnore: Field will be ignored during Serialization.

XmlRoot: Represent XML document's root Element.

System.Xml.Serialization namespace.

Steps for serializing an object:

1. Create a stream.
2. Create an XmlSerializer object.
3. Call XmlSerializer.Serialize() method to serialize the object.

Steps for deserializing an object:

1. Create a stream.
2. Create an XmlSerializer object.
3. Call XmlSerializer.Deserialize() method to deserialize the object.

Code:

```
using System;
using System.IO;
using System.Xml.Serialization;
```

```
namespace XmlDemo
{
    public class Employee
    {
```

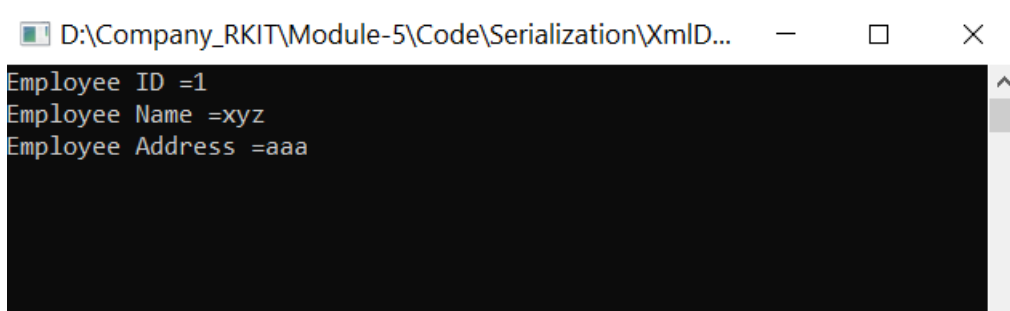
```

public int EmpID { get; set; }
public string EmpName { get; set; }
[XmlElement("EmpAddress")]
public string Address { get; set; }
[XmlIgnore]
public double Salary { get; set; }
public Employee()
{
    EmpID = 0;
    EmpName = "N/A";
    Address = "N/A";
}
}
class Program
{
    public static void SerializeData(Employee emp)
    {
        // create file to save the data
        FileStream fs = new FileStream(@"D:\Company_RKIT\Module-5\test1.txt", FileMode.Create);
        //XmlSerializer object will perform the serialization
        XmlSerializer xmlobj = new XmlSerializer(typeof(Employee));
        //Serialize method serialize the data to the file
        xmlobj.Serialize(fs, emp);
        // close the file
        fs.Close();
    }
    public static void DeSerializeData()
    {
        // open file to read the data
        FileStream fs = new FileStream(@"D:\Company_RKIT\Module-5\test1.txt", FileMode.Open);
        //XmlSerializer object to perform the deserialization
        XmlSerializer xs = new XmlSerializer(typeof(Employee));
        //Use the XmlSerializer object to deserialize the data from the file
        Employee emp = (Employee)xs.Deserialize(fs);
        //close the file
        fs.Close();
        //Display the deserialized data
        Console.WriteLine("Employee ID =" + emp.EmpID);
        Console.WriteLine("Employee Name =" + emp.EmpName);
        Console.WriteLine("Employee Address =" + emp.Address);
    }
    static void Main(string[] args)
    {
        Employee objemployee = new Employee();
        objemployee.EmpID = 1;
        objemployee.EmpName = "xyz";
        objemployee.Address = "aaa";
        SerializeData(objemployee);
        DeSerializeData();

        Console.ReadLine();
    }
}

```

Output:



```
D:\Company_RKIT\Module-5\Code\Serialization\XmlD...
Employee ID =1
Employee Name =xyz
Employee Address =aaa
```

- **JSON serialization:** It serializes only public properties and does not preserve type fidelity. JSON is an open standard that is an attractive choice for sharing data across the web.

Steps for serialization and deserialization:

Step 1: Add the namespace. Newtonsoft.Json/using Nancy.Json;

Step 2: Serializing an object into JSON string.

Step 3: Deserializing JSON.

