

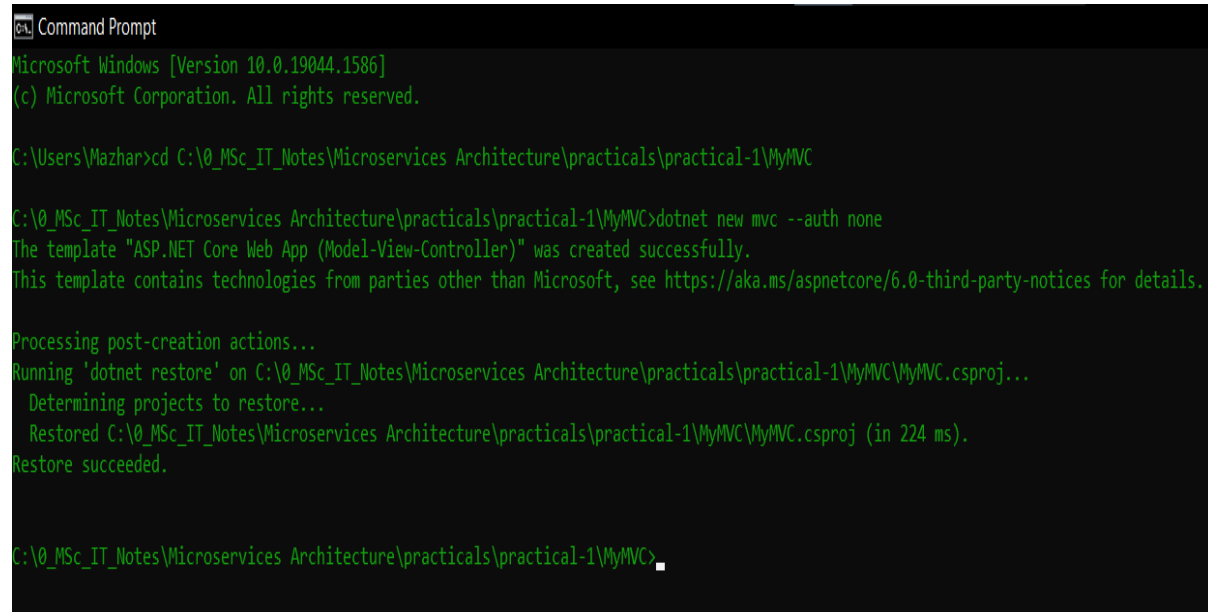
Practical 1

Aim: Create a MVC application using .Net SDK

1)Install .Net Core Sdk (Link: <https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install>)

2)create folder MyMVC folder in D: drive or any other drive

3)open command prompt and perform following operations Command: to create mvc project



```
Command Prompt
Microsoft Windows [Version 10.0.19044.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Mazhar>cd C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC

C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC>dotnet new mvc --auth none
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/6.0-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC\MyMVC.csproj...
  Determining projects to restore...
  Restored C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC\MyMVC.csproj (in 224 ms).
Restore succeeded.

C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC>
```

4) Go to controllers folder and modify HomeController.cs file to match following:

`using System.Diagnostics;`

`using Microsoft.AspNetCore.Mvc;`

`using myMvc.Models;`

`namespace MyMVC.Controllers;`

`public class HomeController : Controller`

`{`
 `private readonly ILogger<HomeController> _logger;`

`public HomeController(ILogger<HomeController> logger)`

`{`
 `_logger = logger;`
`}`

`public String Index()`

`{`
 `return "Hello world __By Mazhar Solkar";`
`}`

`}`

5) Run the project

```
C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC>dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7044
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
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: C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC\
```

Now open browser and type URL: localhost:7044



Hello World __By Mazhar Solkar

6) Now go back to command prompt and stop running project using CTRL+C

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7044
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
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: C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC\
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...

C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC>
```

7) Go to models folder and add new file StockQuote.cs to it with following content

```
namespace myMvc.Models;
public class StockQuote
{
    public string? Symbol {get;set;}
    public int Price {get;set;}
    public string? By {get;set;}
}
```

8) Now go to View folder then home folder and modify index.cshtml file to match following

```
@{  
    ViewData["Title"] = "Home Page";  
}  
  
<div class="text-center">  
    Symbol: @Model.Symbol<br/>  
    Price:$ @Model.Price<br/>  
    By:__@Model.By<br/>  
</div>
```

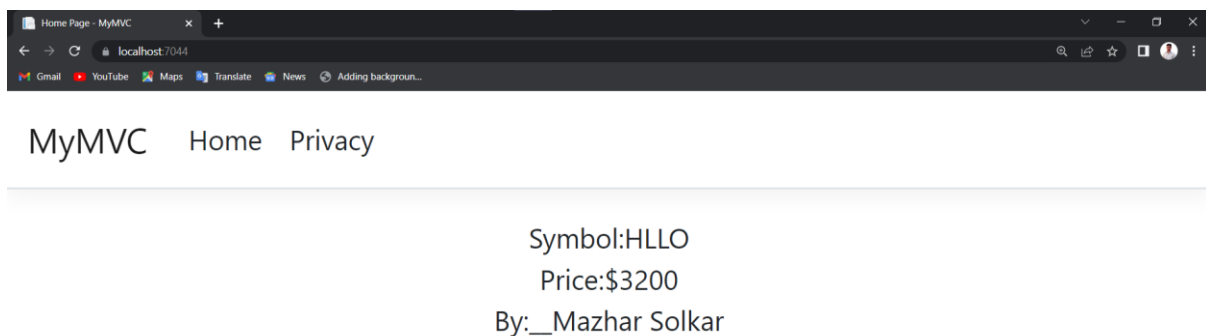
9) Now modify HomeController.cs file to match following:

```
using System.Diagnostics;  
using Microsoft.AspNetCore.Mvc;  
using myMvc.Models;  
  
namespace MyMVC.Controllers;  
  
public class HomeController : Controller  
{  
    private readonly ILogger<HomeController> _logger;  
  
    public HomeController(ILogger<HomeController> logger)  
    {  
        _logger = logger;  
    }  
  
    public IActionResult Index()  
    {  
        var model = new StockQuote{Symbol="HLLO",Price=3200,By="Mazhar Solkar"};  
        return View(model);  
    }  
}
```

10) Now run the project using

```
C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC>dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7044
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5086
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: C:\0_MSc_IT_Notes\Microservices Architecture\practicals\practical-1\MyMVC\
```

11) Now go back to browser and refresh to get modified view response



Practical 2

Aim: Building ASP.Net core REST API.

Software requirement :

1. Download and install

To start building .NET apps you just need to download and install the .NET SDK (Software Development Kit version 3.0 above).

Link:

<https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install>

2. Check you've installed, open new command prompt and run the following command.

>dotnet

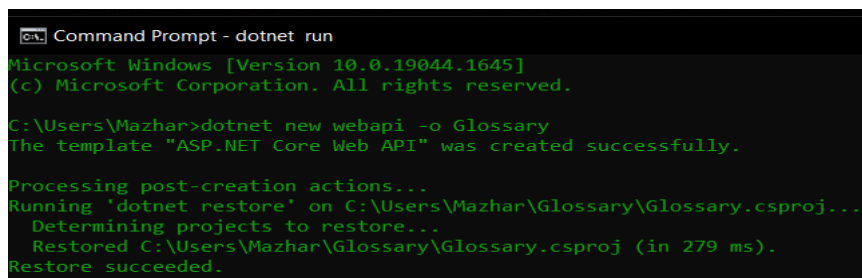
Create your web API

1. Open command prompt

Command :

dotnet new webapi -o Glossary

Output :-



```
Command Prompt - dotnet run
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Mazhar>dotnet new webapi -o Glossary
The template "ASP.NET Core Web API" was created successfully.

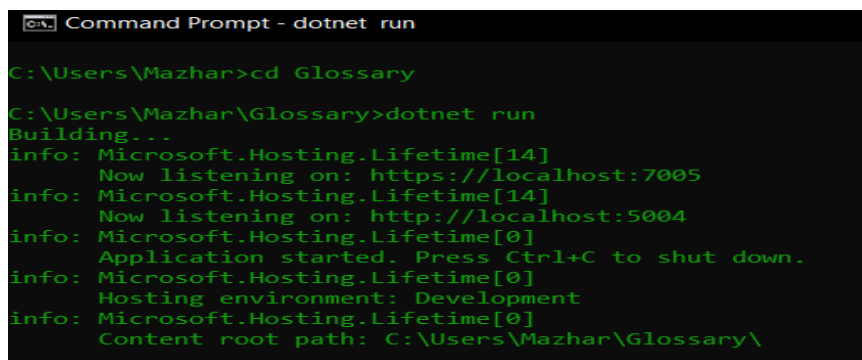
Processing post-creation actions...
Running 'dotnet restore' on C:\Users\Mazhar\Glossary\Glossary.csproj...
  Determining projects to restore...
  Restored C:\Users\Mazhar\Glossary\Glossary.csproj (in 279 ms).
Restore succeeded.
```

Command :

cd Glossary

dotnet run

Output :-

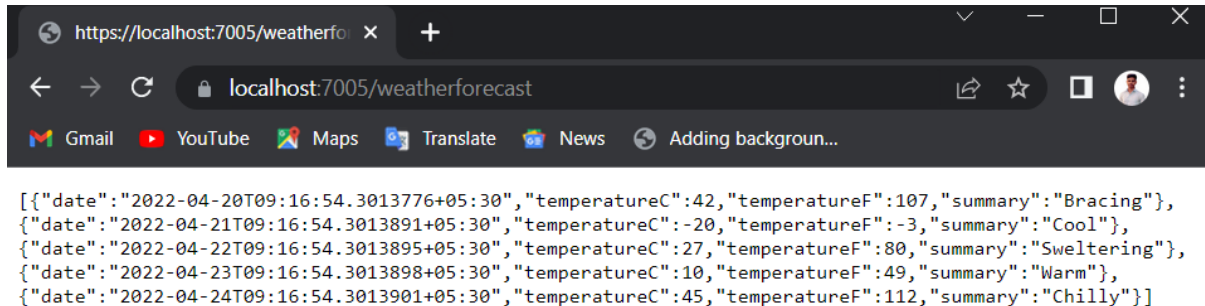


```
Command Prompt - dotnet run

C:\Users\Mazhar>cd Glossary
C:\Users\Mazhar\Glossary>dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7005
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5004
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: C:\Users\Mazhar\Glossary\
```

2. If everything is okay, point your browser to the address <https://localhost:7005/weatherforecast> You should see a page similar to the following:

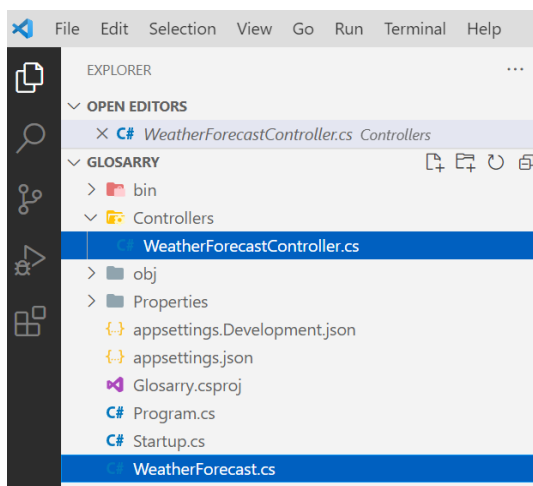
Output:



3. Now change the content:

To get started, go to C:\Users\Mazhar\Glosarry

remove the WeatherForecast.cs file from the root of the project and the WeatherForecastController.cs file from the Controllers folder.



Add Following two files

1) C:\Users\Mazhar\GlossaryController.cs (type the following code)

```
//GlossaryItem.cs
namespace Glossary
{
    public class GlossaryItem
    {
        public string? Term { get; set; }
        public string? Definition { get; set; }
    }
}
```

2) C:\Users\Mazhar\Glossary\Controllers\GlossaryController.cs (type the following code)

```
//Controllers/GlossaryController.cs
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using System.IO;
namespace Glossary.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class GlossaryController : ControllerBase
    {
        private static List<GlossaryItem> Glossary = new List<GlossaryItem> {
new GlossaryItem
{
    Term= "HTML",
    Definition = "Hypertext Markup Language"
},
new GlossaryItem
{
    Term= "MVC",
    Definition = "Model View Controller"
},
new GlossaryItem
{
    Term= "OpenID",
    Definition = "An open standard for authentication"
}
};

