TRƯỜNG ĐẠI HỌC FPT

Working with ASP.NET Core Web API

Objectives

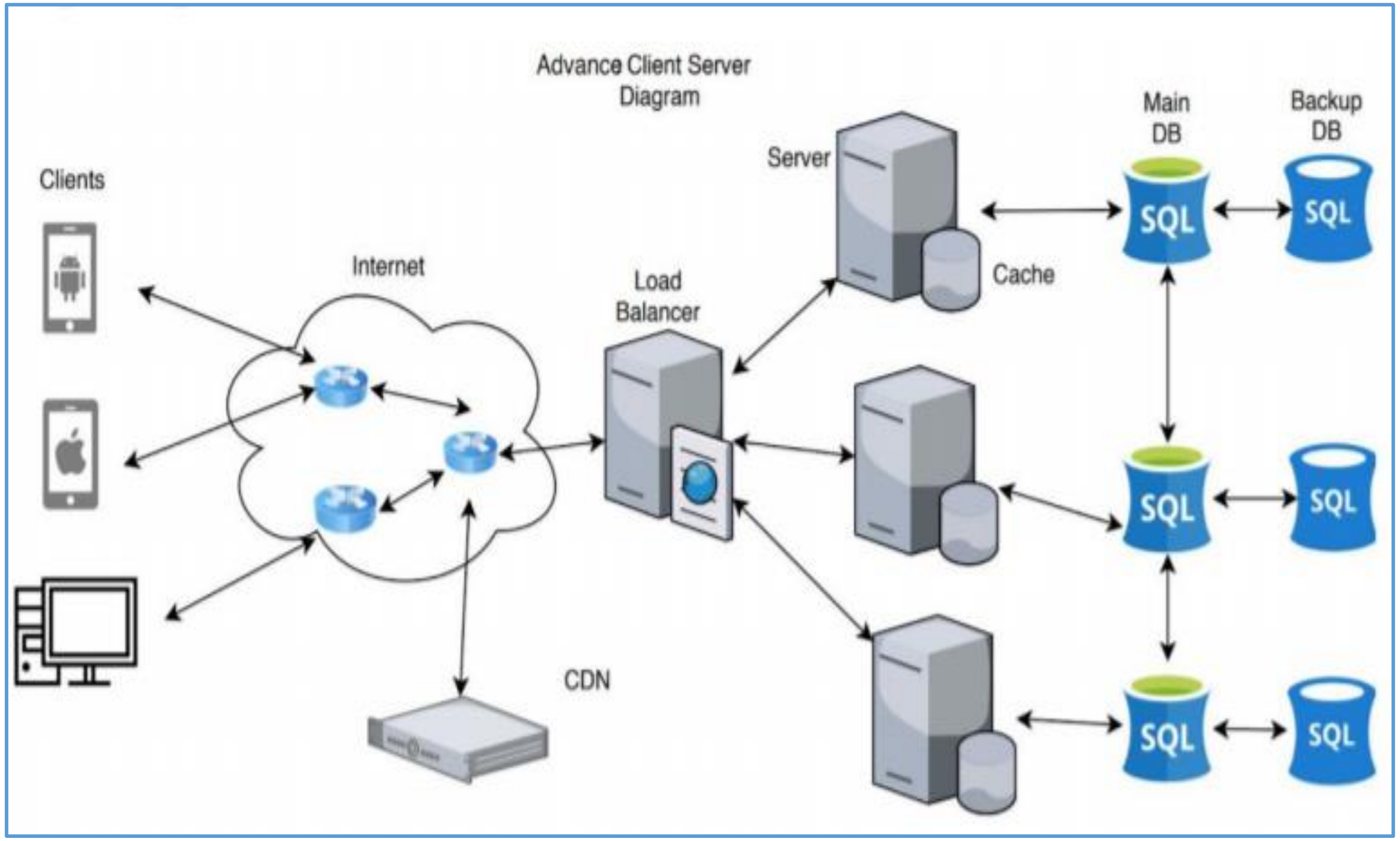
- Overview Client-server Architecture
- Overview ASP.NET Core RESTful Services
- Explain about Web Service
- Explain about ASP.NET Web API Characteristics
- Demo create ASP.NET Core Web API application
- Demo create ASP.NET MVC Core Application consume Web API
- Demo create WinForms Application consume Web API

Client-server Architecture

- Client-server architecture is a computing model in which the server hosts, delivers and manages most ofthe resources and services to be consumed by the client
- This type of architecture has one or more client computers connected to a central server over a network or internet connection
- Client-server architecture is also known as a networking computing model or client-server network because all the requests and services are delivered over a network

Client-server Architecture

1. Client requests data from server
2. Load balancer routes the request to the appropriate server
3. Server processes the request client
4. Server queries appropriate database for some data
5. Database returns the queried data back to the server
6. The server processes the data and sends the data back to the client
7. This process repeats

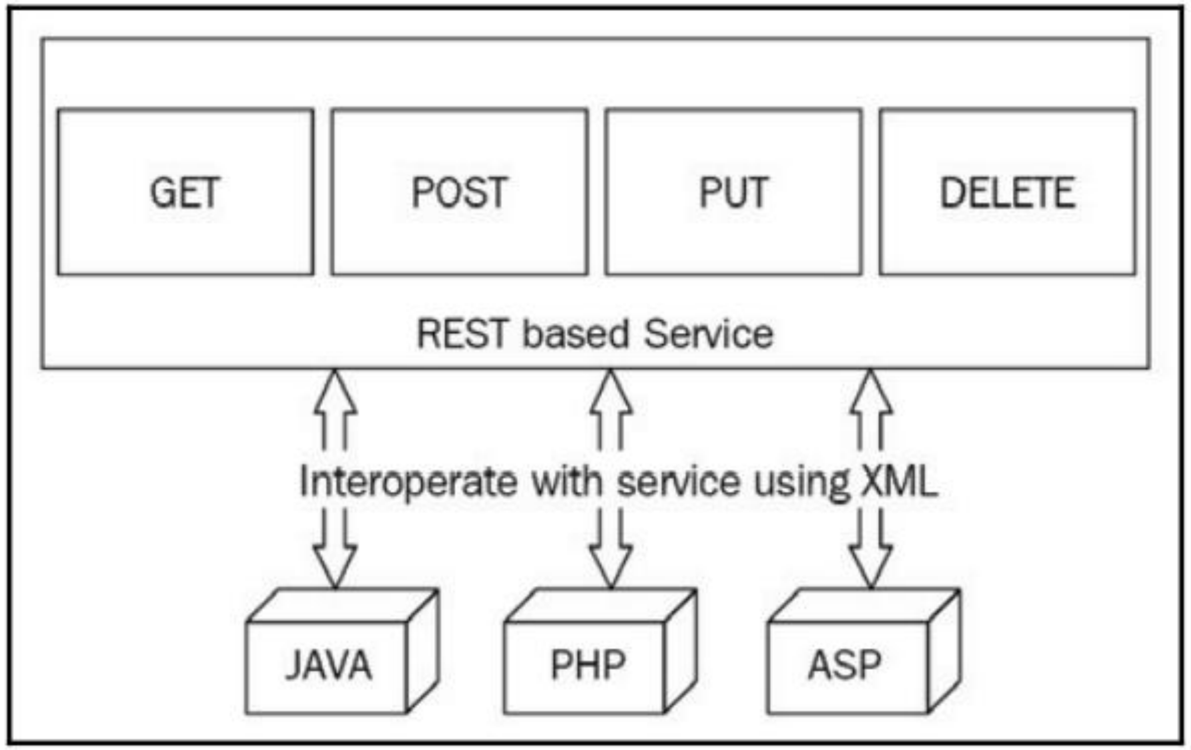


Web Service Acronyms

- Web services are services that use the HTTP communication standard, so they are sometimes called HTTP or RESTful services
- Web services can also mean Simple Object Access Protocol (SOAP) services that implement some of the WS-* standards
- Microsoft .NET Framework 3.0 and later includes a remote procedure call (RPC) technology named Windows Communication Foundation (WCF), which makes it easy for developers to create services including SOAP services that implement WS-* standards
- gRPC is a modern cross-platform open source RPC framework created by Google (the "g" in gRPC)

RESTful Services

- **REST** stands for ***representational state transfer***. It is an architectural style that defines a set of guidelines for building web services
- The following is a simple diagram of a REST-based service:



ASP.NET Core and RESTful services

- ASP.NET Web API from the beginning was designed to be a service-based framework for building RESTful (REpresentational State Transfer) services
- ASP.NET Web API is based on the MVC framework minus the “V” (view), with optimizations for creating headless services. These services can be called by any technology
- Calls to a Web API service are based on the core HTTP Verbs (Get, Put, Post, Delete) through a URI (Uniform Resource Identifier)
- The ASP.NET MVC framework started gaining traction almost immediately, and Microsoft released ASP.NET Web API with ASP.NET MVC 4 and Visual Studio 2012
- ASP.NET Web API 2 was released with Visual Studio 2013 and then updated the framework to version 2.2 with Visual Studio 2013 Update 1

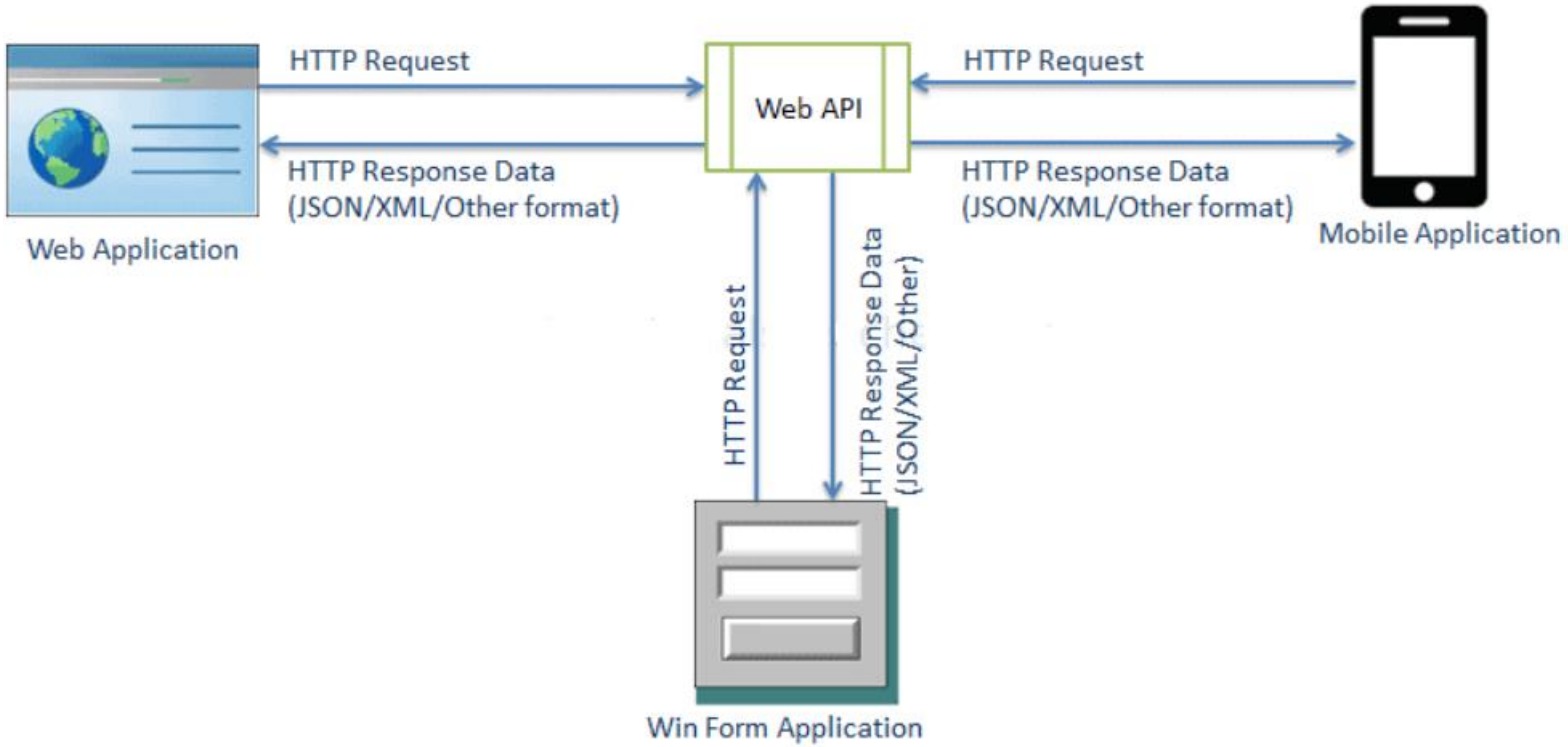
ASP.NET Core and RESTful services

- ASP.NET Web API is an ideal platform for building RESTful services
- ASP.NET Web API is built on top of ASP.NET and supports ASP.NET request/response pipeline
- ASP.NET Web API maps HTTP verbs to method names
- ASP.NET Web API supports different formats of response data. Built-in support for JSON, XML, BSON format
- ASP.NET Web API can be hosted in IIS, Kestrel, Self-hosted or other web server that supports
- ASP.NET Core Web API includes new HttpClient to communicate with Web API server. HttpClient can be used in ASP.MVC server side, Windows Form application, Console application or other apps

ASP.NET Core and RESTful services

- ASP.NET Web API has been built to map the web/HTTP programming model to the .NET Framework programming model. It uses familiar constructs, such as Controller, Action, Filter, and so on, which are used in ASP.NET MVC
- ASP.NET Web API is designed on top of the ASP.NET MVC runtime, along with some components that simplify HTTP programming
- ASP.NET Core supports creating RESTful services. To handle requests, a web API uses controllers
- A Web API consists of one or more controller classes that derive from **ControllerBase**