Code:

```
using Nancy.Json;
using System;

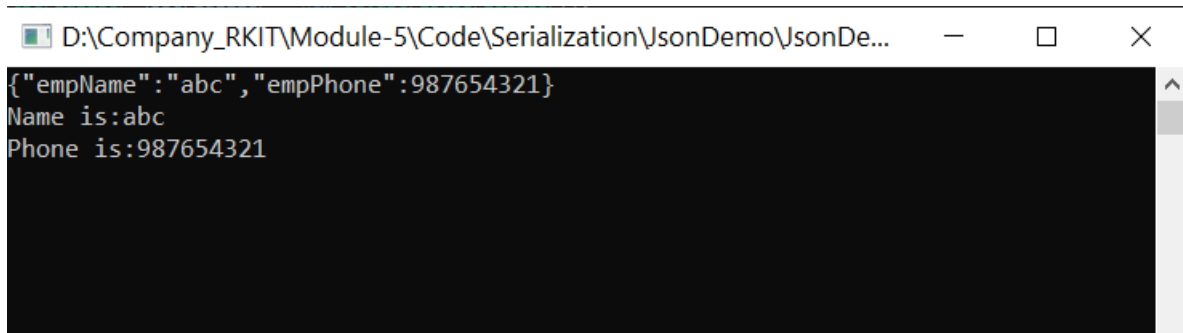
namespace JsonDemo
{
    class Program
    {
        class Employee
        {
            public string EmpName { get; set; }
            public int EmpPhone { get; set; }
        }

        static void Main(string[] args)
        {
            // Creating an Employee object
            Employee objemployee = new Employee()
            {
                EmpName = "abc",
                EmpPhone = 987654321
            };

            //Nancy.Serialization.JsonNet - Use Serialize method to convert the object to JSON
            JavaScriptSerializer jsSerializer = new JavaScriptSerializer();
            string jsonString = jsSerializer.Serialize(objemployee);
            Console.WriteLine(jsonString);
            //Use Deserialize() method to convert JSON to Object
            JavaScriptSerializer jsSerializer2 = new JavaScriptSerializer();
            Employee empobj = jsSerializer2.Deserialize<Employee>(jsonString);
            string name = empobj.EmpName;
            int phone = empobj.EmpPhone;
            Console.WriteLine("Name is:" + name);
            Console.WriteLine("Phone is:" + phone);
            Console.ReadLine();
        }
    }
}
```

```
}  
}  
}
```

Output:



```
D:\Company_RKIT\Module-5\Code\Serialization\JsonDemo\JsonDe...  
{ "empName": "abc", "empPhone": 987654321 }  
Name is:abc  
Phone is:987654321
```

- **Routing:**

Web API Routing:

In ASP.NET Web API, a controller is a class that handles HTTP requests. The public methods of the controller are called action methods or simply actions. When the Web API framework receives a request, it routes the request to an action.

To determine which action to invoke, the framework uses a routing table. The Visual Studio project template for Web API creates a default route:

```
routes.MapHttpRoute(  
    name: "API Default",  
    routeTemplate: "api/{controller}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
);
```

This route is defined in the WebApiConfig.cs file, which is placed in the App_Start directory.

When the Web API framework receives an HTTP request, it tries to match the URI against one of the route templates in the routing table. If no route matches, the client receives a 404 error. For example, the following URIs matches the default route:

- /api/contacts
- /api/contacts/1

However, the following URI does not match, because it lacks the "api" segment:

- /contacts/1

The reason for using "api" in the route is to avoid collisions with ASP.NET MVC routing. That way, you can have "/contacts" go to an MVC controller, and "/api/contacts" go to a Web API controller.

Once a matching route is found, Web API selects the controller and the action.

It routes an incoming HTTP request to a particular action method on a Web API controller. Web API supports two types of routing:

1. Convention-based Routing
2. Attribute Routing

1. Convention-based Routing

In the convention-based routing, Web API uses route templates to determine which controller and action method to execute. At least one route template must be added into route table in order to handle various HTTP requests.

When we created Web API project using WebAPI template in the Create Web API Project section, it also added WebApiConfig class in the App_Start folder with default route as shown below.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Enable attribute routing
        config.MapHttpAttributeRoutes();

        // Add default route using convention-based routing
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

In the above WebApiConfig.Register() method, config.MapHttpAttributeRoutes() enables attribute routing. The config.Routes is a route table. The "DefaultApi" route is added in the route table using MapHttpRoute() extension method.

You can configure multiple routes in the Web API using HttpConfiguration object. The following example demonstrates configuring multiple routes.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();
    }
}
```



```

    // school route
    config.Routes.MapHttpRoute(
        name: "School",
        routeTemplate: "api/myschool/{id}",
        defaults: new { controller="school", id = RouteParameter.Optional }
        constraints: new { id = "/d+" }
    );

    // default route
    config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
}
}

```

In the above example, School route is configured before DefaultApi route. So any incoming request will be matched with the School route first and if incoming request url does not match with it then only it will be matched with DefaultApi route. For example, request url is `http://localhost:1234/api/myschool` is matched with School route template, so it will be handled by SchoolController.

2. Attribute Routing

Attribute routing is supported in Web API 2. As the name implies, attribute routing uses `[Route()]` attribute to define routes. The Route attribute can be applied on any controller or action method.

In order to use attribute routing with Web API, it must be enabled in WebApiConfig by calling `config.MapHttpAttributeRoutes()` method.

Consider the following example of attribute routing.