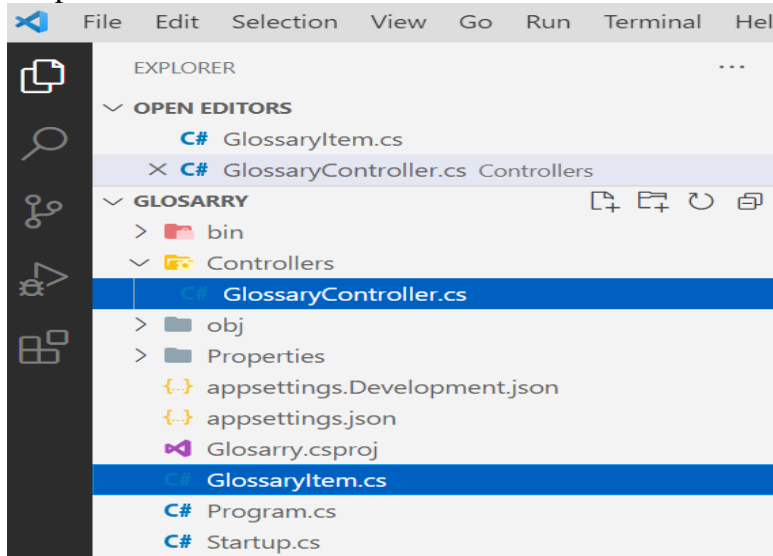
        [HttpGet]
        public ActionResult<List<GlossaryItem>> Get()
        {
            return Ok(Glossary);
        }
        [HttpGet]
        [Route("{term}")]
        public ActionResult<GlossaryItem> Get(string term)
        {
            var glossaryItem = Glossary.Find(item =>
            item.Term.Equals(term,StringComparison.InvariantCultureIgnoreCase));
            if (glossaryItem == null)
            {
                return NotFound();
            }
            else
            {
                return Ok(glossaryItem);
            }
        }
    }
}
```

```
    }  
  }  
  [HttpPost]  
  public ActionResult Post(GlossaryItem glossaryItem)  
  {  
    var existingGlossaryItem = Glossary.Find(item =>  
      item.Term.Equals(glossaryItem.Term,StringComparison.InvariantCultureIgnoreCase)  
);  
    if (existingGlossaryItem != null)  
    {  
      return Conflict("Cannot create the term because it already exists.");  
    }  
    else  
    {  
      Glossary.Add(glossaryItem);  
      var resourceUrl = Path.Combine(Request.Path.ToString(),  
Uri.EscapeUriString(glossaryItem.Term));  
      return Created(resourceUrl, glossaryItem);  
    }  
  }  
  [HttpPut]  
  public ActionResult Put(GlossaryItem glossaryItem)  
  {  
    var existingGlossaryItem = Glossary.Find(item =>  
      item.Term.Equals(glossaryItem.Term,StringComparison.InvariantCultureIgnoreCase)  
);  
    if (existingGlossaryItem == null)  
    {  
      return BadRequest("Cannot update a nont existing term.");  
    }  
    else  
    {  
      existingGlossaryItem.Definition = glossaryItem.Definition;  
      return Ok();  
    }  
  }  
  [HttpDelete]  
  [Route("{term}")]  
  public ActionResult Delete(string term)  
  {  
    var glossaryItem = Glossary.Find(item =>  
      item.Term.Equals(term,StringComparison.InvariantCultureIgnoreCase));  
    if (glossaryItem == null)  
    {  
      return NotFound();  
    }  
    else  
    {  
      }
```



```
Glossary.Remove(glossaryItem);  
return NoContent();  
}  
}  
}  
}
```

Output :-

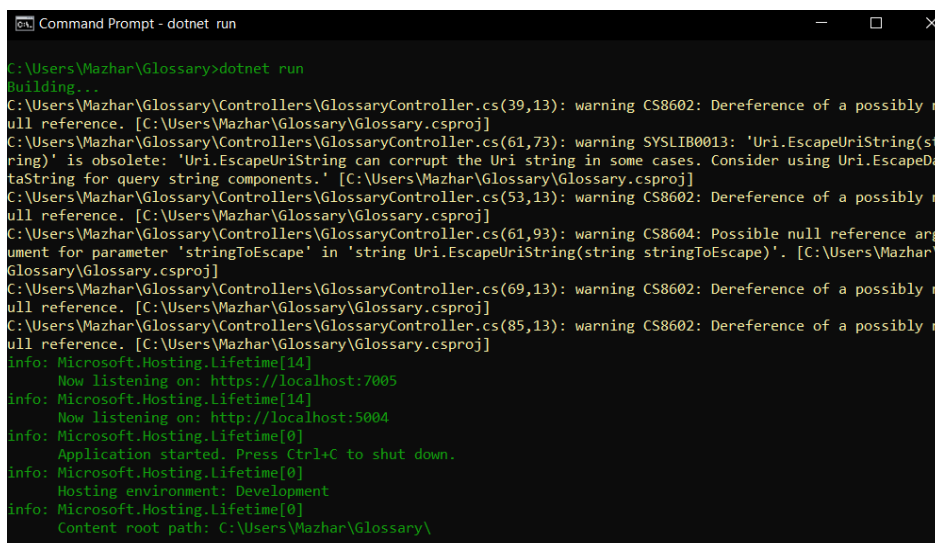


4. Now stop running previous dotnet run on command prompt using Ctrl+C and Run it again for new code.

On Command prompt:

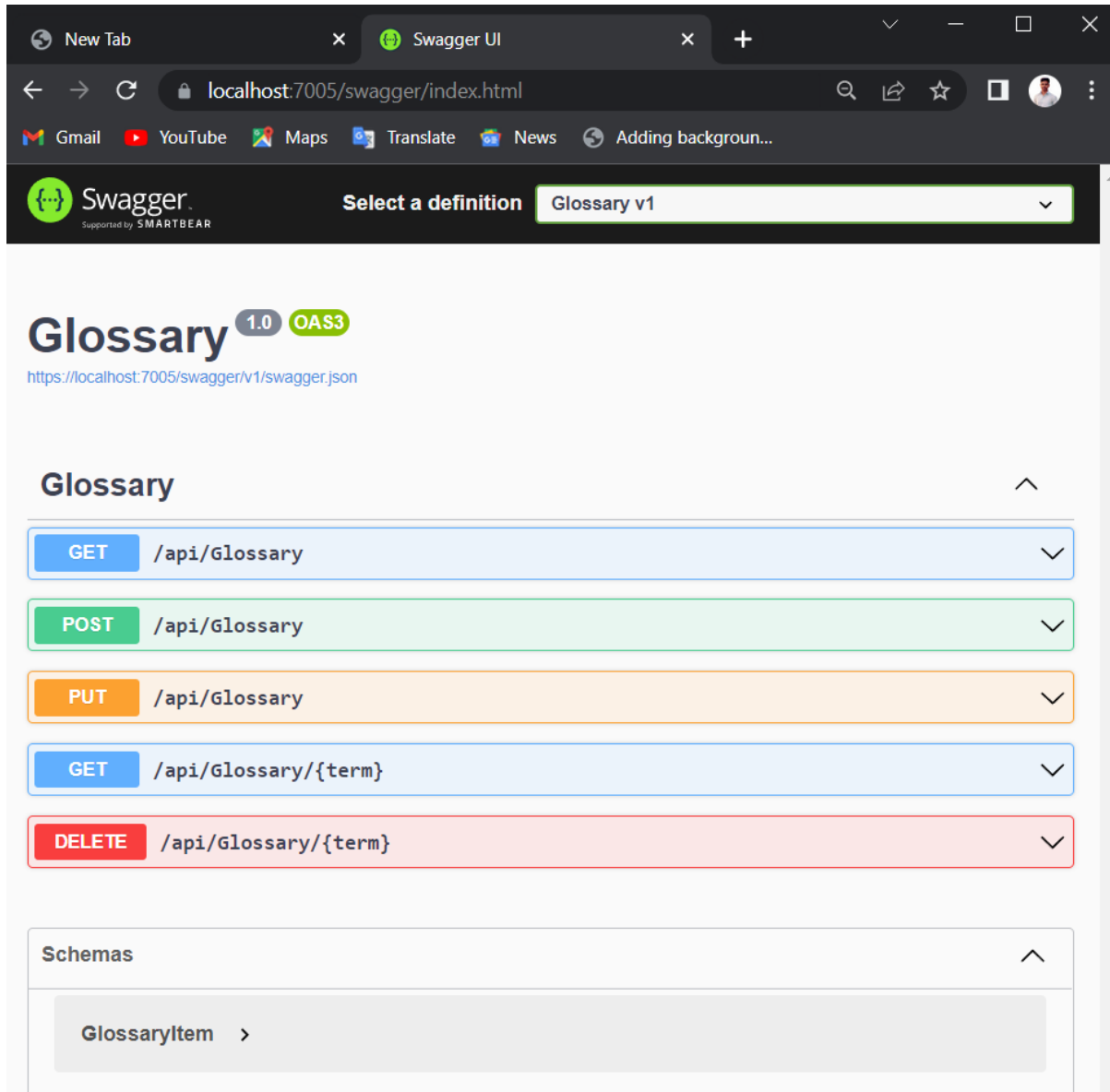
dotnet run

Output :-



5. Then, navigate to <https://localhost:7005/swagger> with your browser.

Output :-



1) Getting a list of items:

Click on try it out the execute

The screenshot displays a REST client interface with the following components:

- Method and URL:** A dropdown menu shows 'GET' and the URL is '/api/Glossary'.
- Parameters:** A section labeled 'Parameters' with a 'Cancel' button and the text 'No parameters'.
- Buttons:** 'Execute' and 'Clear' buttons at the bottom of the request section.
- Responses:** A section titled 'Responses' containing:
 - Curl:** A code block with the command:

```
curl -X 'GET' \
'https://localhost:7005/api/Glossary' \
-H 'accept: text/plain'
```
 - Request URL:** A text field containing 'https://localhost:7005/api/Glossary'.
 - Server response:** A table with two columns: 'Code' and 'Details'.

Code	Details
200	<p>Response body</p> <pre>[{ "term": "HTML", "definition": "Hypertext Markup Language" }, { "term": "MVC", "definition": "Model View Controller" }, { "term": "OpenID", "definition": "An open standard for authentication" }]</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Tue, 19 Apr 2022 05:32:56 GMT server: Kestrel</pre>
- Summary Table:** A table at the bottom with columns 'Code', 'Description', and 'Links'.

Code	Description	Links
200	Success	No links

2) Getting a single item:

GET

/api/Glossary/{term}

Parameters

Cancel

Name	Description
term * required string (path)	<input type="text" value="html"/>

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'https://localhost:7005/api/Glossary/html' \
-H 'accept: text/plain'
```

Request URL

```
https://localhost:7005/api/Glossary/html
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{ "term": "HTML", "definition": "Hypertext Markup Language" }</pre></div><div><div>Download</div></div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Tue, 19 Apr 2022 05:36:02 GMT server: Kestrel</pre></div></div>

Responses

Code	Description	Links
200	Success	No links

3) Creating an item

POST /api/Glossary

Parameters

Cancel

Reset

No parameters

Request body

application/json

```
{  "term": "MFA",  "definition": "an Authentication process"}  
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \  'https://localhost:7005/api/Glossary' \  -H 'accept: */*' \  -H 'Content-Type: application/json' \  -d '{  "term": "MFA",  "definition": "an Authentication process"}'
```

Request URL

https://localhost:7005/api/Glossary

Server response

Code

Details

201

Undocumented

Response body

```
{  "term": "MFA",  "definition": "an Authentication process"}  
```

Download

Response headers

```
content-type: application/json; charset=utf-8  date: Tue, 19 Apr 2022 05:40:15 GMT  location: /api/Glossary\MFA  server: Kestrel
```

Responses

Code	Description	Links
200	Success	No links

4)Update Item

PUT /api/Glossary

Parameters

Cancel

Reset

No parameters

Request body

application/json

```
{  "term": "MVC",  "definition": "Modified record of Model View Controller"}  
```

Execute

Clear

Responses

Curl

```
curl -X 'PUT' \  'https://localhost:7005/api/Glossary' \  -H 'accept: */*' \  -H 'Content-Type: application/json' \  -d '{  "term": "MVC",  "definition": "Modified record of Model View Controller"}'
```

Request URL

https://localhost:7005/api/Glossary

Server response

Code	Details
200	<div>Response headers<div>content-length: 0date: Tue, 19 Apr 2022 04:13:49 GMTserver: Kestrel</div></div>

Responses

Code	Description	Links
200	Success	No links

5) Delete Item

DELETE /api/Glossary/{term}

Parameters

Cancel

Name	Description
term * required string (path)	<input type="text" value="openid"/>

Execute

Clear

Responses

Curl