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
```


HTTP Request
Web API
HTTP Request
HTTP Response Data
(JSON/XML/Other format)
HTTP Response Data
(JSON/XML/Other format)
Web Application
Mobile Application
HTTP Request
HTTP Response Data
(JSON/XML/Other)
Win Form Application

Web API

ControllerBase Class

- The ControllerBase class provides the core functionality for both ASP.NET Core web applications and services, in addition to helper methods for returning HTTP status codes and properties
  - Some of the Properties provided by the ControllerBase Class:

Properties	Description
HttpContext	Returns the HttpContext for the currently executing action
Request	Returns the HttpRequest for the currently executing action
Response	Returns the HttpResponse for the currently executing action
ModelState	Returns the state of the model in regard to model binding and validation
Url	Returns an instance of the IUrlHelper, providing access to building URLs for ASP.NET MVC Core applications and services
RouteData	Returns the RouteData for the currently executing action

ControllerBase Class

- Some of the Helper Methods provided by the ControllerBase Class:

Method	Notes
BadRequest	Returns 400 status code
NotFound	Returns 404 status code
PhysicalFile	Returns a file
TryUpdateModelAsync	Invokes model binding
TryValidateModel	Invokes model validation
NoContent	Creates a NoContentResult object that produces an empty Status204NoContent response
Ok	Creates a OkResult object that produces an empty Status200OK response
Accepted()	Creates a AcceptedResult object that produces an Status202Accepted response
Content(String)	Creates a ContentResult object by specifying a content string

Request and Response

- A REST request generally consists of the following:
  - **HTTP verb**: This denotes what kind of operation the requests want to perform on the server
  - **Header**: This element of the REST request allows the client to pass more information about the request
  - **URL**: The actual path to the resource that the REST request wants to operate on
  - **Body**: The body can contain extra data related to a resource to identify or update it. This is optional though

HTTP verbs

- The following are basic HTTP verbs used while requesting a REST system for resource interaction:
  - **GET**: Used to retrieve a specific resource by its identity or a collection of resources
  - **POST**: Used to create/insert a new resource
  - **PUT**: Used to update a specific resource by its identity
  - **DELETE**: Used to remove a specific resource by its identity

Status Code

- When a server returns responses, it includes status codes. These status codes inform the client how the request performed on the server

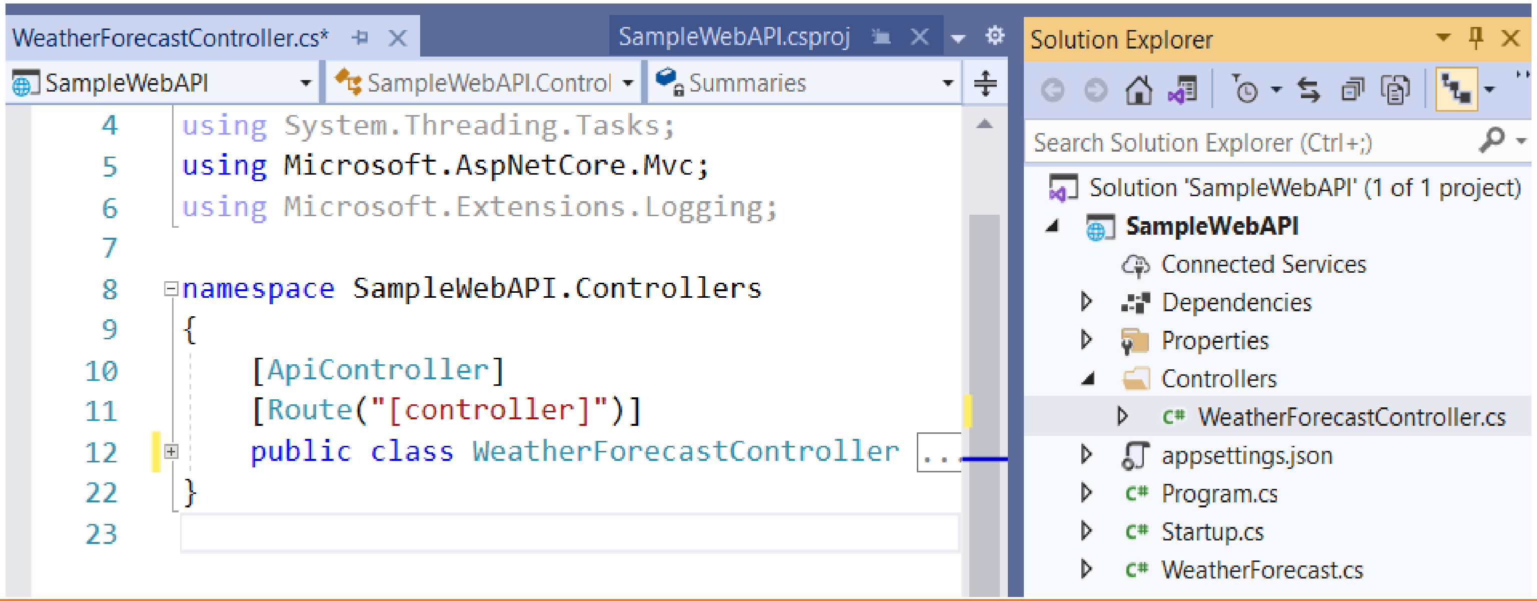
Status	Code Explanation
200	OK Standard response for successful HTTP requests
201	CREATED Standard response for an HTTP request when an item is successfully
204	NO CONTENT Standard response for successful HTTP requests, if nothing is returned in the response body
400 BAD REQUEST	Request cannot be processed because of bad request syntax, excessive size, or another client error
403 FORBIDDEN	Client does not have permission to access the requested resource
404 NOT FOUND	Resource could not be found at this time. It might have been deleted, or does not exist yet
500 INTERNAL SERVER ERROR	This response comes whenever there is a failure or exception happens while processing the server side codes

A web API sample

- Create Web API by **dotnet** CLI

```
D:\Demo\FU>dotnet new api --no-https -o SampleWebAPI
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on SampleWebAPI\SampleWebAPI.csproj...
  Determining projects to restore...
  Restored D:\Demo\FU\SampleWebAPI\SampleWebAPI.csproj (in 250 ms).
Restore succeeded.
```

- Open **SampleWebAPI** project by Visual Studio



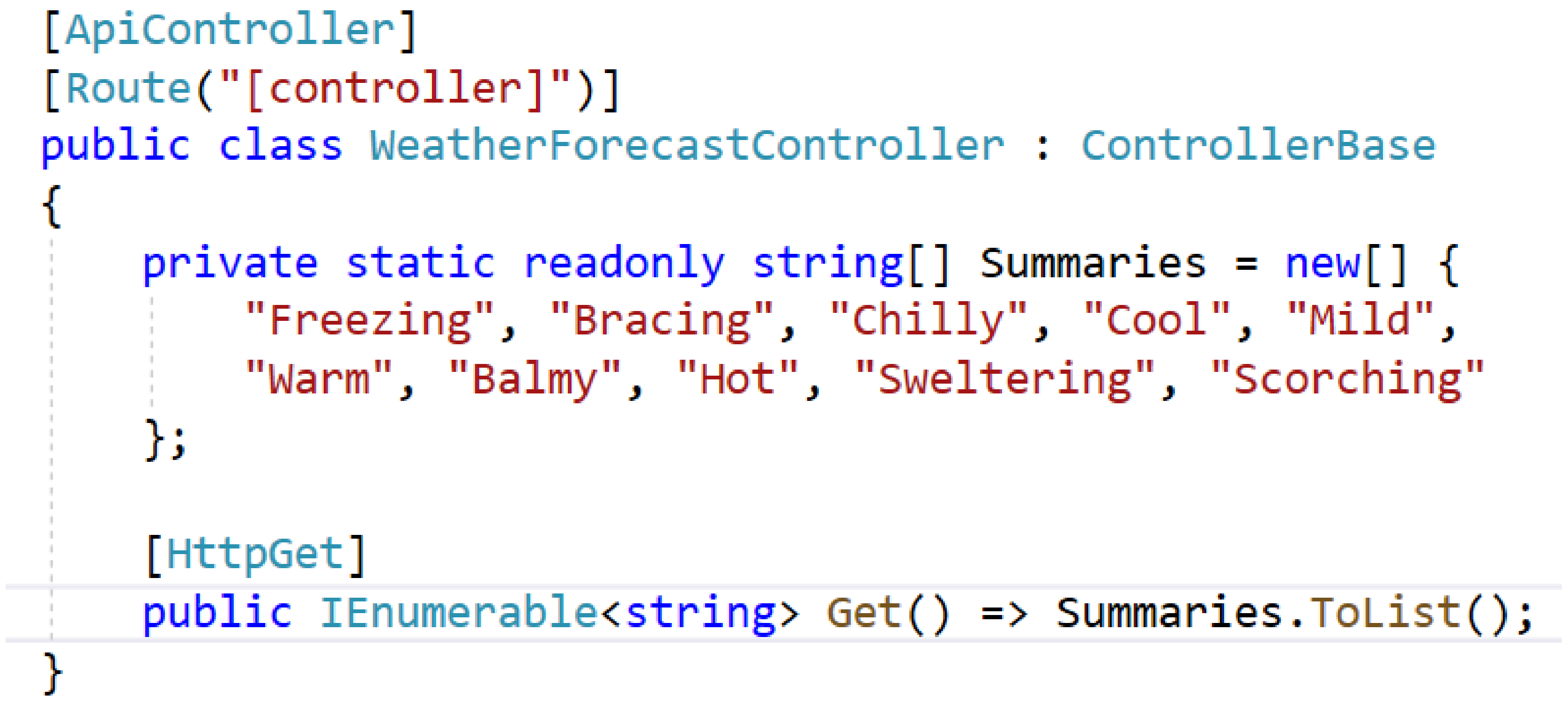
```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace SampleWebAPI.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class WeatherForecastController ...
}
```

Solution Explorer
- Solution 'SampleWebAPI' (1 of 1 project)
  - SampleWebAPI
    - Connected Services
    - Dependencies
    - Properties
    - Controllers
      - WeatherForecastController.cs
    - appsettings.json
    - Program.cs
    - Startup.cs
    - WeatherForecast.cs

WeatherForecastController.cs*
SampleWebAPI.csproj
SampleWebAPI
SampleWebAPI.Control
Summaries
Search Solution Explorer (Ctrl+;)

A web API sample

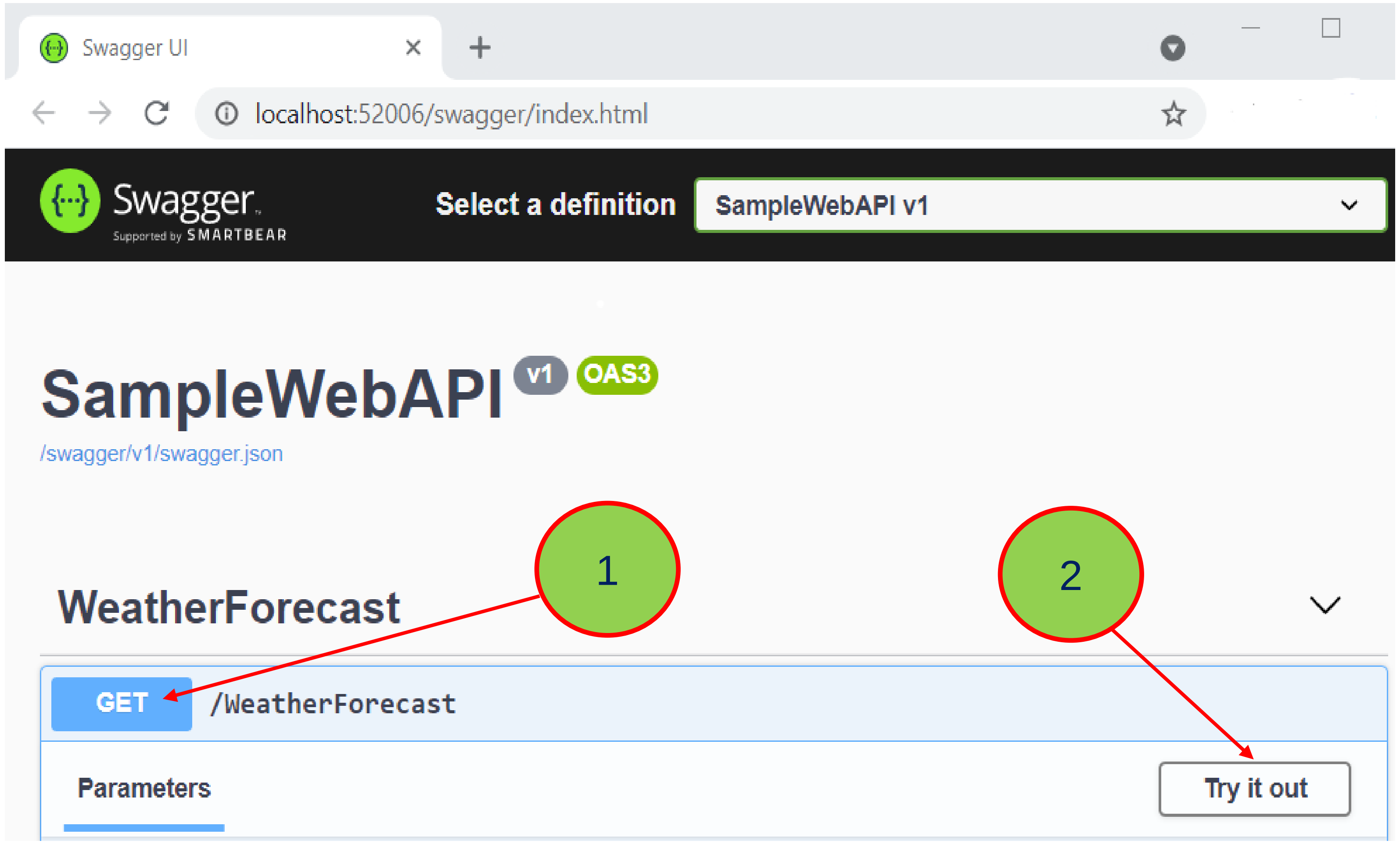
- Update code for **WeatherForecastController.cs** as follows:

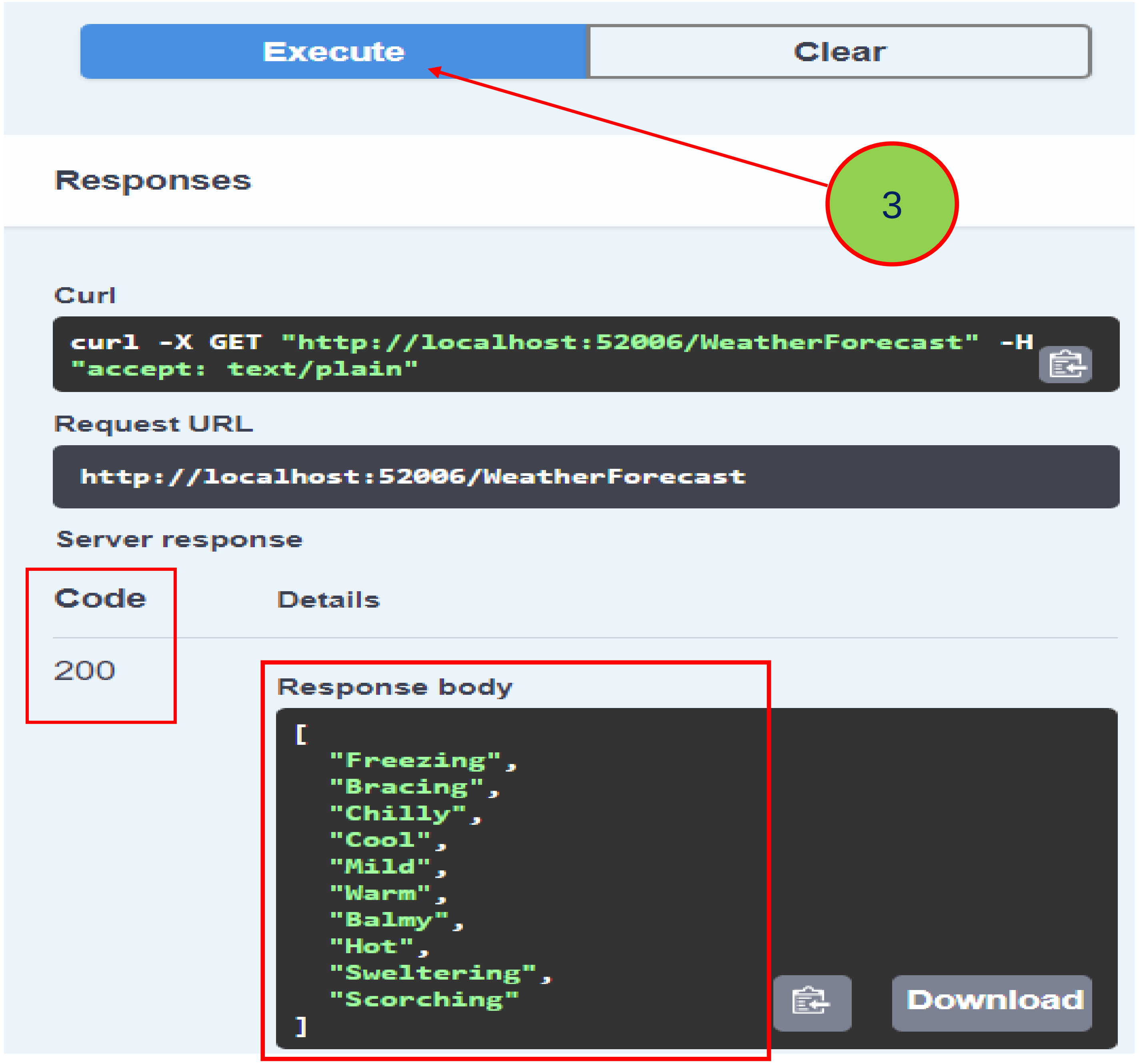