```

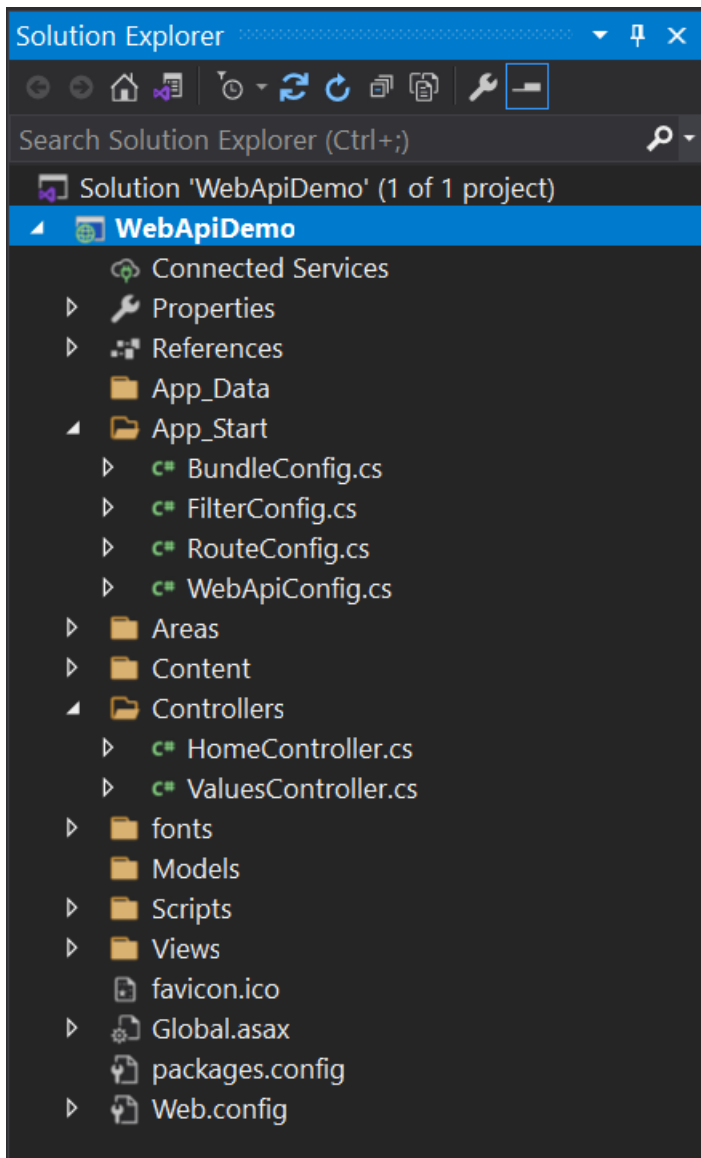
namespace WebApiDemo.Controllers
{
    public class StudentController : ApiController
    {
        [Route("api/student/names")]
        public IEnumerable<string> Get()
        {
            return new string[] { "student1", "student2" };
        }
    }
}

```

In the above example, the Route attribute defines new route "api/student/names" which will be handled by the Get() action method of StudentController. Thus, an HTTP GET request `http://localhost:1234/api/student/names` will return list of student names.

- **Config:**

Web API supports code based configuration. We created a simple Web API project. Web API project includes default WebApiConfig class in the App_Start folder and also includes Global.asax as shown below.



//Global.asax.cs

```
public class WebAPIApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        GlobalConfiguration.Configure(WebApiConfig.Register);

        //other configuration
    }
}
```

//WebApiConfig.cs

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
    }
```

```
config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);

// configure additional webapi settings here..
}
```

Web API configuration process starts when the application starts. It calls `GlobalConfiguration.Configure(WebApiConfig.Register)` in the `Application_Start` method. The `Configure()` method requires the callback method where Web API has been configured in code. By default this is the static `WebApiConfig.Register()` method.

As you can see above, `WebApiConfig.Register()` method includes a parameter of `HttpConfiguration` type which is then used to configure the Web API.