```
curl -X 'DELETE' \
'https://localhost:7005/api/Glossary/openid' \
-H 'accept: */*'
```

Request URL

```
https://localhost:7005/api/Glossary/openid
```

Server response

Code	Details
204 <i>Undocumented</i>	<div><div>Response headers</div><div><pre>date: Tue, 19 Apr 2022 04:16:15 GMT server: Kestrel</pre></div></div>

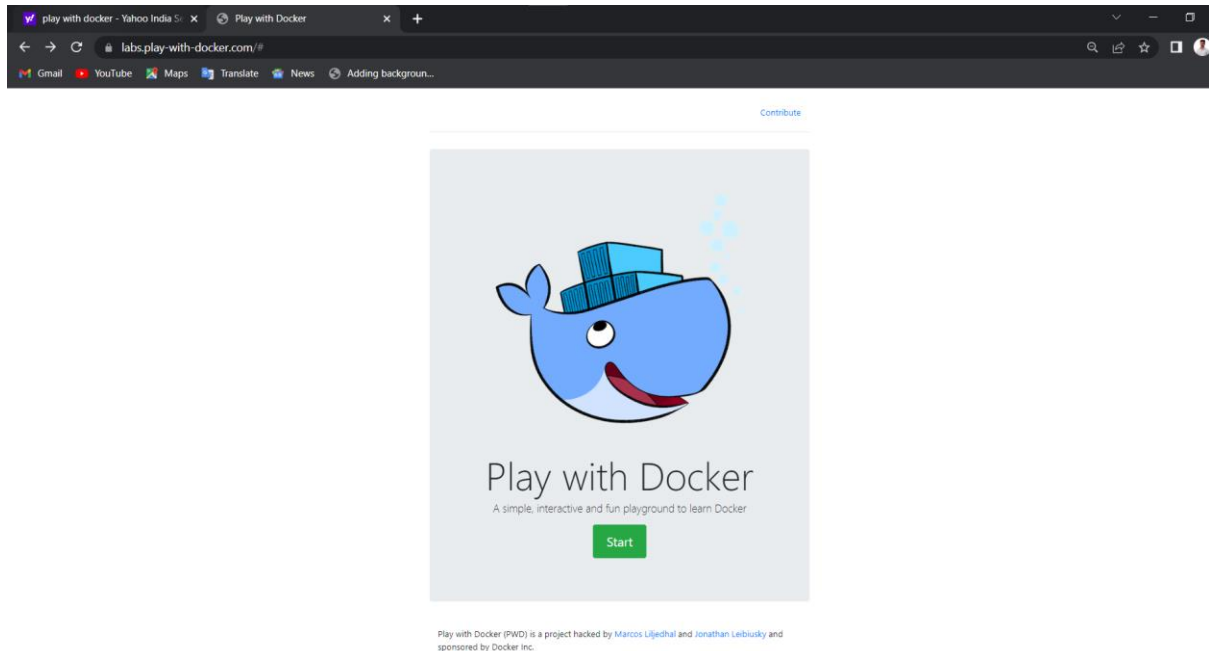
Responses

Code	Description	Links
200	Success	No links

Practical 3

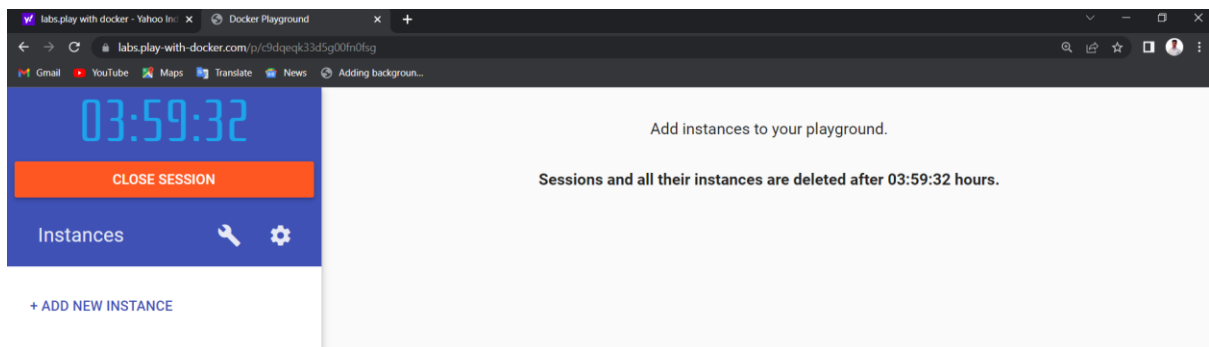
Aim: Working with Docker

- 1) create Docker Hub account (sign up)
- 2) login to <https://labs.play-with-docker.com/>



Click on start

- 3)add new instance.



4)perform following:

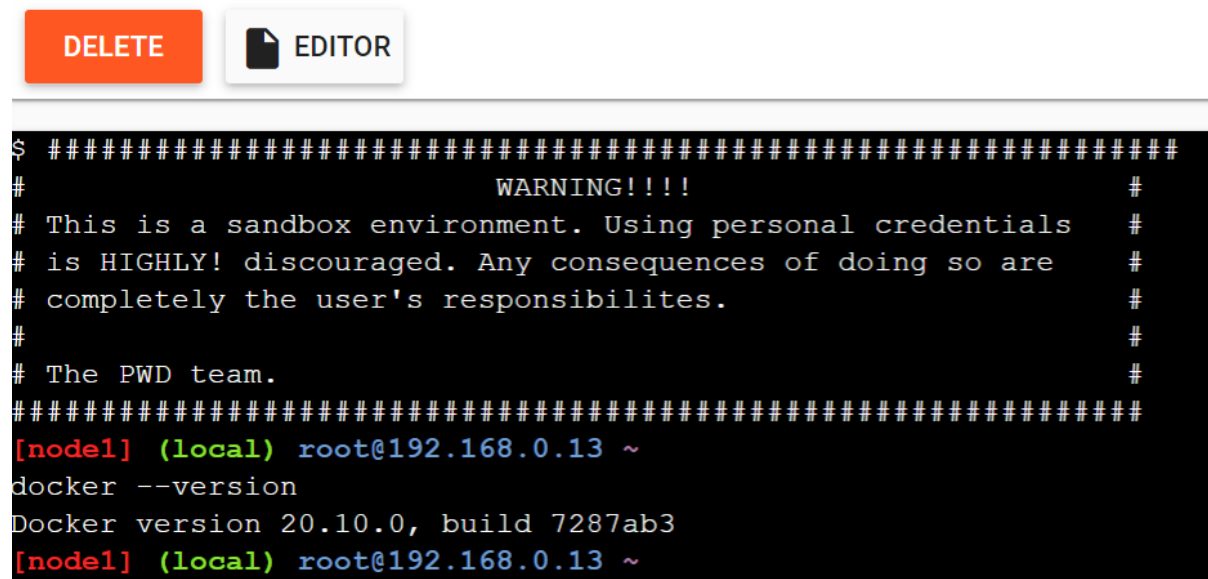
Method 1 :-

To pull and push images using docker

Command: to check docker version

docker --version

Output :-

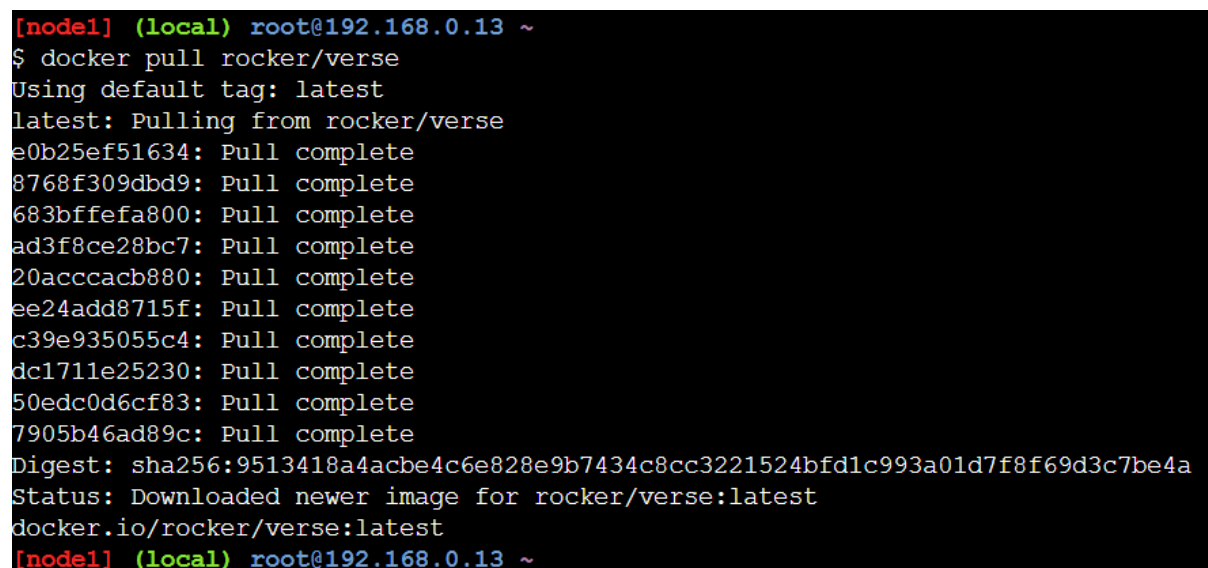


```
$ #####  
#                               WARNING!!!!                               #  
# This is a sandbox environment. Using personal credentials             #  
# is HIGHLY! discouraged. Any consequences of doing so are             #  
# completely the user's responsibilities.                                #  
#                                                                       #  
# The PWD team.                                                         #  
#####  
[node1] (local) root@192.168.0.13 ~  
docker --version  
Docker version 20.10.0, build 7287ab3  
[node1] (local) root@192.168.0.13 ~
```

Command: to pull readymade image

docker pull rocker/verse

Output :-



```
[node1] (local) root@192.168.0.13 ~  
$ docker pull rocker/verse  
Using default tag: latest  
latest: Pulling from rocker/verse  
e0b25ef51634: Pull complete  
8768f309dbd9: Pull complete  
683bffefa800: Pull complete  
ad3f8ce28bc7: Pull complete  
20acccacb880: Pull complete  
ee24add8715f: Pull complete  
c39e935055c4: Pull complete  
dc1711e25230: Pull complete  
50edc0d6cf83: Pull complete  
7905b46ad89c: Pull complete  
Digest: sha256:9513418a4acbe4c6e828e9b7434c8cc3221524bfd1c993a01d7f8f69d3c7be4a  
Status: Downloaded newer image for rocker/verse:latest  
docker.io/rocker/verse:latest  
[node1] (local) root@192.168.0.13 ~
```

Command: to check images in docker

docker images

Output :-

```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
rocker/verse        latest             c5aa21663dfd       3 days ago         4.11GB
[node1] (local) root@192.168.0.13 ~
```

Now Login to docker hub and create repository

Output :-

Repositories > Create

Create Repository

mazharsolkar protips

first repository

Visibility

Using 0 of 1 private repositories. [Get more](#)

☒ Public Appears in Docker Hub search results

☐ Private Only visible to you

[Cancel](#) [Create](#)

Pro tip

You can push a new image to this repository using the CLI

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change *tagname* with your desired image repository tag.

Click on Create button

Now check repository created

mazharsolkar Repositories > protips

Using 0 of 1 private repositories. [Get more](#)

[General](#) [Tags](#) [Builds](#) [Collaborators](#) [Webhooks](#) [Settings](#)

Advanced Image Management
View all your images and tags in this repository, clean up unused content, recover untagged images. Available with Pro, Team and Business subscriptions. [View preview](#)

mazharsolkar / protips

first repository

Last pushed: never

Docker commands [Public View](#)

To push a new tag to this repository,

```
docker push mazharsolkar/protips:tagname
```

Command: to login to your docker account

docker login --username=mazharsolkar

password:

note: mazharsolkar is my docker ID . You will use your docker ID here. And enter your password .

Output :-

```
[node1] (local) root@192.168.0.13 ~
$ docker login --username=mazharsolkar
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Command : to tag image

Docker tag c5aa21663dfd mazharsolkar/protips:firsttry

note: here c5aa21663dfd this is image id which you can get from docker images command. Protips is repository name and firsttry is tag name.