```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private static readonly string[] Summaries = new[] {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild",
        "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
    };

    [HttpGet]
    public IEnumerable<string> Get() => Summaries.ToList();
}
```

A web API sample

- Run project to test **Get** method by **Swagger**





Curl

```
curl -X GET "http://localhost:52006/WeatherForecast" -H
"accept: text/plain"
```

Request URL

```
http://localhost:52006/WeatherForecast
```

Server response

Code	Details
200	Response body

```
[
  "Freezing",
  "Bracing",
  "Chilly",
  "Cool",
  "Mild",
  "Warm",
  "Balmy",
  "Hot",
  "Sweltering",
  "Scorching"
]
```

ASP.NET Core attributes

- The Microsoft.AspNetCore.Mvc namespace provides attributes that can be used to configure the behavior of web API controllers and action methods
- The following example uses attributes to specify the supported HTTP action verb and any known HTTP status codes that could be returned:

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public ActionResult<Product> Create(Product product)
{
   //………
   return CreatedAtAction(nameof(GetById), new { id = product.Id }, product);
}
```

ASP.NET Core attributes

- Some more examples of attributes that are available:

Method	Notes
HttpPostAttribute	Identifies an action that supports the HTTP POST method
HttpPutAttribute	Identifies an action that supports the HTTP PUT method
HttpDeleteAttribute	Identifies an action that supports the HTTP DELETE method
HttpGetAttribute	Identifies an action that supports the HTTP GET method
RouteAttribute	Specifies an attribute route on a controller
HttpHeadAttribute	Identifies an action that supports the HTTP HEAD method
HttpOptionsAttribute	Identifies an action that supports the HTTP OPTIONS method
HttpPatchAttribute	Identifies an action that supports the HTTP PATCH method

More attributes: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc?view=aspnetcore-5.0

ASP.NET Core attributes

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase{
    // GET: api/Products
    [HttpGet] // Get all products
    public ActionResult<IEnumerable<Product>> GetProducts(){
        //Implement get product list
    }
    // POST: api/Products
    [HttpPost]
    public IActionResult PostProduct(Product product){
        //Implement add new a product
    }//end PostProduct
    [HttpPut]
    public IActionResult PutProduct(int id,Product product){
        //Implement update a product
    }//end PostProduct
    // DELETE: api/Products/5
    [HttpDelete("{id}")]
    public IActionResult DeleteProduct(int id){
        //Implement delete a product
    }//end DeleteProduct
}//end class
```

Binding Source Parameter Inference

- A binding source attribute defines the location at which an action parameter's value is found
- The following binding source attributes exist:

Attribute	Binding source
[FromBody]	Inferred for complex types. Only one FromBody parameter can exist or an exception will be thrown. Exceptions exist for IFormCollection and CancellationToken
[FromForm]	Inferred for action parameters of type IFormFile and IFormFileCollection. When parameter is marked with FromForm, the multipart/form-data content type is inferred
[FromHeader]	Request header
[FromQuery]	Inferred for any other action parameters
[FromRoute]	Inferred for any parameter name that matches a route token name
[FromServices]	The request service injected as an action parameter

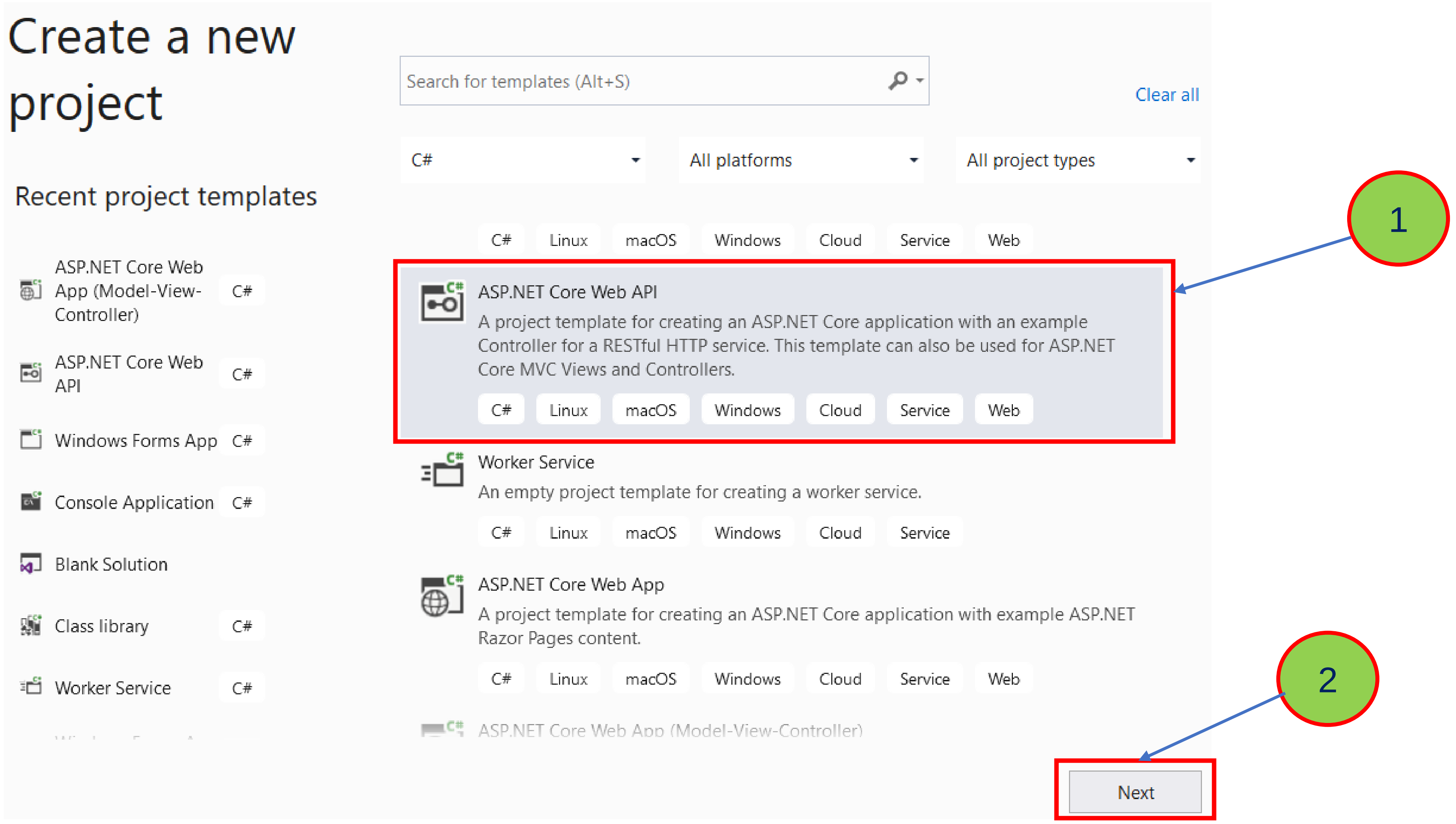
TRƯỜNG ĐẠI HỌC FPT

Binding Source Parameter Inference

```
[HttpGet] public ActionResult<List<Product>> Get( [FromQuery] bool discontinuedOnly = false) {
   List<Product> products = null;
   if (discontinuedOnly) {
       products = _productsInMemoryStore.Where(p => p.IsDiscontinued).ToList();
   }
   else {
       products = _productsInMemoryStore;
   }
   return products;
}
```

Demo 01- Create a Web API Application

1. Open Visual Studio.NET , File | New | Project


Create a new project

Search for templates (Alt+S)

Clear all

C#

All platforms

All project types

Recent project templates

ASP.NET Core Web App (Model-View-Controller)	C#
ASP.NET Core Web API	C#
Windows Forms App	C#
Console Application	C#
Blank Solution	
Class library	C#
Worker Service	C#

C# Linux macOS Windows Cloud Service Web

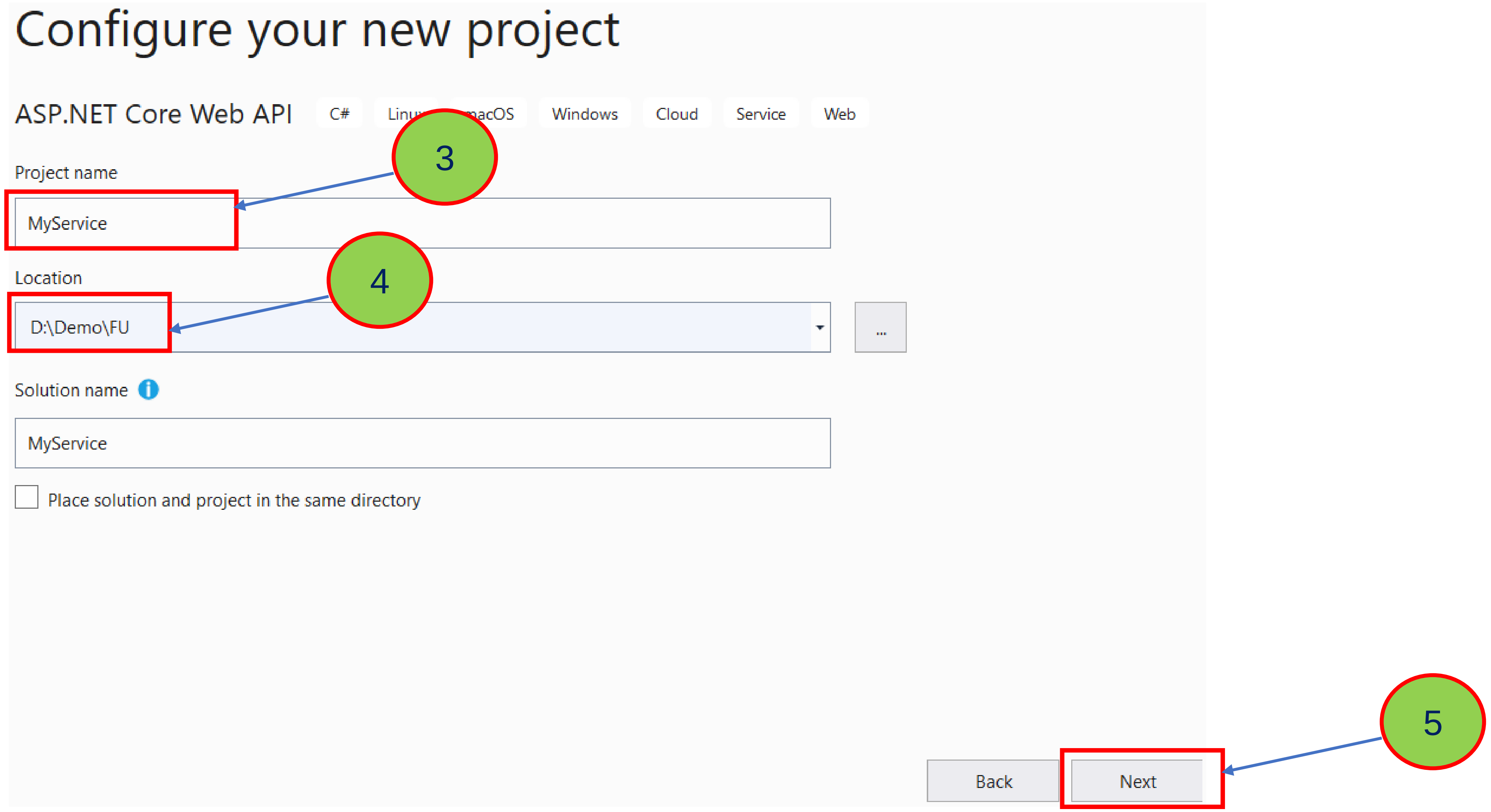
ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web

Worker Service
An empty project template for creating a worker service.
C# Linux macOS Windows Cloud Service

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)

2. Fill out **Project name**: MyWebApp and **Location** then click **Next**


Configure your new project

ASP.NET Core Web API

Project name

MyService

Location

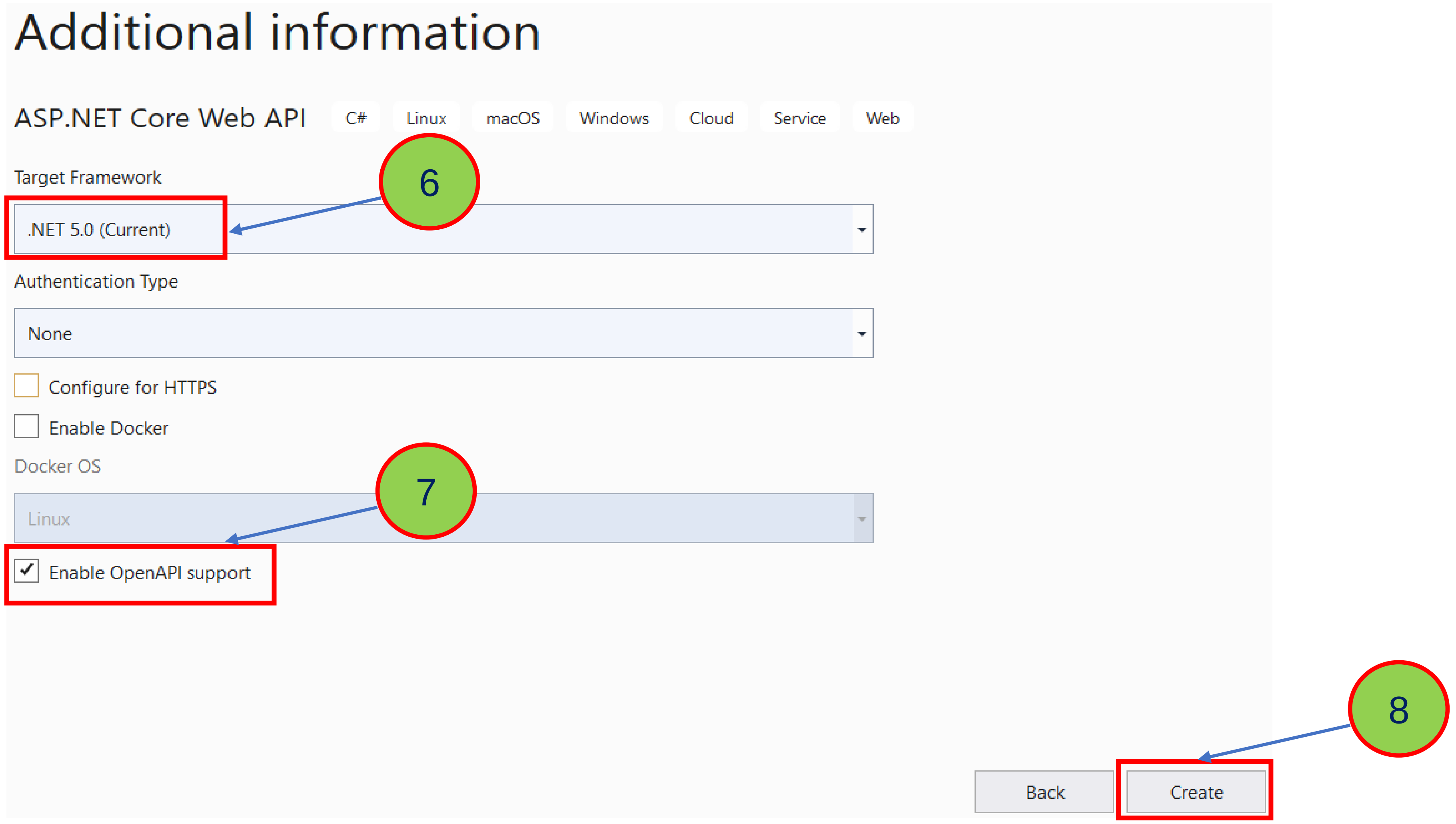
D:\Demo\FU

Solution name

MyService

☐ Place solution and project in the same directory

3. Config as follows then click **Create**


Additional information

ASP.NET Core Web API

Target Framework: .NET 5.0 (Current)

Authentication Type: None

☐ Configure for HTTPS

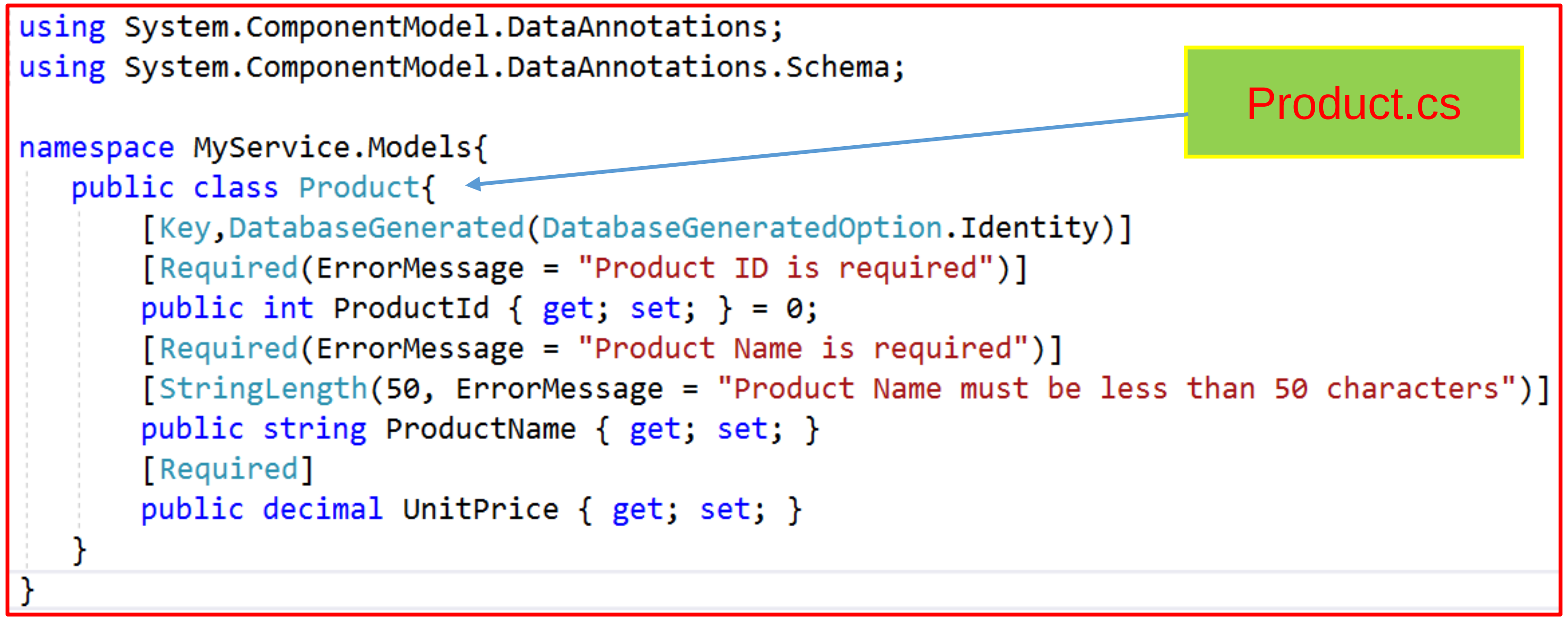
☐ Enable Docker

Docker OS: Linux

☑ Enable OpenAPI support

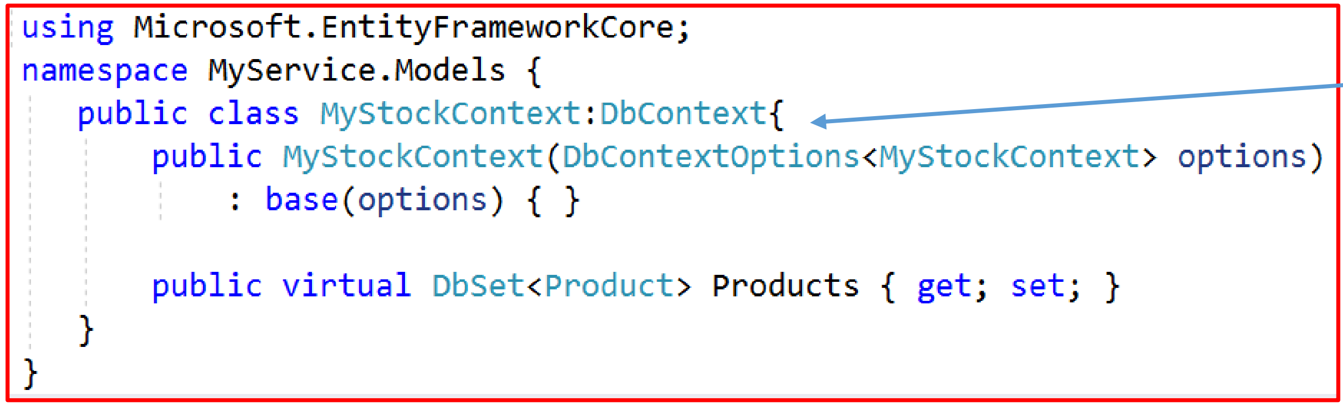
4. Install the package **Microsoft.EntityFrameworkCore.InMemory** from Nuget

5.Right-click on **Models** folder | Add | Class, named **Product.cs** and a class named **MyStockContext.cs** then write codes as follows:



Product.cs

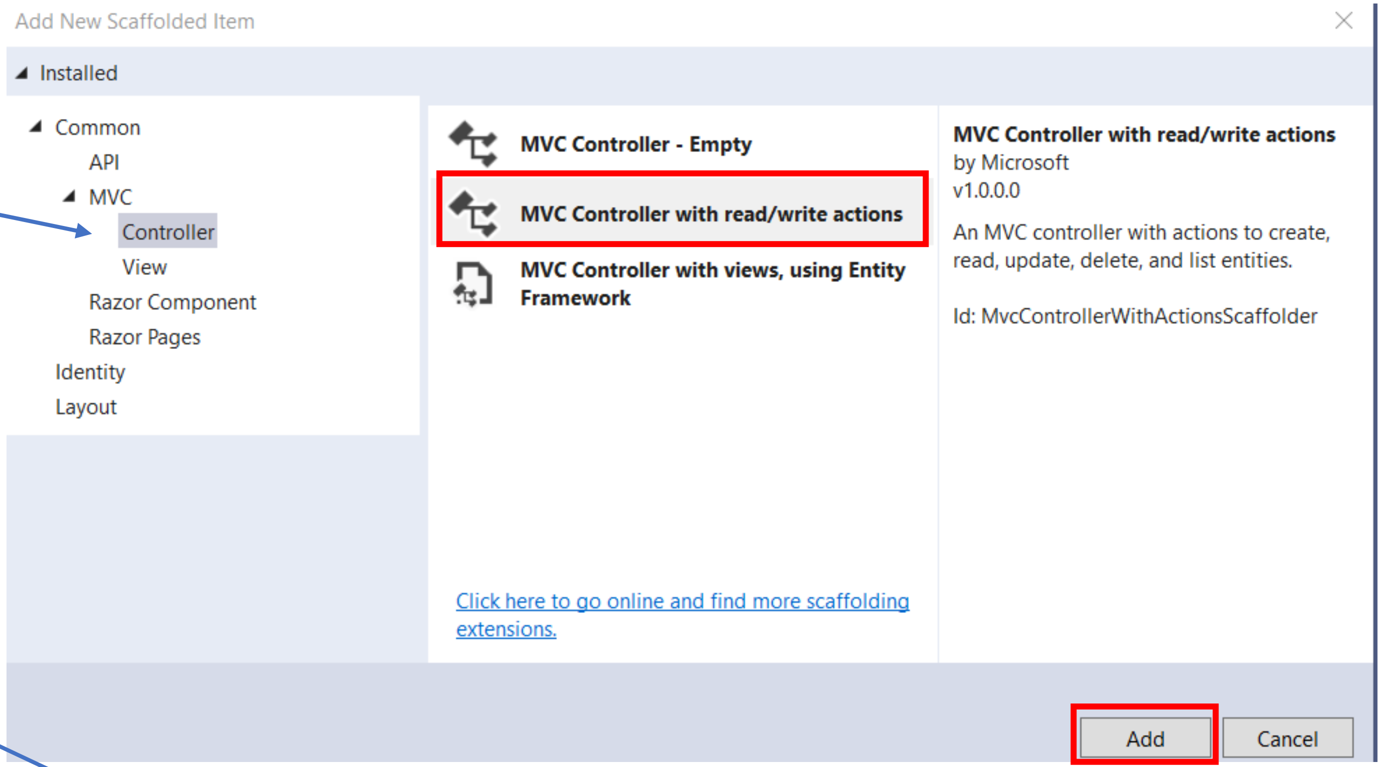
```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MyService.Models{
    public class Product{
        [Key,DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        [Required(ErrorMessage = "Product ID is required")]
        public int ProductId { get; set; } = 0;
        [Required(ErrorMessage = "Product Name is required")]
        [StringLength(50, ErrorMessage = "Product Name must be less than 50 characters")]
        public string ProductName { get; set; }
        [Required]
        public decimal UnitPrice { get; set; }
    }
}
```



MyStockContext.cs