Output :-

```
$ docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
rocker/verse        latest         c5aa21663dfd    3 days ago      4.11GB
[node1] (local) root@192.168.0.13 ~
$ docker tag c5aa21663dfd mazharsolkar/protips:firsttry
[node1] (local) root@192.168.0.13 ~
```

Command: to push image to docker hub account

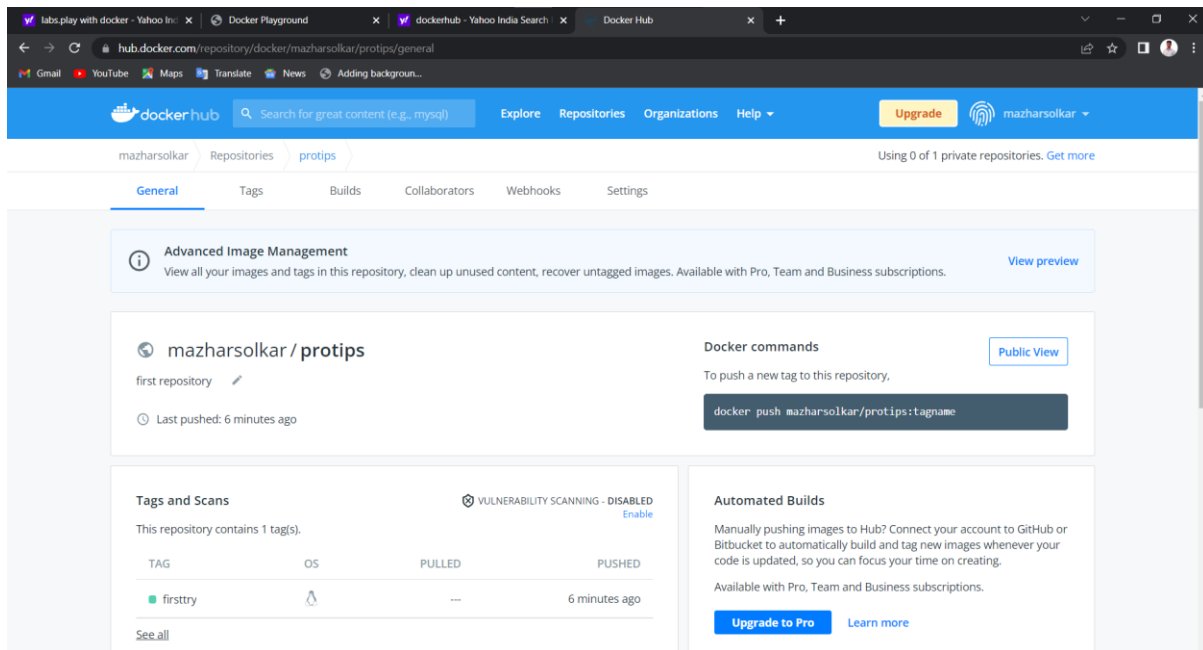
docker push mazharsolkar/protips:firsttry

note: firsttry is tag name created above.

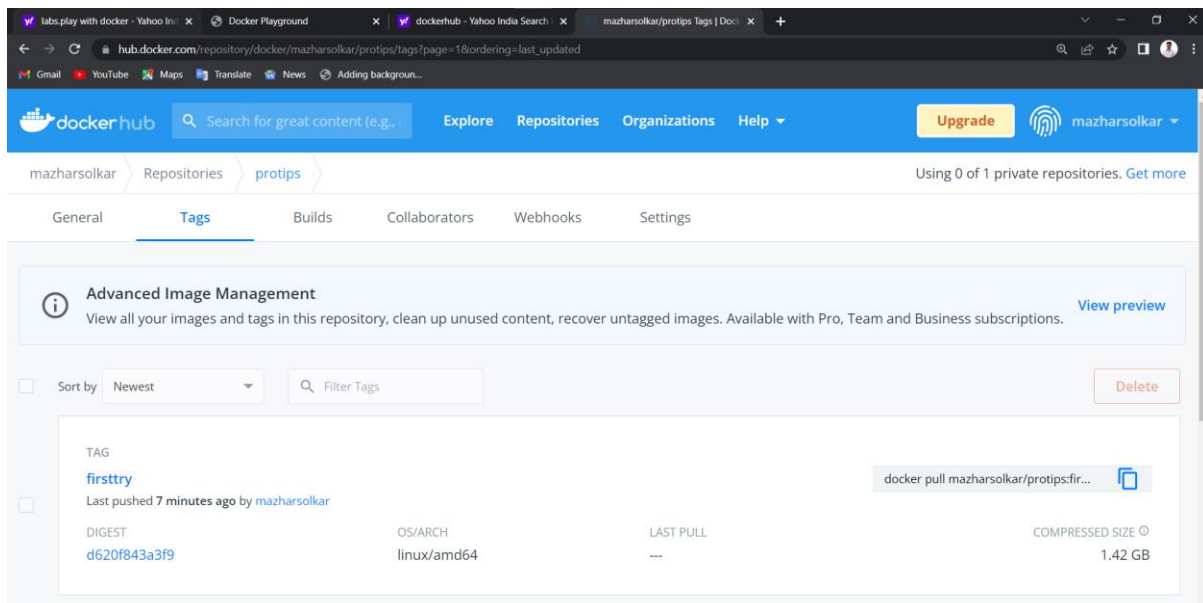
Output :-

```
$ docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
rocker/verse        latest         c5aa21663dfd    3 days ago      4.11GB
[node1] (local) root@192.168.0.13 ~
$ docker tag c5aa21663dfd mazharsolkar/protips:firsttry
[node1] (local) root@192.168.0.13 ~
$ ^C
[node1] (local) root@192.168.0.13 ~
$ docker push mazharsolkar/protips:firsttry
The push refers to repository [docker.io/mazharsolkar/protips]
9282aa176fef: Mounted from rocker/verse
bc1cf6db0c22: Mounted from rocker/verse
f0aeeb4f3116: Mounted from rocker/verse
314a576b4061: Mounted from rocker/verse
39bb2c9d4851: Mounted from rocker/verse
c248a6725dbd: Mounted from rocker/verse
d13413426caf: Mounted from rocker/verse
937357db62dd: Mounted from rocker/verse
f74791019906: Mounted from rocker/verse
c5ec52c98b31: Mounted from rocker/verse
firsttry: digest: sha256:d620f843a3f9c63f83830a06f046155da538f216bdc651ac1359a62d775dc9c5 size: 2431
[node1] (local) root@192.168.0.13 ~
```

Check it in docker hub now



Click on tags and check



Method 2 :-

Build an image then push it to docker and run it

Command : to create docker file

1. cat > Dockerfile <<EOF
2. FROM busybox
3. CMD echo "Hello world! This is my first Docker image."
4. EOF

Output :-

```
[node1] (local) root@192.168.0.13 ~
$ cat > Dockerfile << EOF
> From busybox
> CMD echo "Hello World! Thsis is my first docker image"
> EOF
[node1] (local) root@192.168.0.13 ~
$
```

Command : to build image from docker file

docker build -t mazharsolkar/repo2 .

Output :-

```
[node1] (local) root@192.168.0.13 ~
$ docker build -t mazharsolkar/repo2 .
Sending build context to Docker daemon   47MB
Step 1/2 : From busybox
latest: Pulling from library/busybox
50e8d59317eb: Pull complete
Digest: sha256:d2b53584f580310186df7a2055ce3ff83cc0df6caacf1e3489bff8cf5d0af5d8
Status: Downloaded newer image for busybox:latest
----> 1a80408de790
Step 2/2 : CMD echo "Hello World! Thsis is my first docker image"
----> Running in 3a00f37a97b1
Removing intermediate container 3a00f37a97b1
----> 4d2eb4c072cf
Successfully built 4d2eb4c072cf
Successfully tagged mazharsolkar/repo2:latest
[node1] (local) root@192.168.0.13 ~
$
```

Command: to check docker images

docker images

Output :-

```
[node1] (local) root@192.168.0.13 ~
$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
mazharsolkar/repo2   latest       4d2eb4c072cf      About a minute ago 1.24MB
busybox              latest       1a80408de790      3 days ago       1.24MB
rocker/verse         latest       c5aa21663dfd      3 days ago       4.11GB
mazharsolkar/protips firsttry     c5aa21663dfd      3 days ago       4.11GB
[node1] (local) root@192.168.0.13 ~
$
```

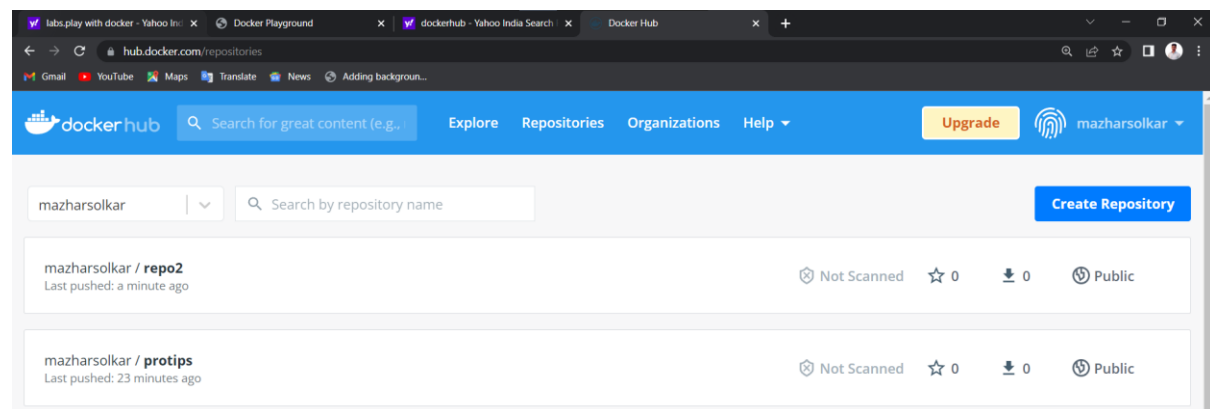
Command: to push image to docker hub

docker push mazharsolkar/repo2

Output :-

```
[node1] (local) root@192.168.0.13 ~
$ docker push mazharsolkar/repo2
Using default tag: latest
The push refers to repository [docker.io/mazharsolkar/repo2]
eb6b01329ebe: Mounted from library/busybox
latest: digest: sha256:10ef332cdd87ba7414e0a5872dbeb726ed3b1775bcf050c313e0d58cc3aac1ab size
: 527
[node1] (local) root@192.168.0.13 ~
$
```

Now check it on docker hub



command: to run docker image:

docker run mazharsolkar/repo2

Output :-

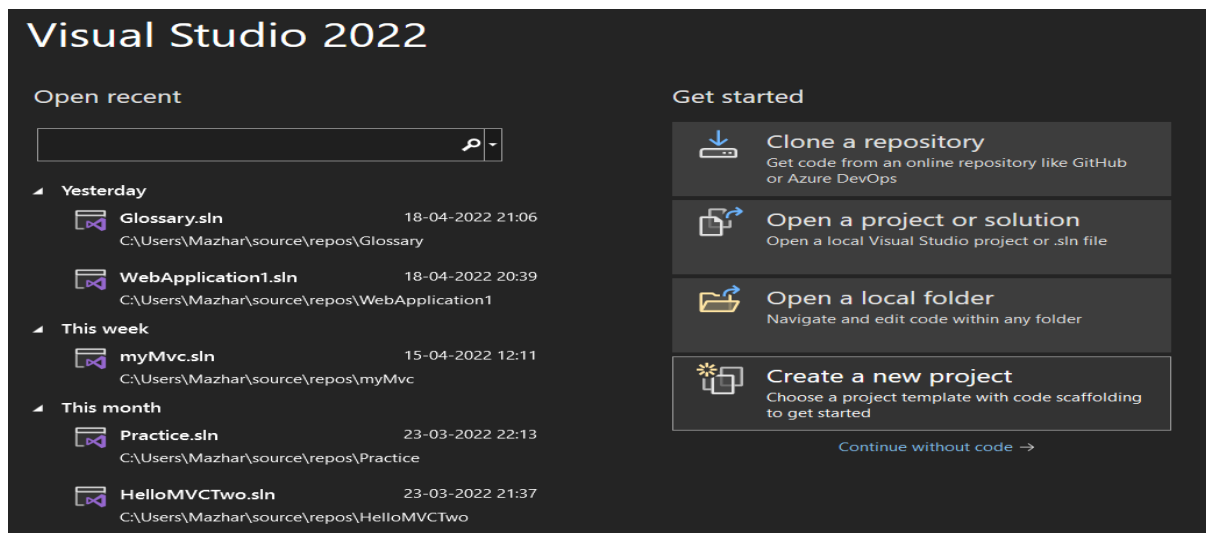
```
$ docker run mazharsolkar/repo2
Hello World! Thsis is my first docker image
[node1] (local) root@192.168.0.13 ~
$
```