```
using Microsoft.EntityFrameworkCore;
namespace MyService.Models {
    public class MyStockContext:DbContext{
        public MyStockContext(DbContextOptions<MyStockContext> options)
            : base(options) { }

        public virtual DbSet<Product> Products { get; set; }
    }
}
```

6.Right-click on **Controller** folder | Add | Controller, named **ProductsController.cs**


Add New Scaffolded Item

Installed
Common
API
MVC
Controller
View
Razor Component
Razor Pages
Identity
Layout
MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
Click here to go online and find more scaffolding extensions.
Add
Cancel

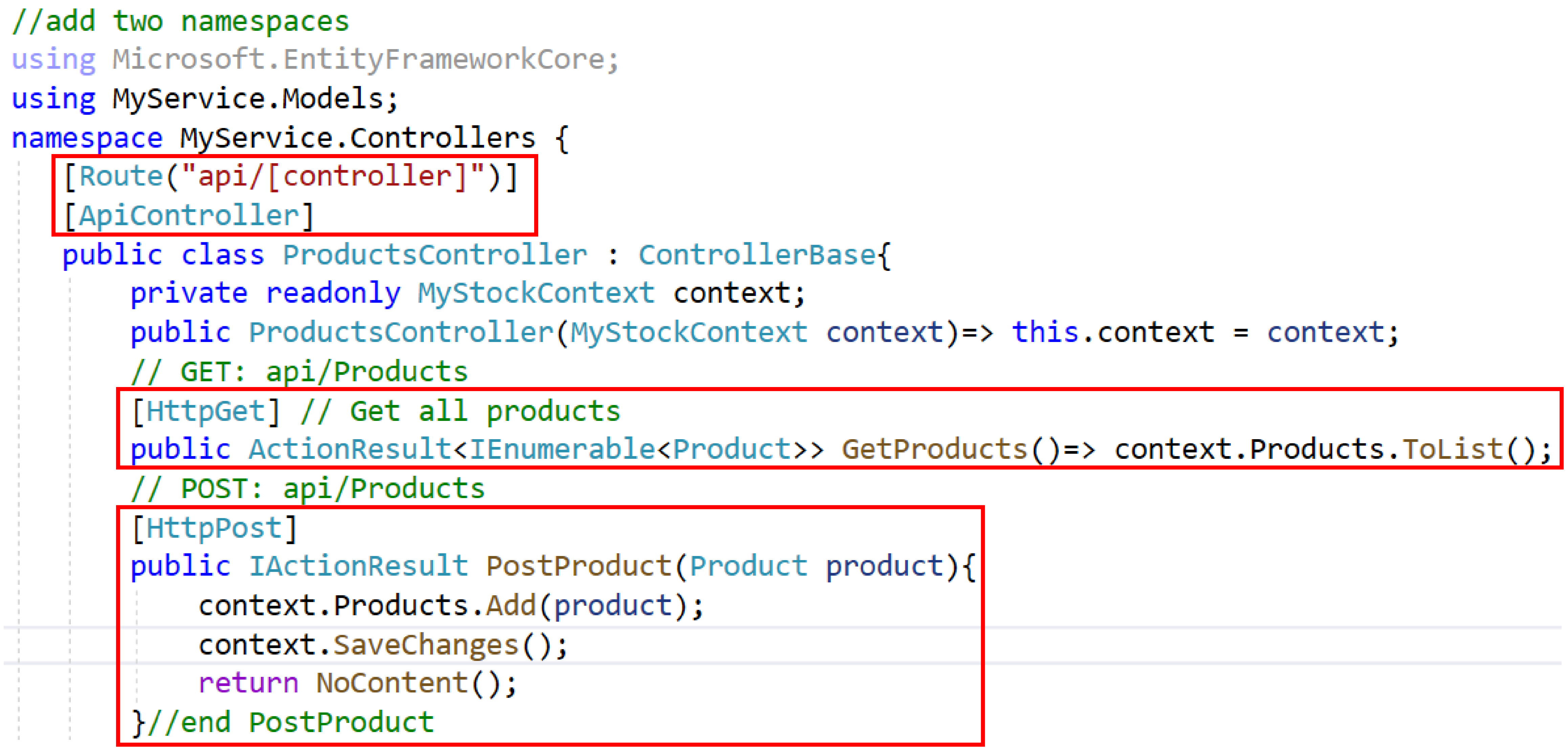
MVC Controller with read/write actions
by Microsoft
v1.0.0.0
An MVC controller with actions to create, read, update, delete, and list entities.
Id: MvcControllerWithActionsScaffolder

7.Open **Startup.cs** and update code for **ConfigureService** method as follows:

```
//add two namespaces
using MyService.Models;
using Microsoft.EntityFrameworkCore;
//…
```

```
public void ConfigureServices(IServiceCollection services){
    //Using InMemoryDatabase with Entity Framework Core
    services.AddDbContext<MyStockContext>(opt =>opt.UseInMemoryDatabase("MyStockDB"));

    services.AddControllers();
    services.AddSwaggerGen(c =>...);
}
```

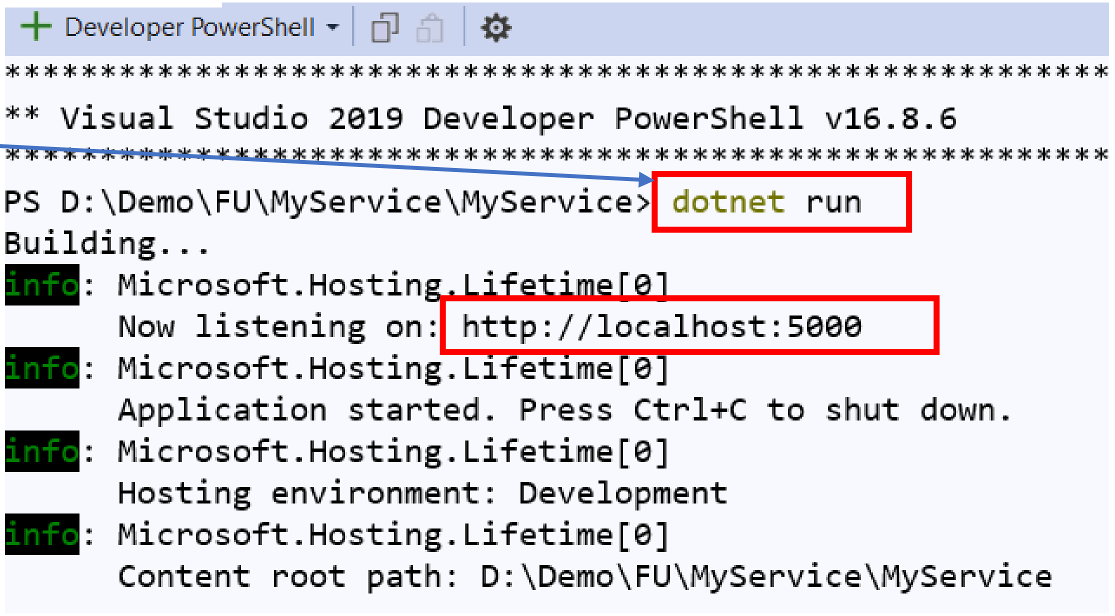
8.Write codes for **ProductsController**.**cs** as follows



```
//add two namespaces
using Microsoft.EntityFrameworkCore;
using MyService.Models;
namespace MyService.Controllers {
   [Route("api/[controller]")]
   [ApiController]
   public class ProductsController : ControllerBase{
       private readonly MyStockContext context;
       public ProductsController(MyStockContext context)=> this.context = context;
       // GET: api/Products
       [HttpGet] // Get all products
       public ActionResult<IEnumerable<Product>> GetProducts()=> context.Products.ToList();
       // POST: api/Products
       [HttpPost]
       public IActionResult PostProduct(Product product){
           context.Products.Add(product);
           context.SaveChanges();
           return NoContent();
       }//end PostProduct
```

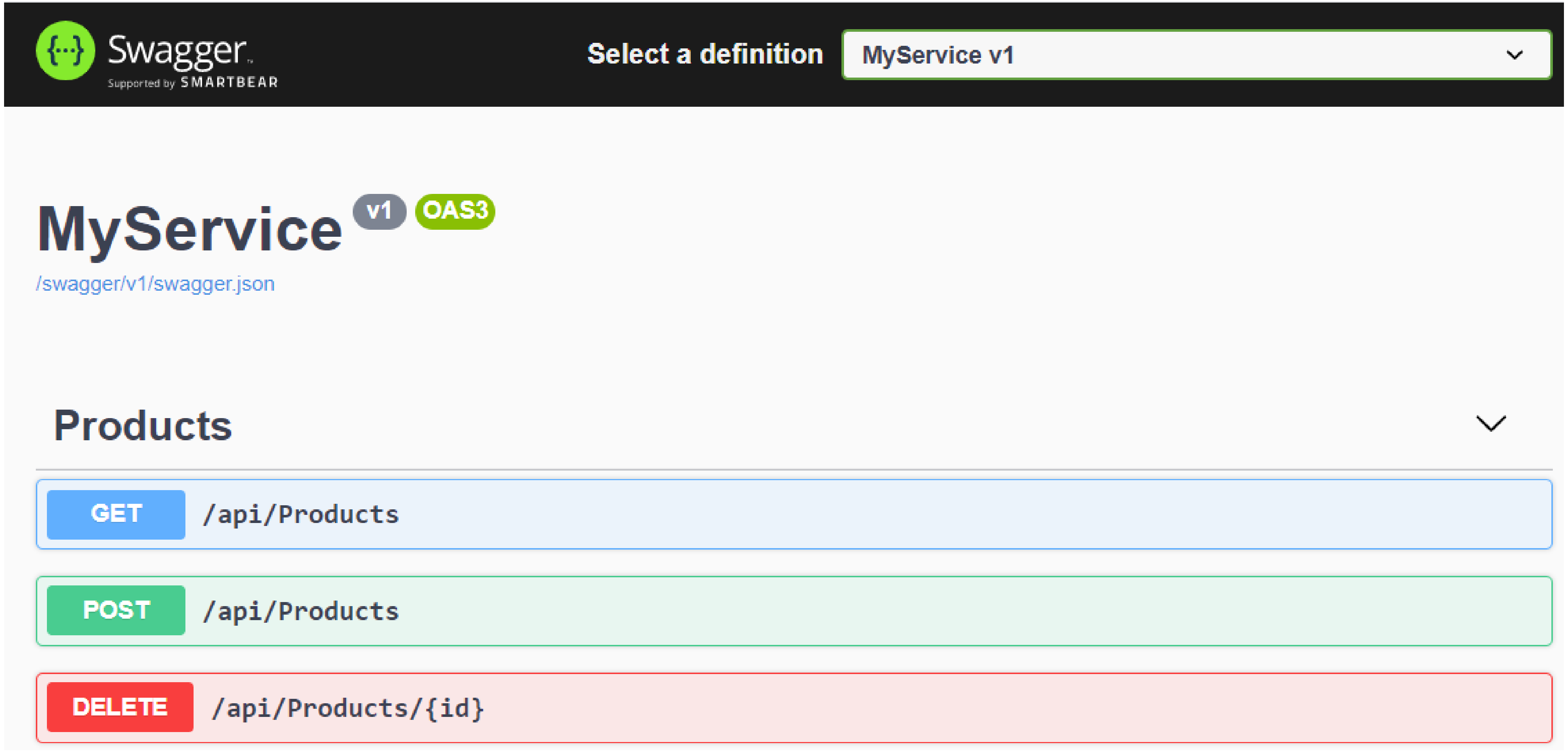
```
            // DELETE: api/Products/5
            [HttpDelete("{id}")]
            public IActionResult DeleteProduct(int id){
                var product = context.Products.Find(id);
                if (product == null){
                    return NotFound();
                }
                context.Products.Remove(product);
                context.SaveChanges();
                return NoContent();
            }//end DeleteProduct
        }//end class
    }//end namespace
```

9.Right-click on the project, select **Open in Terminal.** On **Developer PowerShell** dialog, execute the following command to run Web API project:



```
**********************************************************************
** Visual Studio 2019 Developer PowerShell v16.8.6
**********************************************************************
PS D:\Demo\FU\MyService\MyService> dotnet run
Building...
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\Demo\FU\MyService\MyService
```

10.Open the web browser and go to link to test action methods with **Swagger** :
http://localhost:5000/swagger/index.html



MyService v1 OAS3

/swagger/v1/swagger.json

Products

GET /api/Products

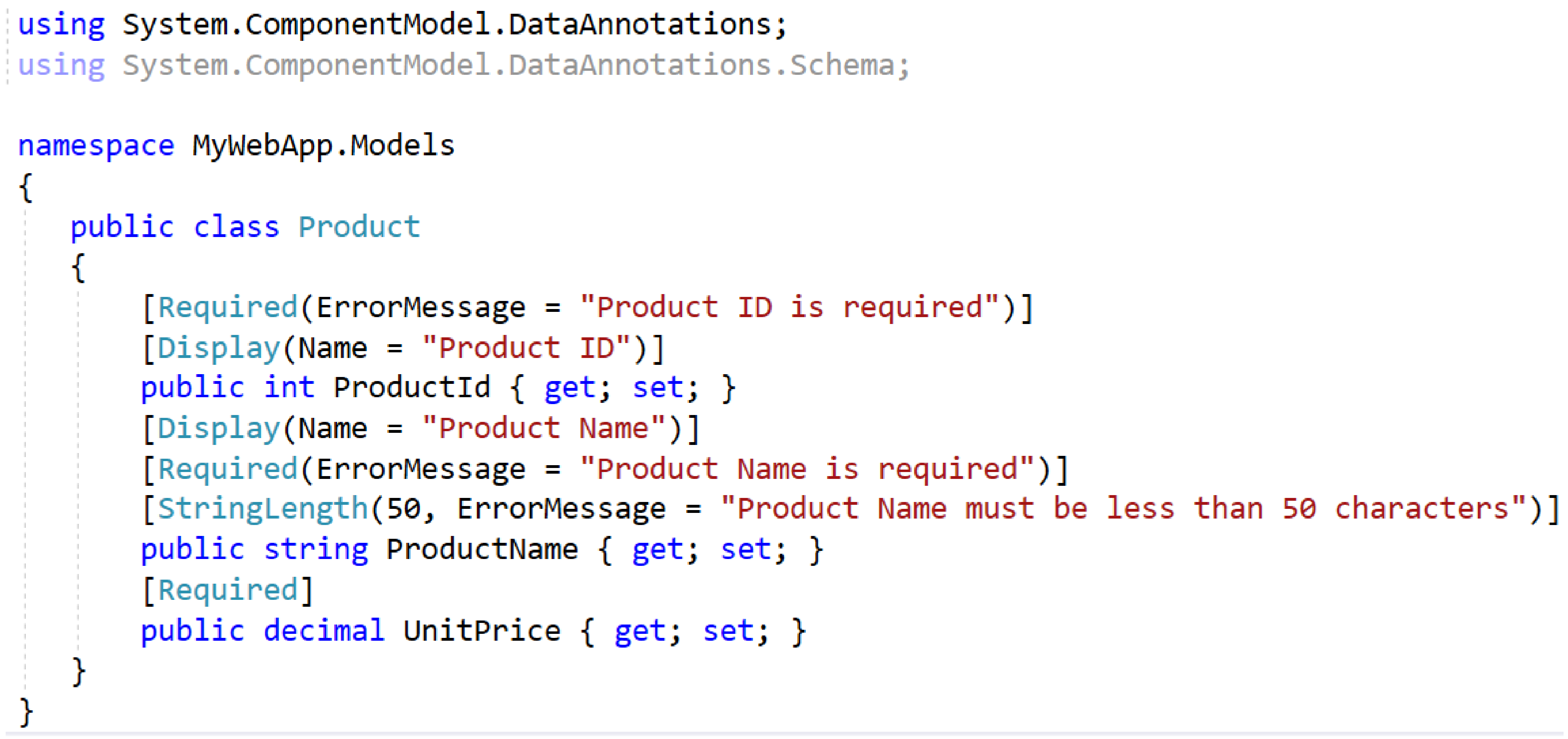
POST /api/Products

DELETE /api/Products/{id}

Demo 02- Create a ASP.NET MVC Core Application to consume Web API

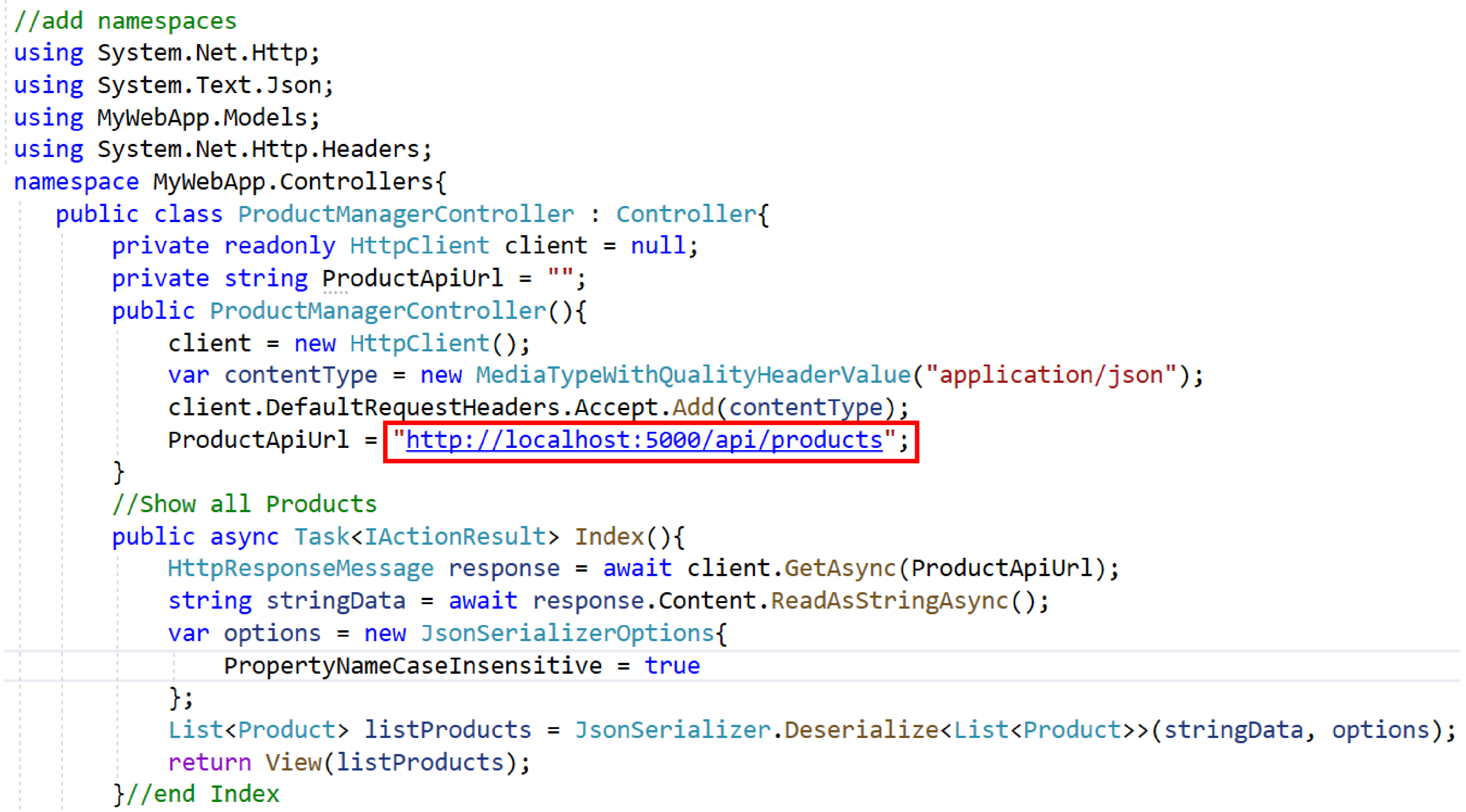
1.Create a ASP.NET MVC Core application named **MyWebApp**

2.Right-click on **Models** folder | Add | Class, named **Product.cs** and write codes as follows:

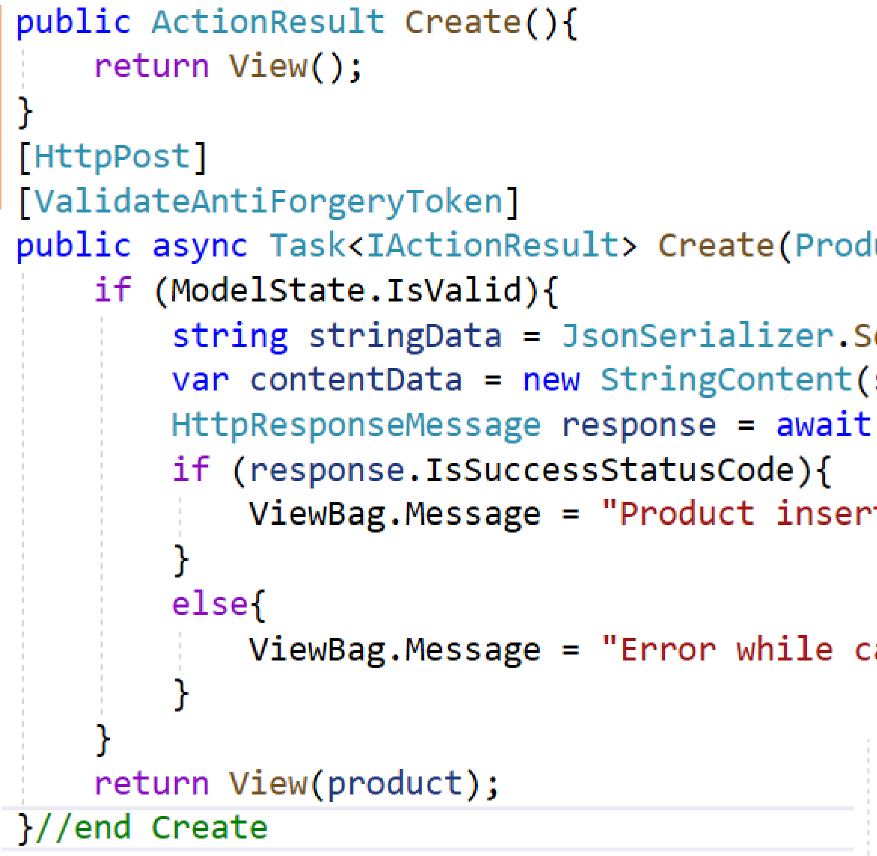


```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MyWebApp.Models
{
    public class Product
    {
        [Required(ErrorMessage = "Product ID is required")]
        [Display(Name = "Product ID")]
        public int ProductId { get; set; }
        [Display(Name = "Product Name")]
        [Required(ErrorMessage = "Product Name is required")]
        [StringLength(50, ErrorMessage = "Product Name must be less than 50 characters")]
        public string ProductName { get; set; }
        [Required]
        public decimal UnitPrice { get; set; }
    }
}
```

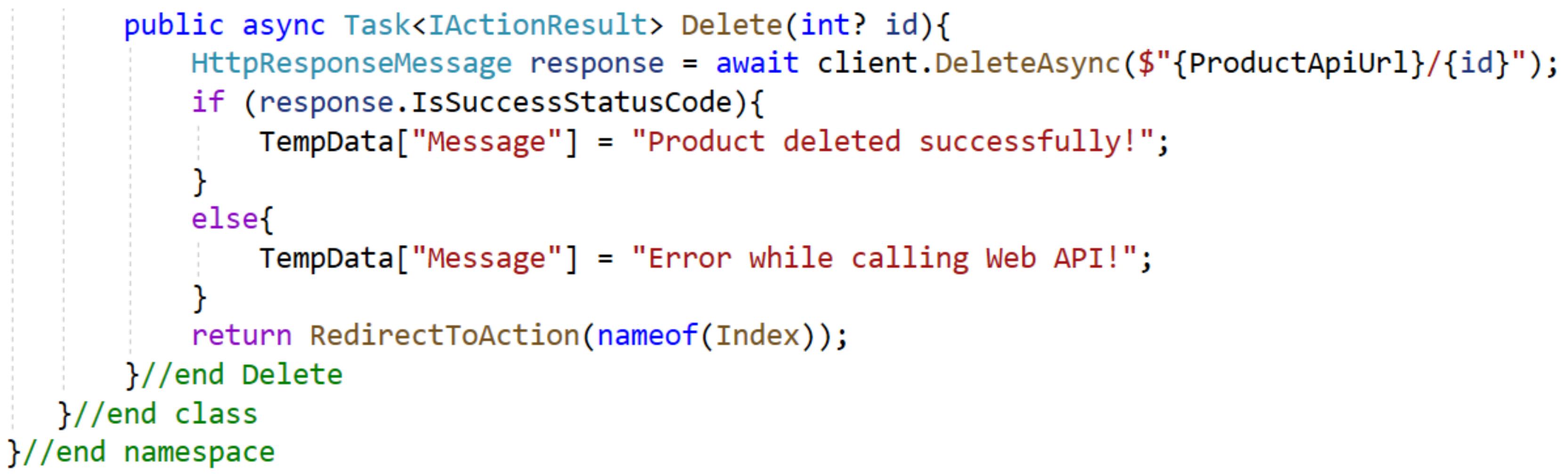
3.Right-click on **Controllers** folder | Add | Controller named **ProductManagerController.cs** and write codes as follows:



```
//add namespaces
using System.Net.Http;
using System.Text.Json;
using MyWebApp.Models;
using System.Net.Http.Headers;
namespace MyWebApp.Controllers{
   public class ProductManagerController : Controller{
       private readonly HttpClient client = null;
       private string ProductApiUrl = "";
       public ProductManagerController(){
           client = new HttpClient();
           var contentType = new MediaTypeWithQualityHeaderValue("application/json");
           client.DefaultRequestHeaders.Accept.Add(contentType);
           ProductApiUrl = "http://localhost:5000/api/products";
       }
       //Show all Products
       public async Task<IActionResult> Index(){
           HttpResponseMessage response = await client.GetAsync(ProductApiUrl);
           string stringData = await response.Content.ReadAsStringAsync();
           var options = new JsonSerializerOptions{
               PropertyNameCaseInsensitive = true
           };
           List<Product> listProducts = JsonSerializer.Deserialize<List<Product>>(stringData, options);
           return View(listProducts);
       }//end Index
```



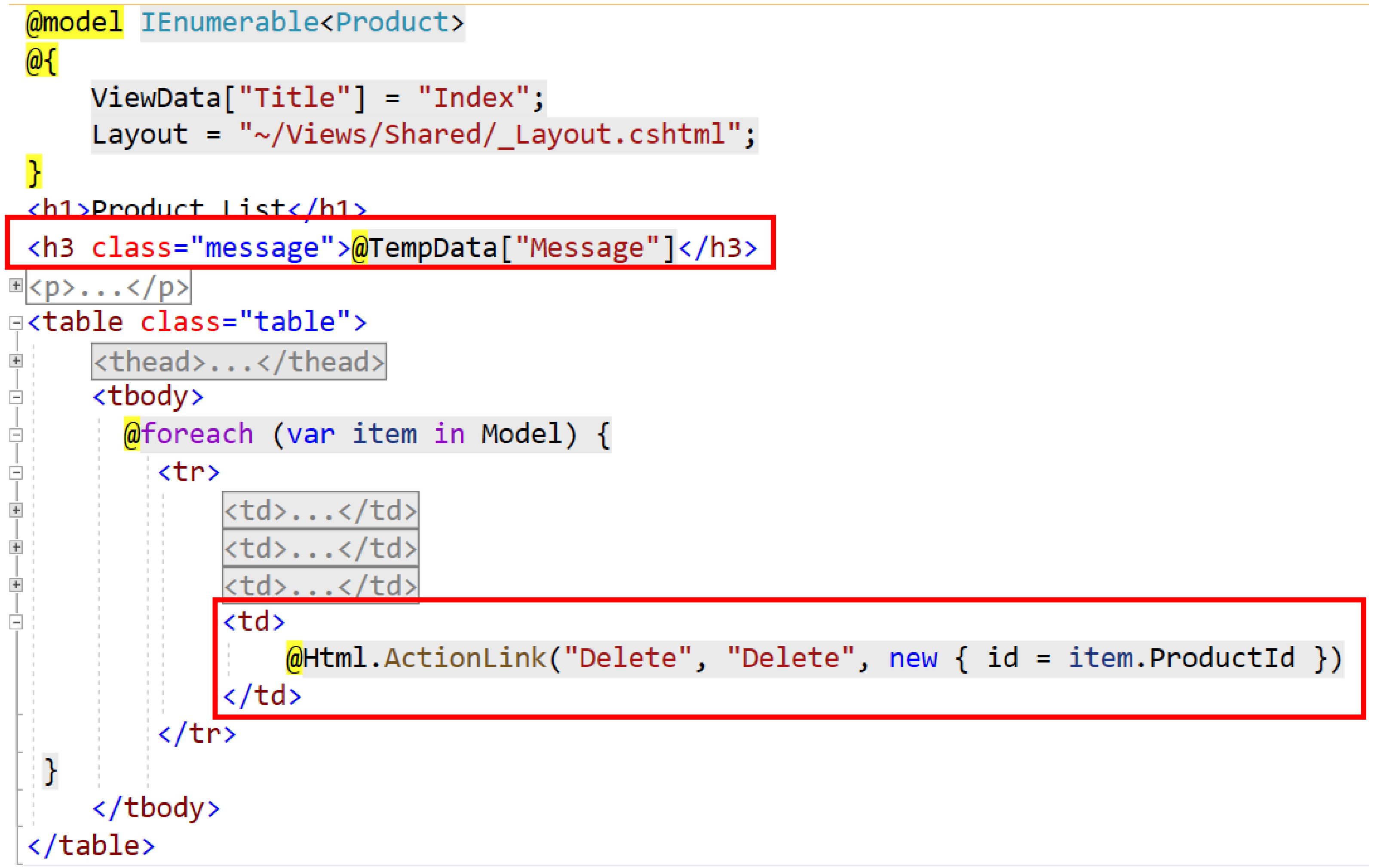
```
public ActionResult Create(){
    return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Product product){
    if (ModelState.IsValid){
        string stringData = JsonSerializer.Serialize(product);
        var contentData = new StringContent(stringData, System.Text.Encoding.UTF8, "application/json");
        HttpResponseMessage response = await client.PostAsync(ProductApiUrl, contentData);
        if (response.IsSuccessStatusCode){
            ViewBag.Message = "Product inserted successfully!";
        }
        else{
            ViewBag.Message = "Error while calling Web API!";
        }
    }
    return View(product);
}//end Create
```



```
        public async Task<IActionResult> Delete(int? id){
            HttpResponseMessage response = await client.DeleteAsync($"{ProductApiUrl}/{id}");
            if (response.IsSuccessStatusCode){
                TempData["Message"] = "Product deleted successfully!";
            }
            else{
                TempData["Message"] = "Error while calling Web API!";
            }
            return RedirectToAction(nameof(Index));
        }//end Delete
    }//end class
}//end namespace
```

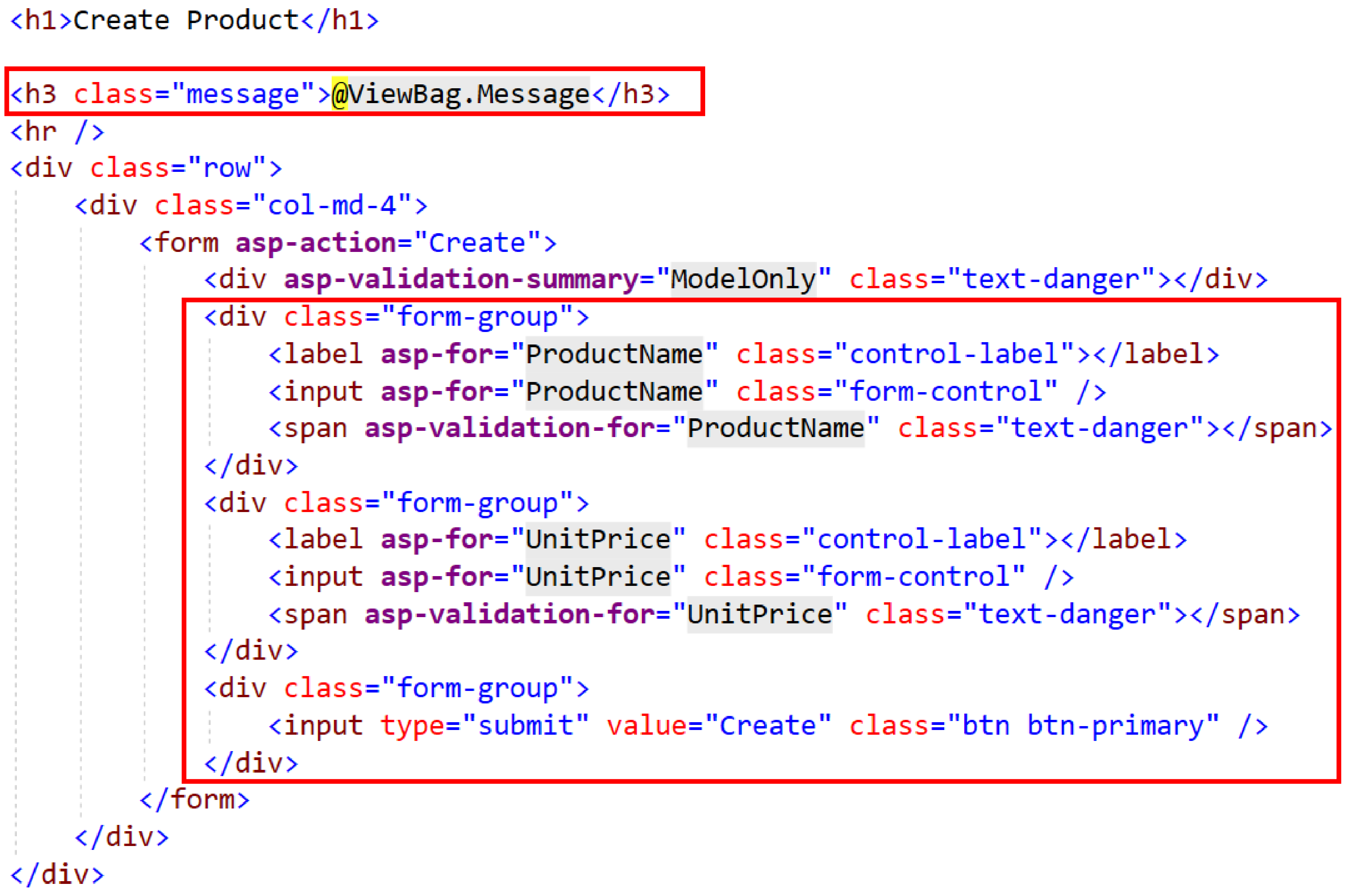
4.Right-click on **View** folder | **Add | New Folder** named **ProductManager**

5.Right-click on **ProductManager** folder | **Add** | **View** named **Index (List)** to display product list then update codes as follows:



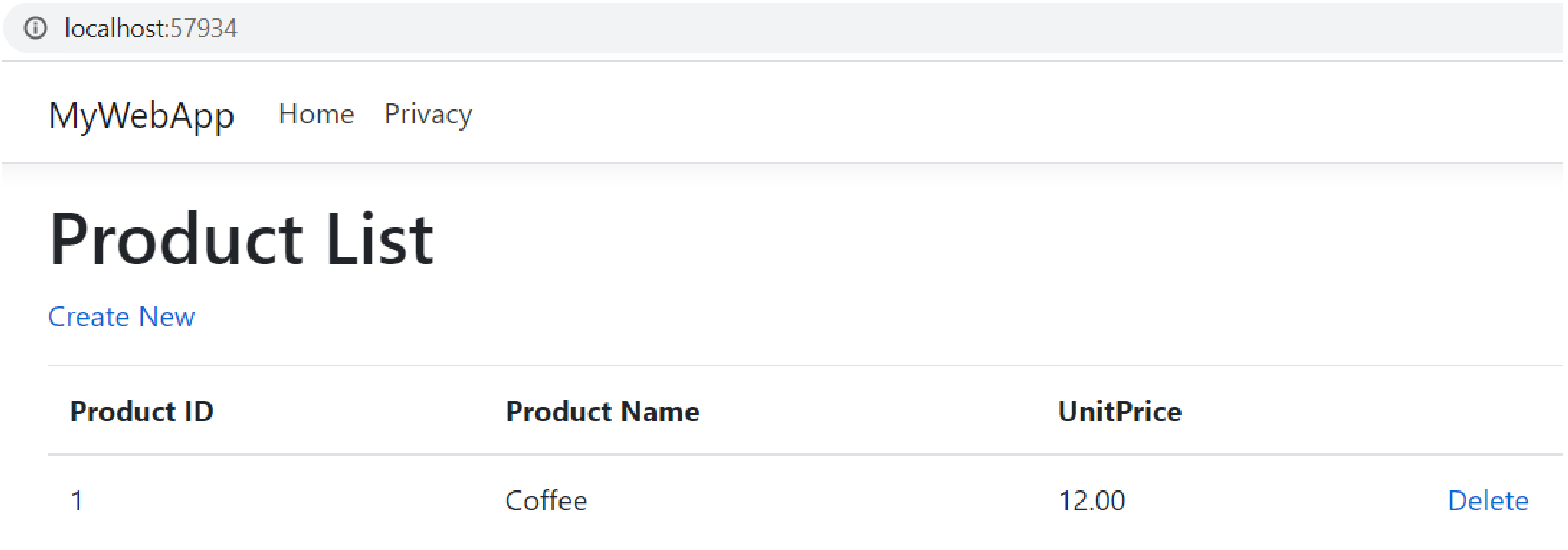
```
@model IEnumerable<Product>
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h1>Product List</h1>
<h3 class="message">@TempData["Message"]</h3>
<p>...</p>
<table class="table">
    <thead>...</thead>
    <tbody>
      @foreach (var item in Model) {
        <tr>
            <td>...</td>
            <td>...</td>
            <td>...</td>
            <td>
                @Html.ActionLink("Delete", "Delete", new { id = item.ProductId })
            </td>
        </tr>
 }
    </tbody>
</table>
```

- Repeat this step to add views: **Create** as the below figure:



```
<h1>Create Product</h1>

<h3 class="message">@ViewBag.Message</h3>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="ProductName" class="control-label"></label>
                <input asp-for="ProductName" class="form-control" />
                <span asp-validation-for="ProductName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="UnitPrice" class="control-label"></label>
                <input asp-for="UnitPrice" class="form-control" />
                <span asp-validation-for="UnitPrice" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

6.Run **MyService** project (reference to **Step 9** of **Demo-01**) then run **MyWebApp** project



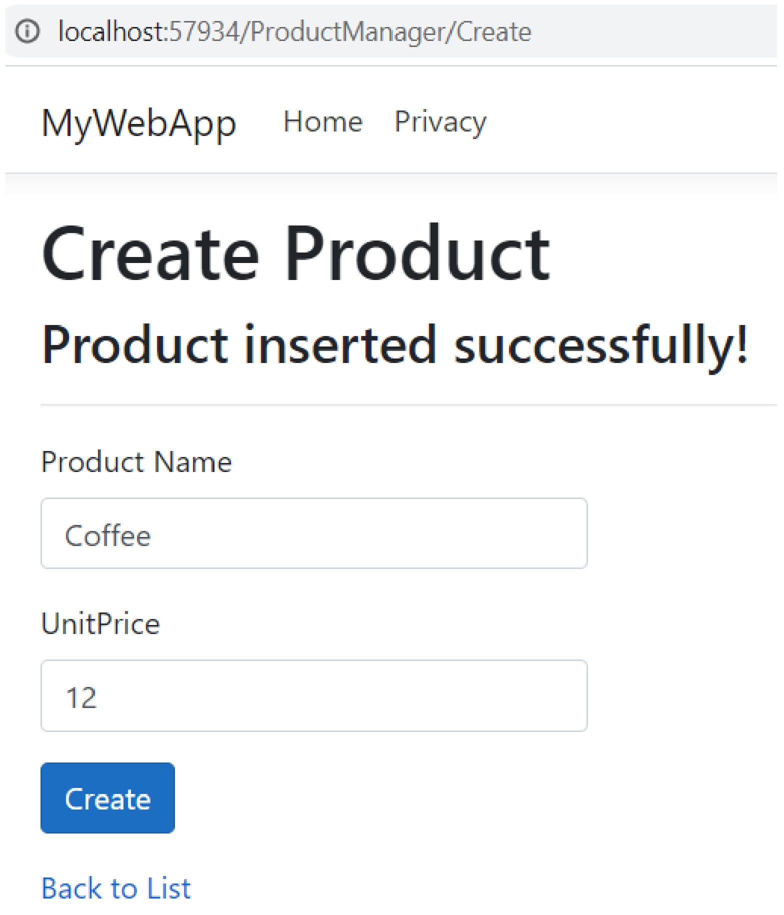
localhost:57934

MyWebApp Home Privacy

Product List

Create New

Product ID	Product Name	UnitPrice	
1	Coffee	12.00	Delete



localhost:57934/ProductManager/Create

MyWebApp Home Privacy

Create Product

Product inserted successfully!