Practical 4

Aim: To Create ASP.NET Core Application

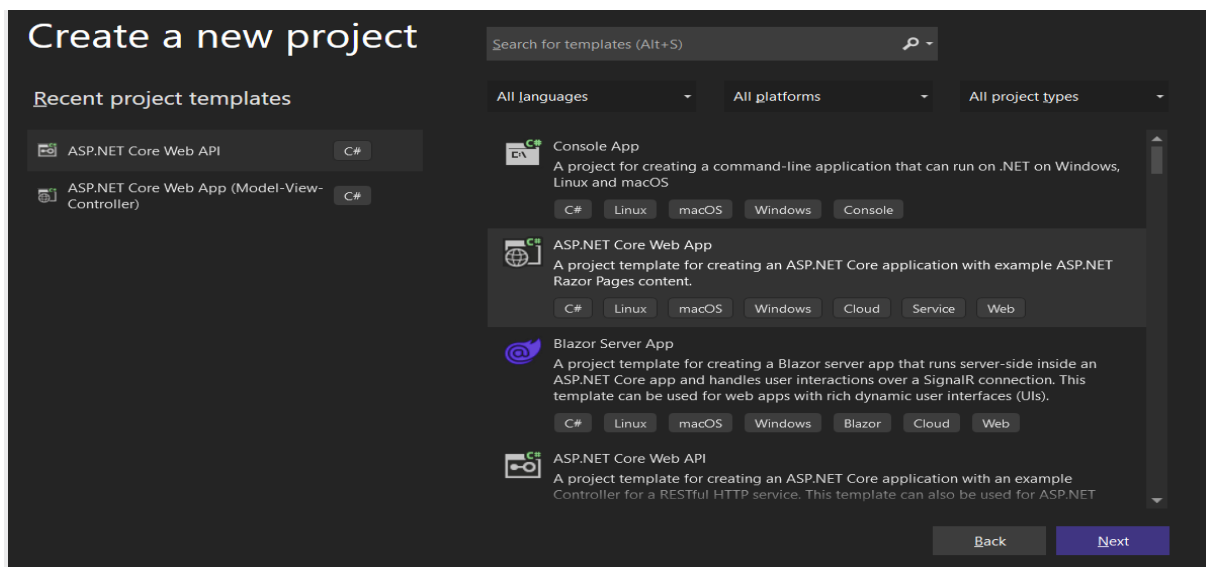
Here, we will learn how to create our first ASP.NET Core 3.0 application in Visual Studio 2022.

Open Visual Studio 2022 and click on **Create a new project**, as shown below.



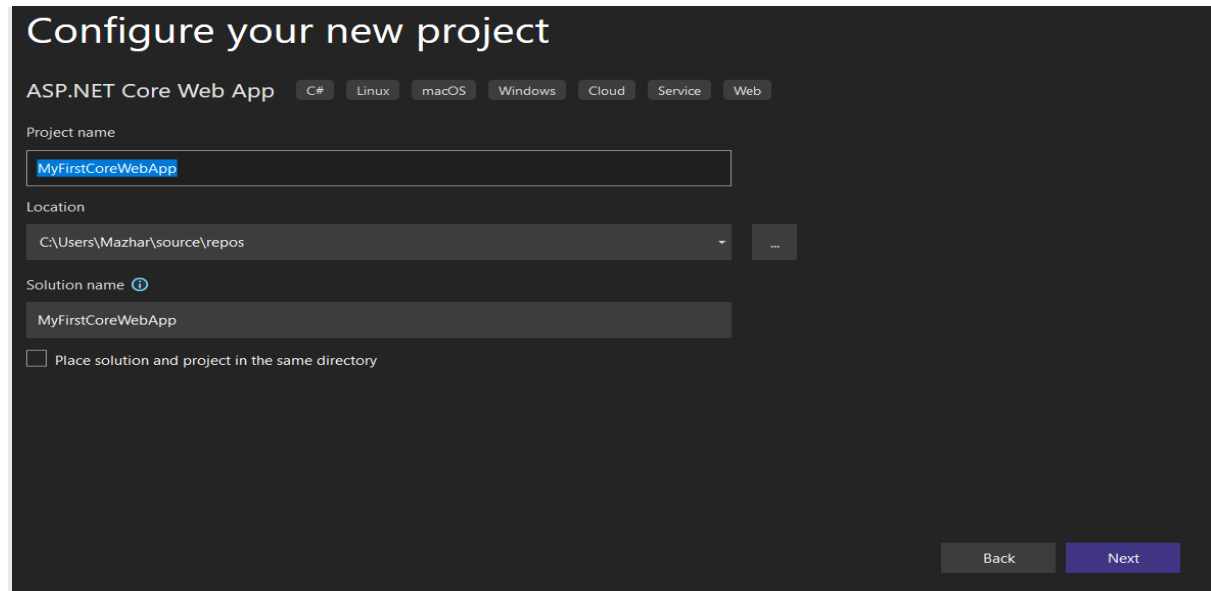
Create a New ASP.NET Core 3.0 Project

The "Create a new project" dialog box includes different .NET Core 3.0 application templates. Each will create predefined project files and folders depends on the application type. Here we will create a simple web application, so select **ASP.NET Core Web Application** template and click **Next**, as shown below.



Give a Name to your Project

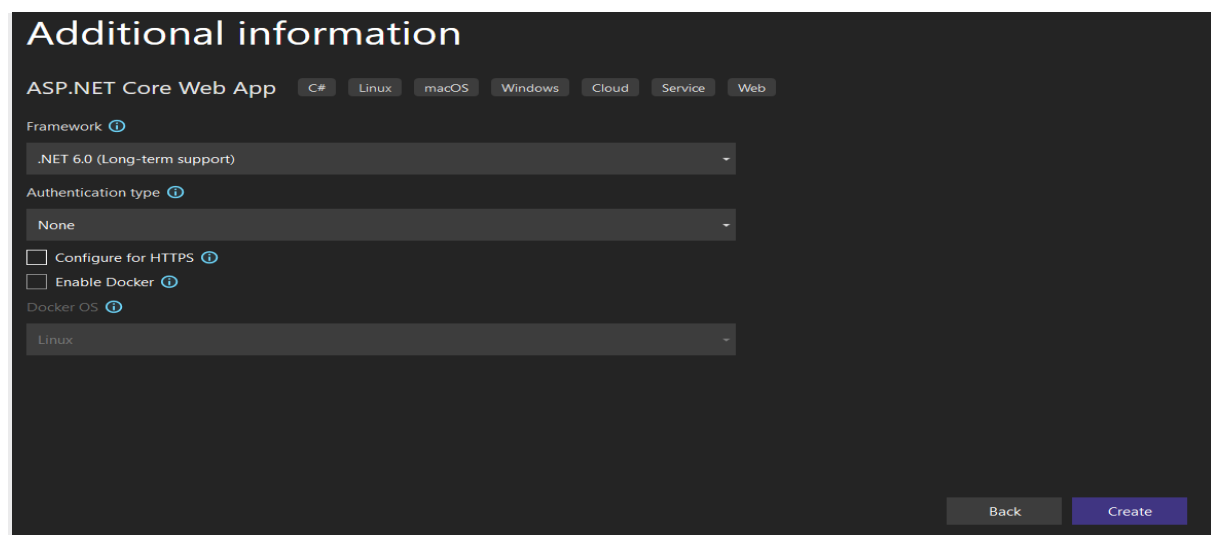
Give the appropriate name, location, and the solution name for the ASP.NET Core application. In this example, we will give the name "MyFirstCoreWebApp" and click on the **Next** button, as shown below.



The screenshot shows the 'Configure your new project' dialog box. At the top, it says 'ASP.NET Core Web App' followed by tabs for C#, Linux, macOS, Windows, Cloud, Service, and Web. The 'Project name' field contains 'MyFirstCoreWebApp'. The 'Location' field shows 'C:\Users\Mazhar\source\repos'. The 'Solution name' field contains 'MyFirstCoreWebApp'. There is an unchecked checkbox labeled 'Place solution and project in the same directory'. At the bottom right, there are 'Back' and 'Next' buttons.

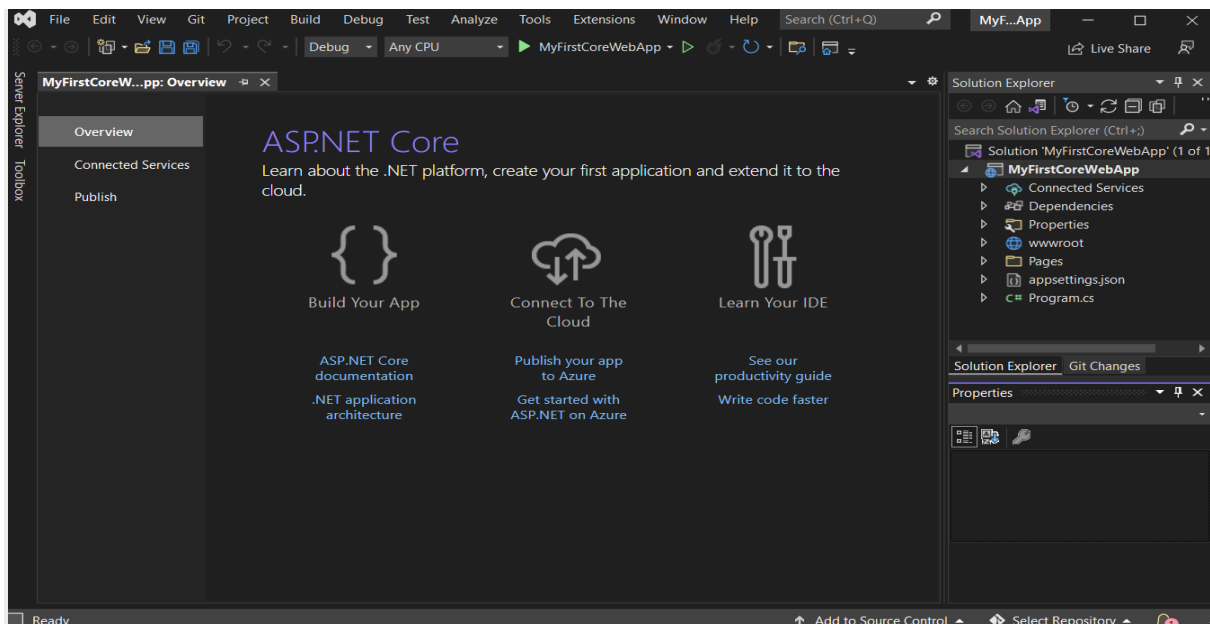
Configure Project

We don't want HTTPS at this point, so uncheck **Configure for HTTPS** checkbox, as shown below. Also, make sure you have selected the appropriate .NET Core and ASP.NET Core versions. Click on the **Create** button to create a project.



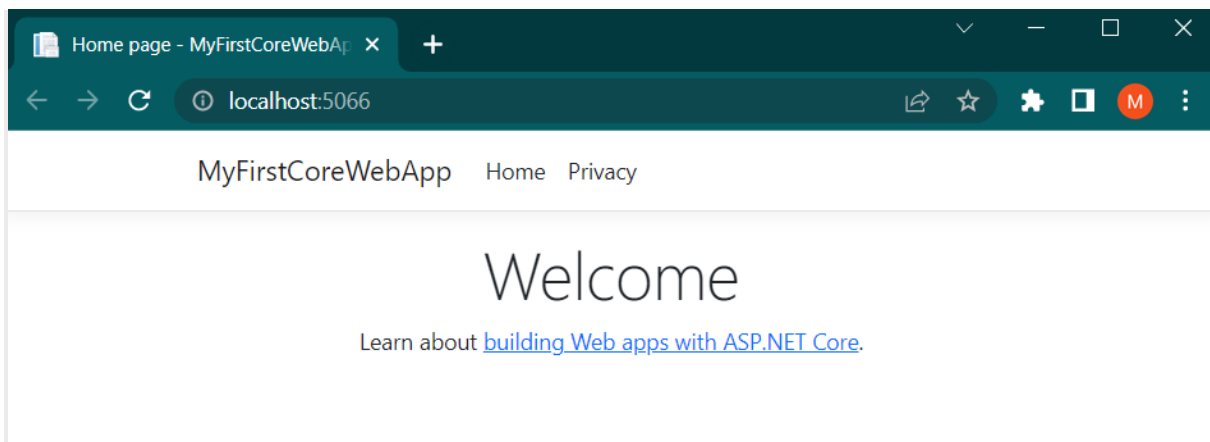
The screenshot shows the 'Additional information' dialog box. It has the same tabs as the previous dialog. The 'Framework' dropdown is set to '.NET 6.0 (Long-term support)'. The 'Authentication type' dropdown is set to 'None'. There are two unchecked checkboxes: 'Configure for HTTPS' and 'Enable Docker'. The 'Docker OS' dropdown is set to 'Linux'. At the bottom right, there are 'Back' and 'Create' buttons.

This will create a new ASP.NET Core web project in Visual Studio 2019, as shown below. Wait for some time till Visual Studio restores the packages in the project. Restoring process means Visual Studio will automatically add, update or delete configured dependencies as NuGet packages in the project.



Create ASP.NET Core 3 Application

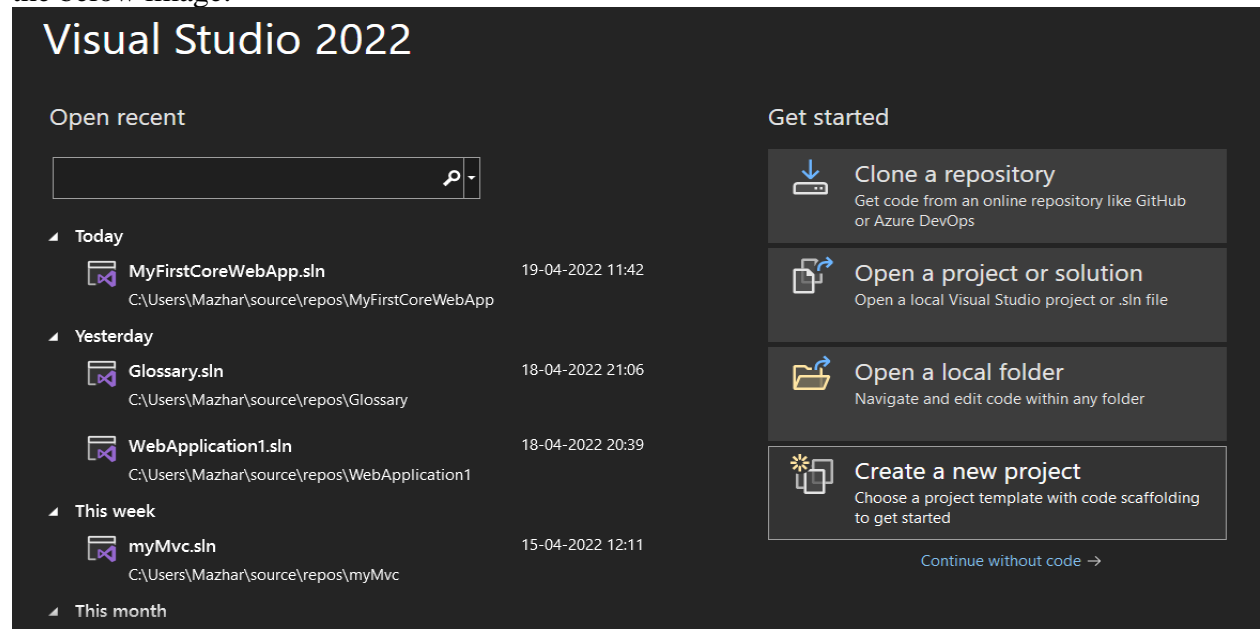
To run this web application, click on **IIS Express** or press Ctrl + F5. This will open the browser and display the following result.



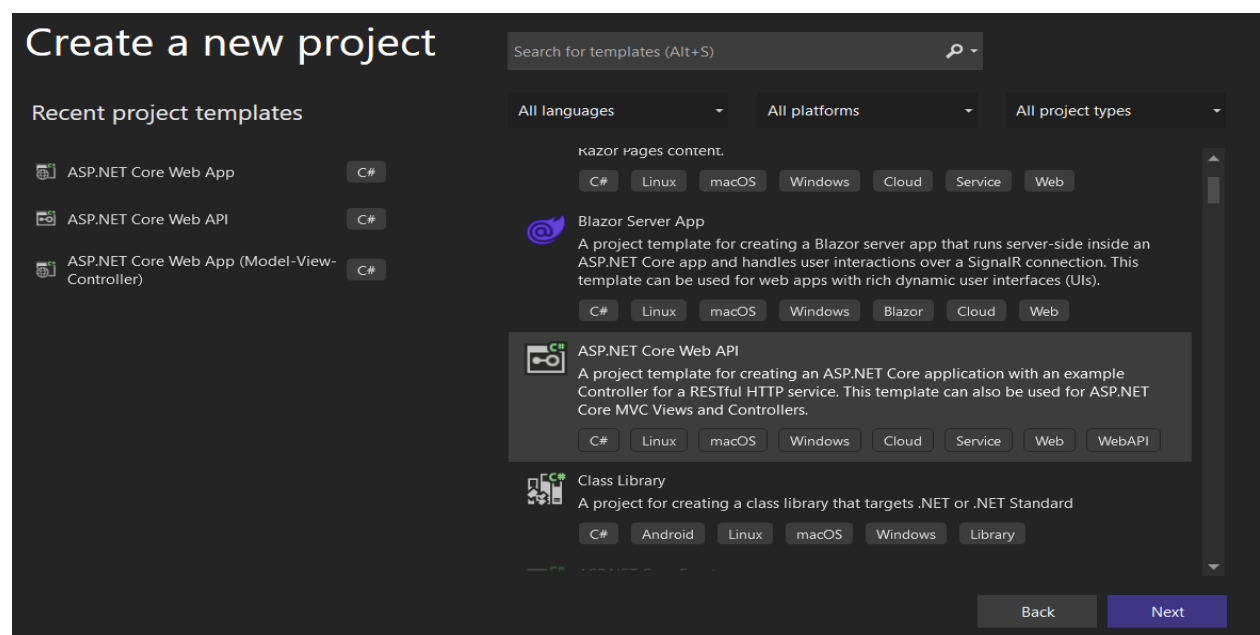
Practical 5

Aim: Creating ASP.NET Core Web API Project Using Visual Studio

Now, we are going to create the ASP.NET Core Web API project using Visual Studio 2022. So, open Visual Studio 2022 and then click on the Create a new project option as shown in the below image.



Once you click on the Create a new project option, the following Create a new project window will open. Here, you can find two projects template for creating the **ASP.NET Core Web API** project. One is using C# language and the other one is using F# language. I am going to use C# as the programming language, so I select the project template which uses C# Language as shown in the below image.



Once you click on the **Next** button, then the configure your new project window will open. Here, you need to specify the Project name Solution name, and the location where you want to create the project. I am providing the Project name as StudentAdmissionManagement, solution name as SchoolManagementSystem, and finally, click on the **Next** button as shown in the below image.

Configure your new project

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web WebAPI

Project name
StudentAdmissionManagement

Location
C:\Users\Mazhar\source\repos

Solution name ⓘ
SchoolManagementSystem

☐ Place solution and project in the same directory

Back Next

Once you click on the Next button, it will open the Additional Information window. Here, you need to select the Target .NET Framework version. The authentication Types. Whether you want to configure HTTPS and enable Docker. Select .NET Core 3.1 for now because it is having long Term support from Microsoft, select authentication type as None, check the Configure for HTTPS and uncheck the Enable Docker checkboxes and then click on the Create button as shown in the below image.

Additional information

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web WebAPI

Framework ⓘ
.NET 6.0 (Long-term support)

Authentication type ⓘ
None

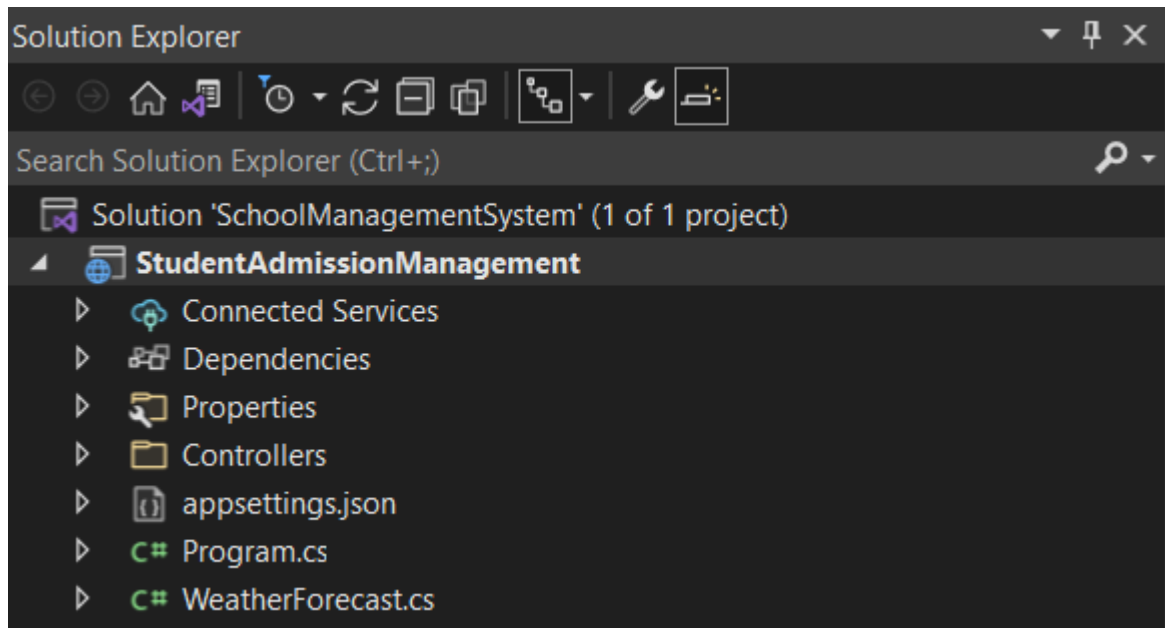
☒ Configure for HTTPS ⓘ
☐ Enable Docker ⓘ

Docker OS ⓘ
Linux

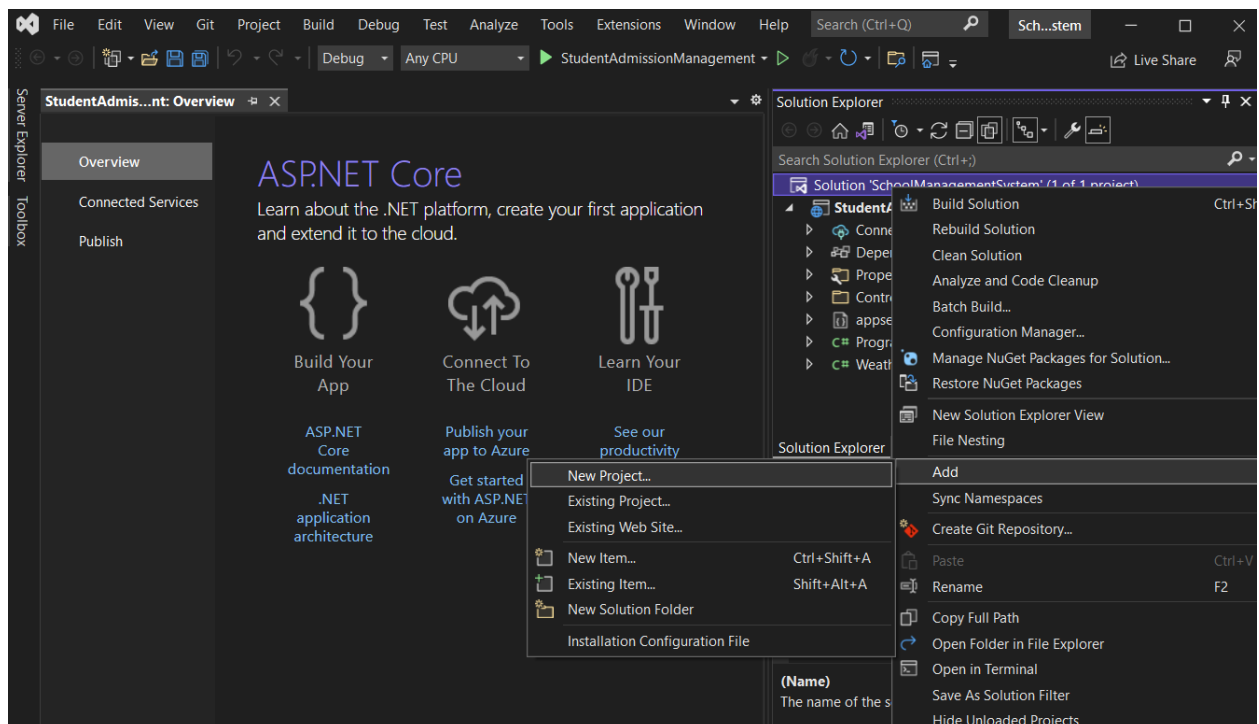
☒ Use controllers (unchecked to use minimal APIs) ⓘ
☒ Enable OpenAPI support ⓘ