Product Name: Coffee

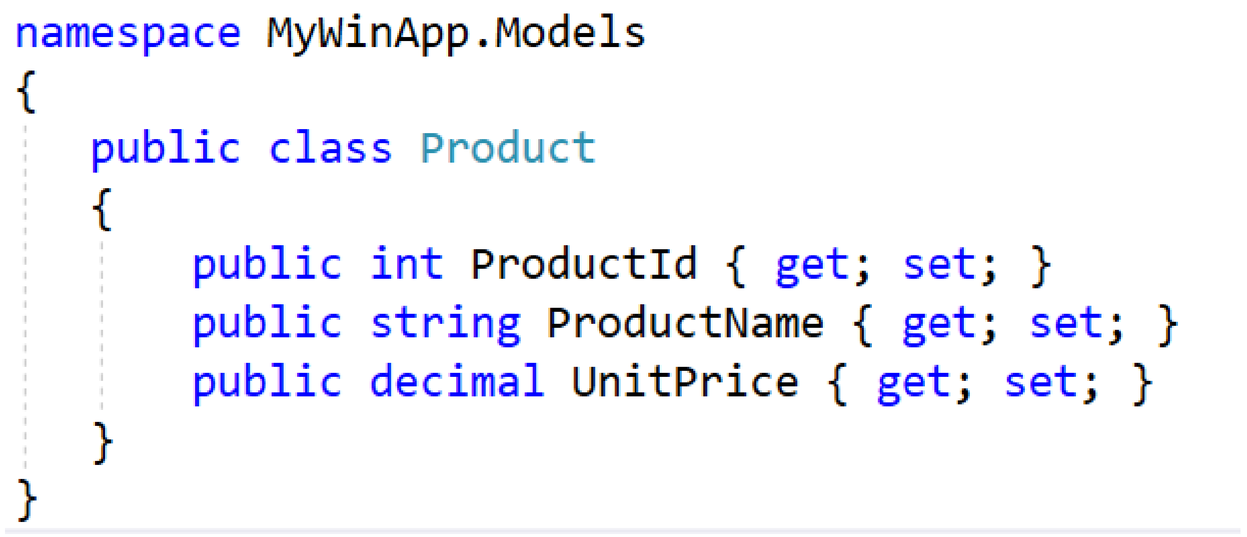
UnitPrice: 12

Demo 03- Create a Windows Forms Application to consume Web API

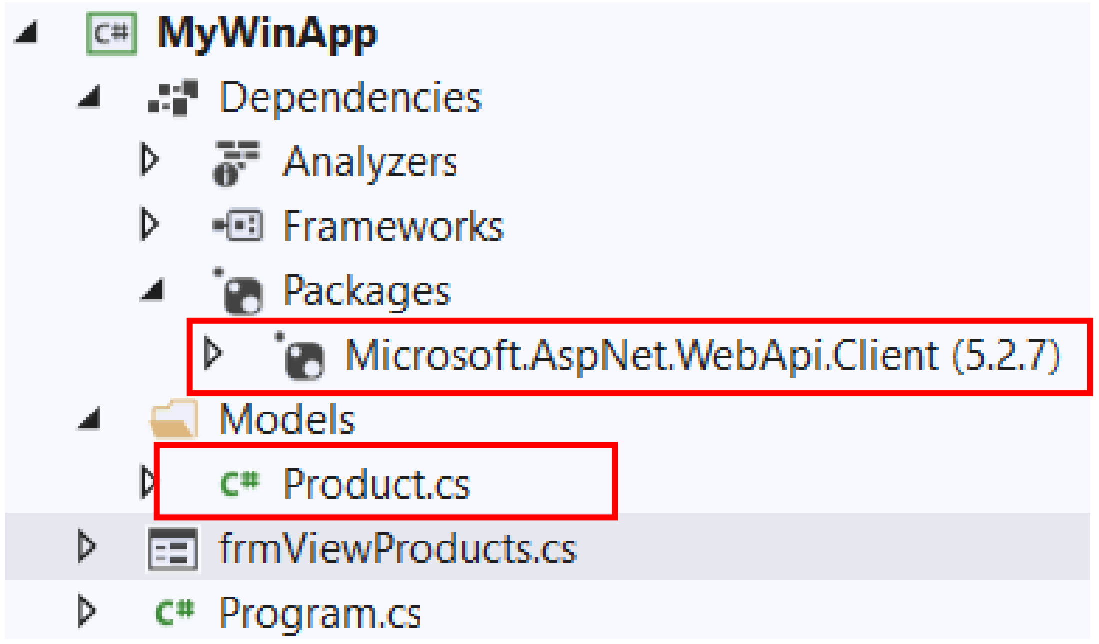
1.Create a Windows Forms application named **MyWinApp**

2. Install the package **Microsoft.AspNet.WebApi.Client** from Nuget

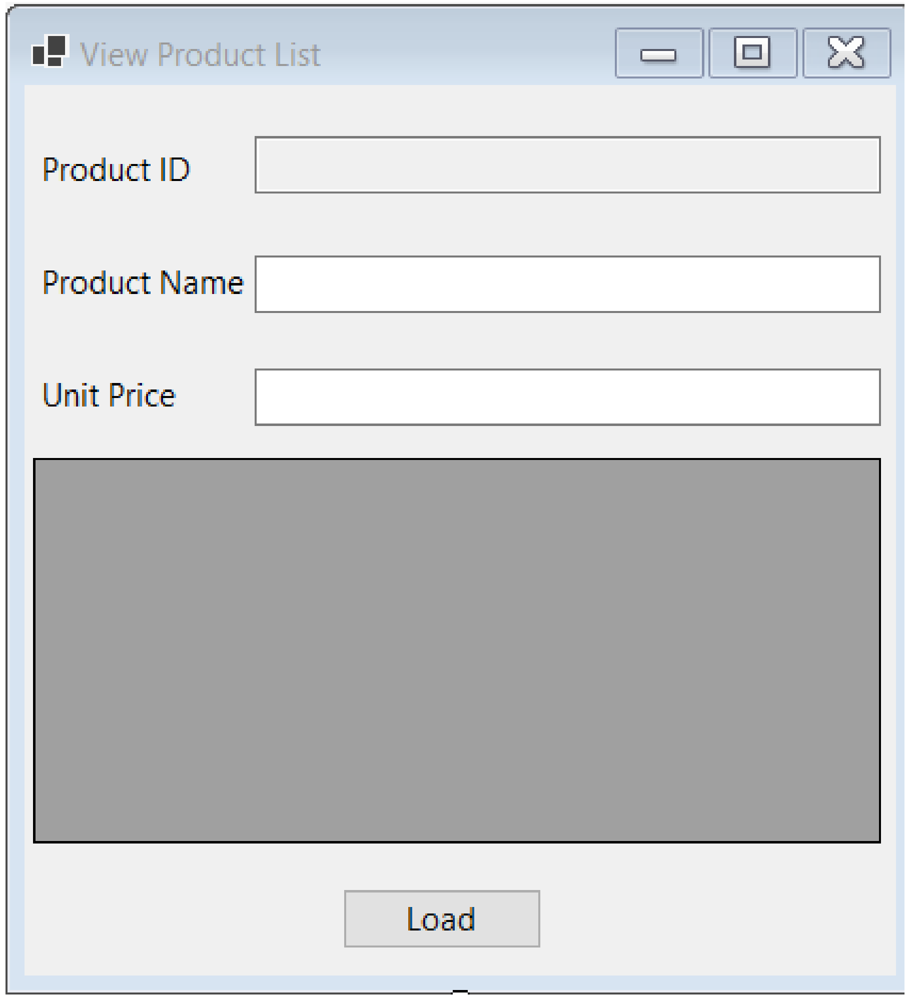
3. Right-click on the project add a folder named **Models** then add into this folder a class named **Product.cs** and write codes as follows:



```
namespace MyWinApp.Models
{
    public class Product
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public decimal UnitPrice { get; set; }
    }
}
```

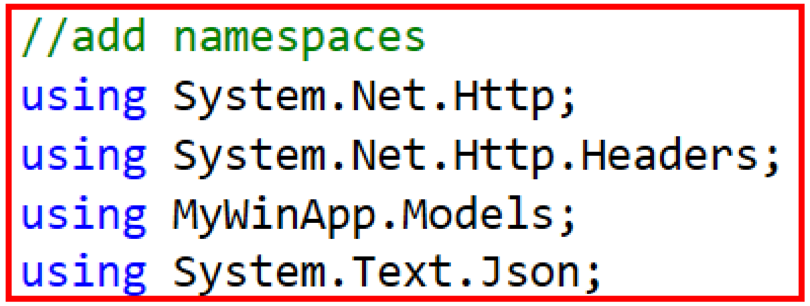


4. Create a form named **frmViewProducts** has UI as follows:

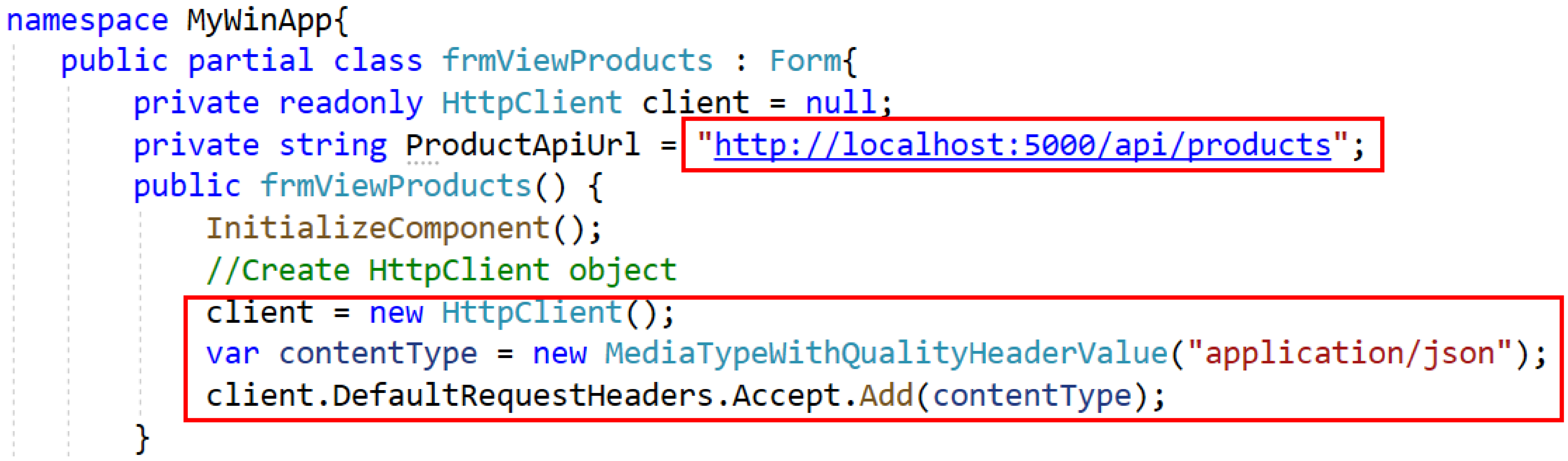


Object Type	Object name	Properties / Events
Label	lbProductID	Text: Product ID
Label	lbProductName	Text: Product Name
Label	lbUnitPrice	Text: Unit Price
TextBox	txtProductID	ReadOnly: True
TextBox	txtProductName	
TextBox	txtUnitPrice	
Button	btnLoad	Text: Delete Event Handler: Click
DataGridView	dgvProductList	ReadOnly: True SelectionMode:FullRowSelect
Form	frmViewProducts	StartPosition: CenterScreen Text: View Product List

5. Write codes in the **frmViewProducts.cs** as follows and run project:



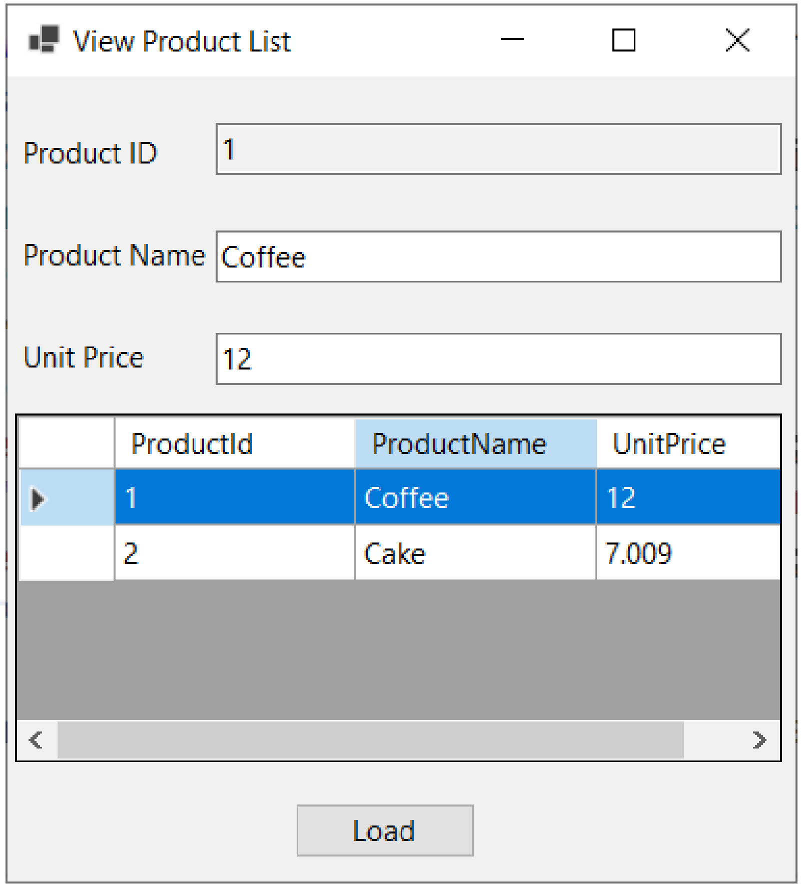
```
//add namespaces
using System.Net.Http;
using System.Net.Http.Headers;
using MyWinApp.Models;
using System.Text.Json;
```



```
namespace MyWinApp{
    public partial class frmViewProducts : Form{
        private readonly HttpClient client = null;
        private string ProductApiUrl = "http://localhost:5000/api/products";
        public frmViewProducts() {
            InitializeComponent();
            //Create HttpClient object
            client = new HttpClient();
            var contentType = new MediaTypeWithQualityHeaderValue("application/json");
            client.DefaultRequestHeaders.Accept.Add(contentType);
        }
```

TRƯỜNG ĐẠI HỌC FPT

```
        //Get product list
        private async void LoadProducts(){
            HttpResponseMessage response = await client.GetAsync(ProductApiUrl);
            string stringData = await response.Content.ReadAsStringAsync();
            var options = new JsonSerializerOptions{PropertyNameCaseInsensitive = true};
            List<Product> listProducts = JsonSerializer.Deserialize<List<Product>>(stringData, options);
            txtProductID.DataBindings.Clear();
            txtProductName.DataBindings.Clear();
            txtUnitPrice.DataBindings.Clear();
            txtProductID.DataBindings.Add("Text", listProducts, "ProductID");
            txtProductName.DataBindings.Add("Text", listProducts, "ProductName");
            txtUnitPrice.DataBindings.Add("Text", listProducts, "UnitPrice");
            dgvProductList.DataSource = listProducts;
        }//end LoadProduct
        private void btnLoad_Click(object sender, EventArgs e)=> LoadProducts();
    }//end class
}//end namespace
```


View Product List

Product ID	1
Product Name	Coffee
Unit Price	12

ProductId	ProductName	UnitPrice
1	Coffee	12
2	Cake	7.009

Lab and Assigment

1. Do Hands-on Lab:

   Lab_03_AutomobileManagement_Using_ASP.NET MVC and EF Core.pdf

2. Do Assigment:

   Assignment_03_eStoreManagement.pdf

Summary

- Concepts were introduced:
  - Overview Client-server Architecture
  - Overview ASP.NET Core and RESTful Services
  - Explain about Web Service
  - Explain about ASP.NET Web API Characteristics
  - Demo create ASP.NET Core Web API application
  - Demo create ASP.NET MVC Core Application consume Web API
  - Demo create WinForms Application consume Web API