Back Create

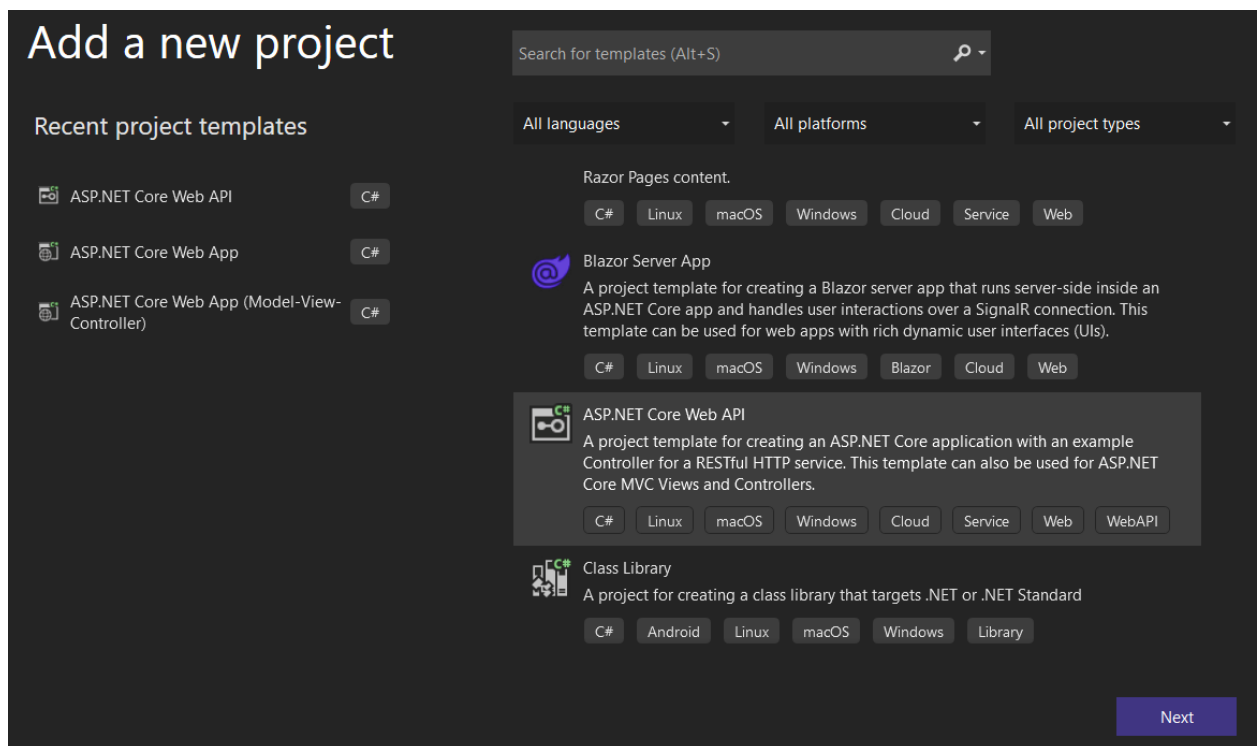
Once you click on the Create button, it will create the ASP.NET Core Web API project with the following file and folder structure. Initially, I have created only one separate project for student admission purposes which can be considered as a single microservice that works for student admission.



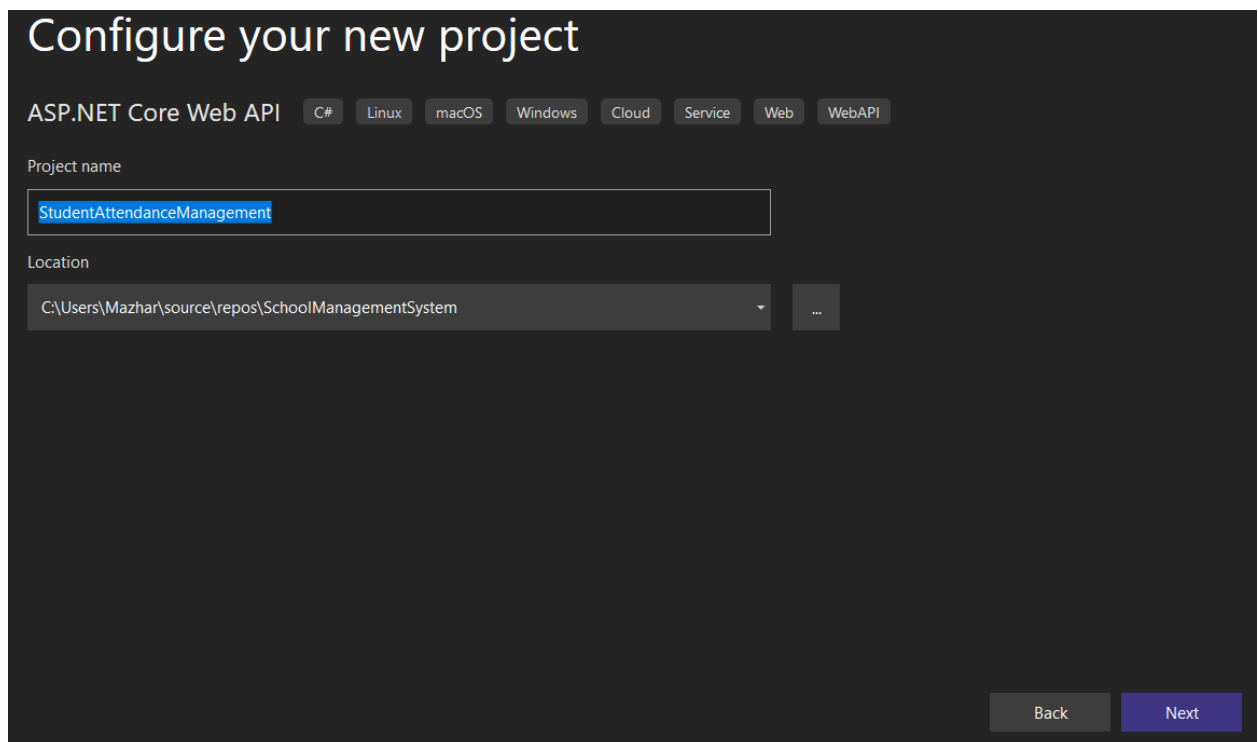
Let us create one more project under the solution which is another microservice for purpose of student attendance. To do so, right-click on the solution and then select Add => New Project option from the context menu as shown in the below image.



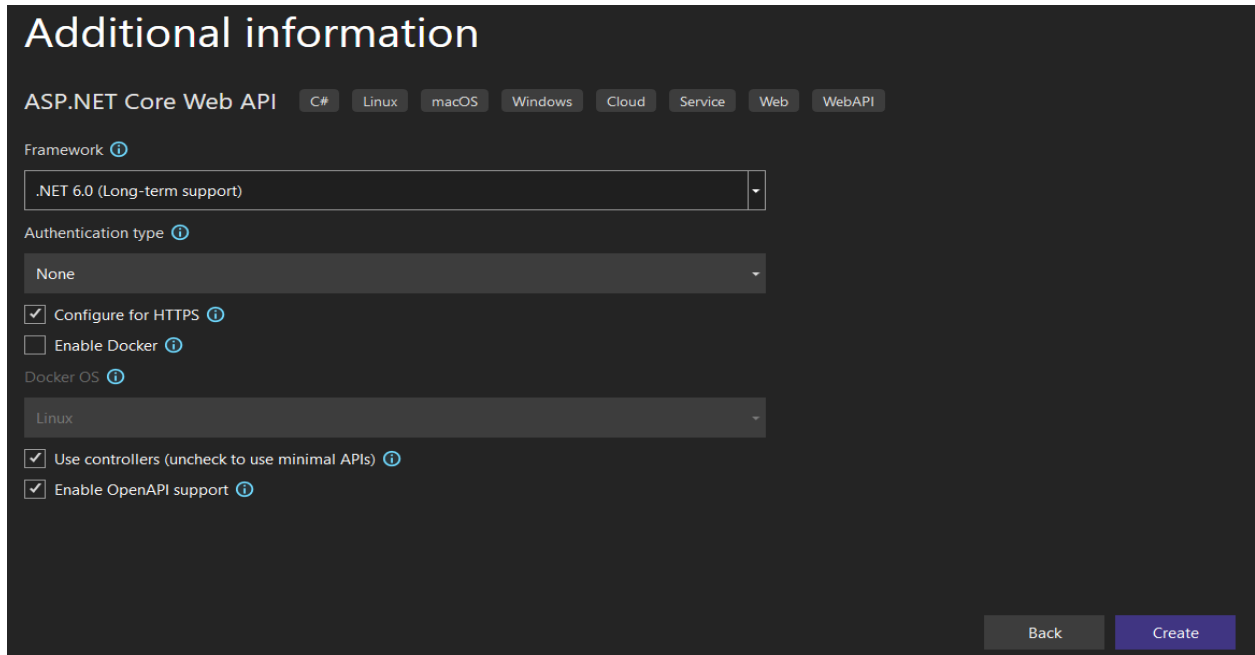
Once you click on the Add => New Project option, it will open the Add New Project window. From this window, select ASP.NET Core Web API (which uses C# language) and click on the Next button as shown in the below image.



Once you click on the Next button, it will open Configure your new project window. From this window, provide the project name as StudentAttendanceManagement and click on the Next button as shown in the below image.



Once you click on the Next button, it will open the Additional Information window. Here, Select .NET Core 3.1 as Target Framework, select authentication type as None, check the Configure for HTTPS and uncheck the Enable Docker checkboxes and then click on the Create button as shown in the below image.



Additional information

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web WebAPI

Framework ⓘ
.NET 6.0 (Long-term support)

Authentication type ⓘ
None

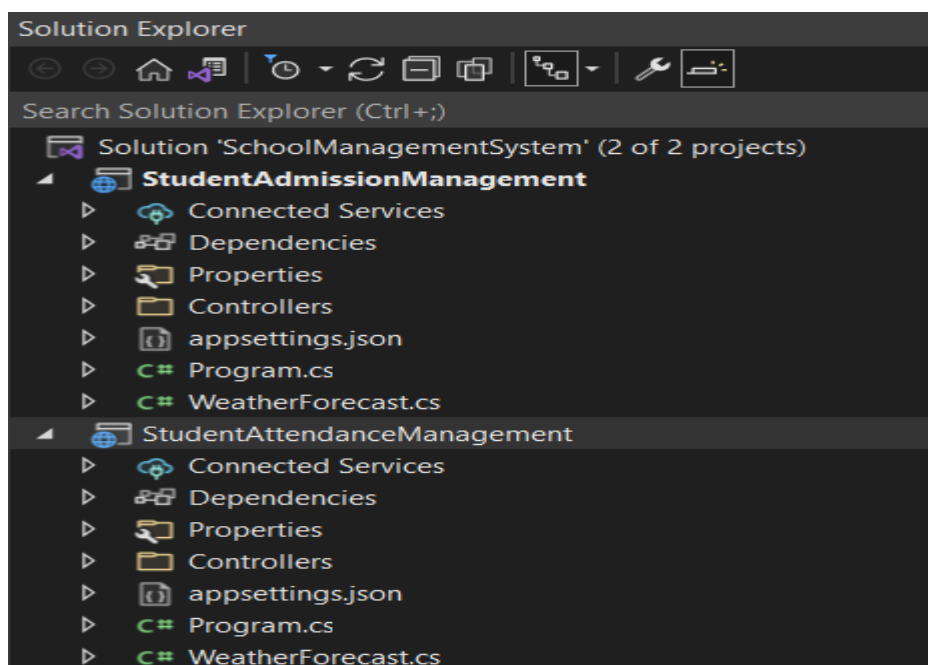
☒ Configure for HTTPS ⓘ
☐ Enable Docker ⓘ

Docker OS ⓘ
Linux

☒ Use controllers (unchecked to use minimal APIs) ⓘ
☒ Enable OpenAPI support ⓘ

Back Create

Once you click on the Create button, then it will add the new project to the existing solution. Now, our solution containing two projects with the following file and folder structure.



Now we have two microservices defined one for Student Admission purposes and the other for Student Attendance purposes.

Creating Models:

Now let's create a model class to hold details of Admission and Attendance in both the projects respectively and create a CRUD operation Controllers in each of project respectively under the Controllers folder. So, right-click on the StudentAdmissionManagement project and add a class file with the name StudentAdmissionDetailsModel.cs and then copy and paste the following code in it.

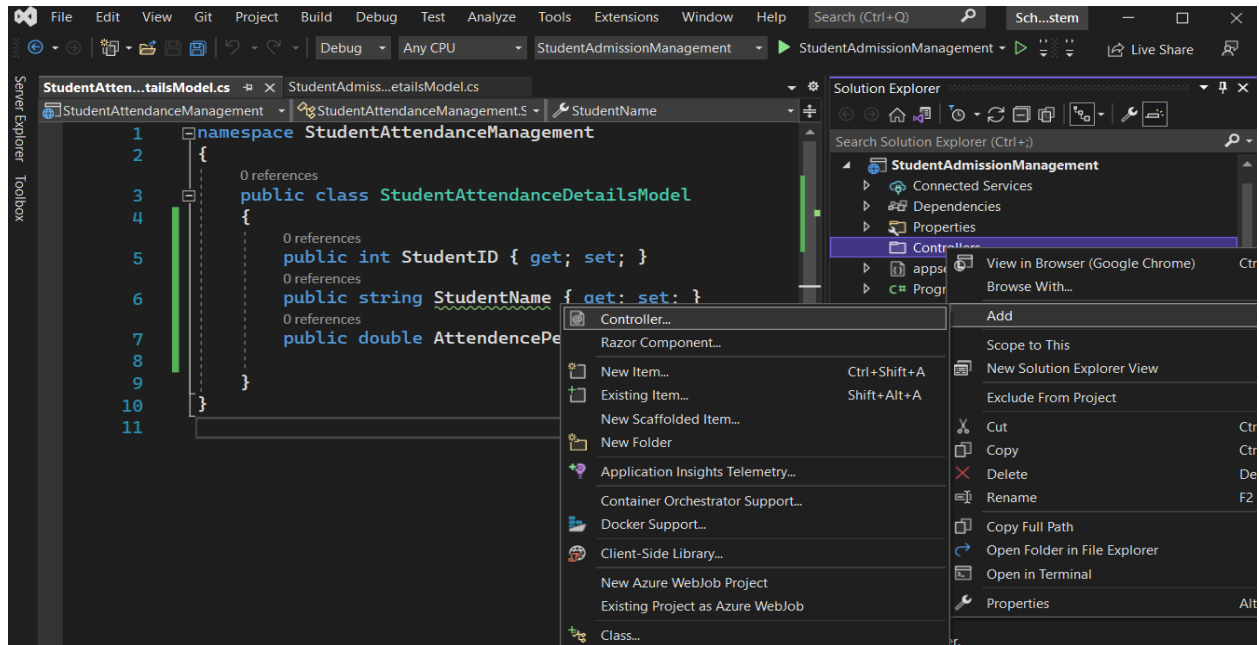
```
namespace StudentAdmissionManagement
{
    public class StudentAdmissionDetailsModel
    {
        public int StudentID { get; set; }
        public string? StudentName { get; set; }
        public string? StudentClass { get; set; }
        public DateTime DateofJoining { get; set; }
    }
}
```

Now, right-click on the StudentAttendanceManagement project and add a class file with the name StudentAttendanceDetailsModel.cs and then copy and paste the following code in it.

```
namespace StudentAttendanceManagement
{
    public class StudentAttendanceDetailsModel
    {
        public int StudentID { get; set; }
        public string? StudentName { get; set; }
        public double AttendancePercentage { get; set; }
    }
}
```

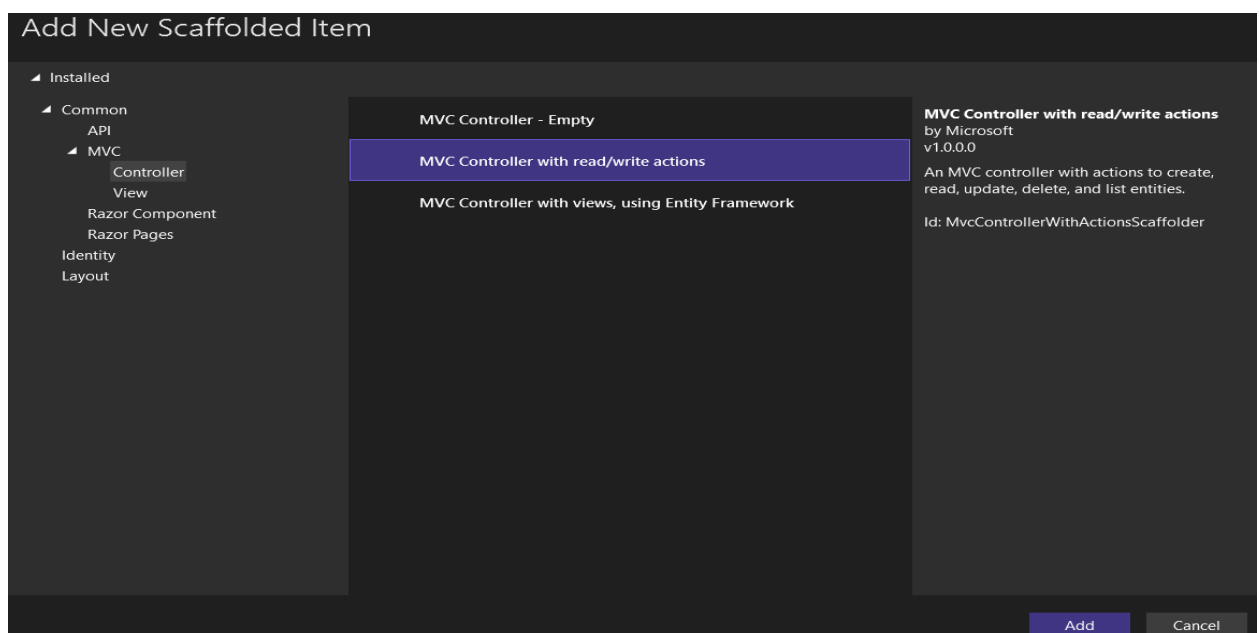
Creating Controllers:

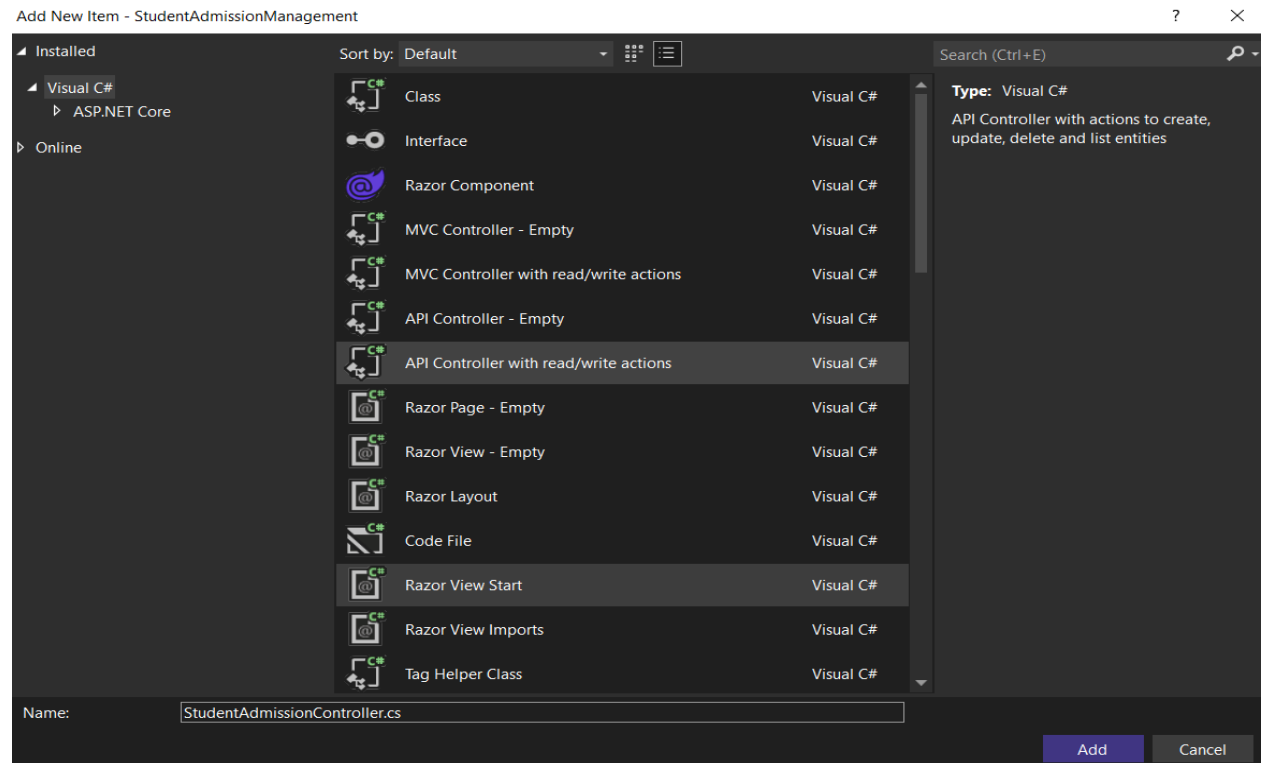
Now Let us create controllers which are endpoints for microservices for defining CRUD operations. Right-click on the Controllers folder of StudentAdmissionManagement and then select Add => Controller option from the context menu as shown in the below image.



Once you select the Add => Controller option, then it will open the Add New Scaffolded Item window. Here, first, select the API template and then select the Add Controller with read/write action template and click on the Add button as shown in the below image.

Once you click on the Add button, from the next window provide the name for your controller. Here, I am providing the name as StudentAdmissionController and click on the Add button as shown in the below image.





Once you click on the Add button, then it will add the StudentAdmissionController within the Controllers folder of your StudentAdmissionManagement project.

For the demonstration purpose, we create a GET endpoint with details of two students which will return details of two students to client request over HTTP GET. So, modify the StudentAdmissionController as follows.

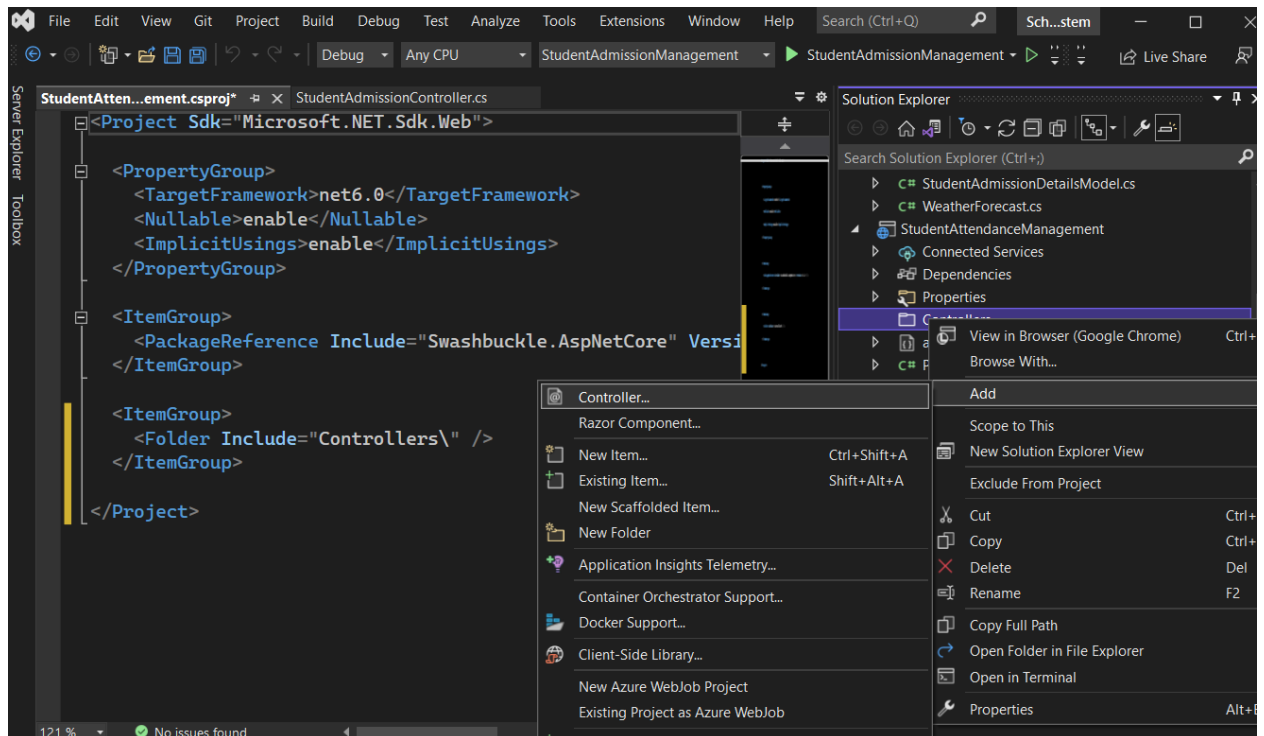
`using Microsoft.AspNetCore.Mvc;`

`// For more information on enabling Web API for empty projects, visit
https://go.microsoft.com/fwlink/?LinkID=397860`

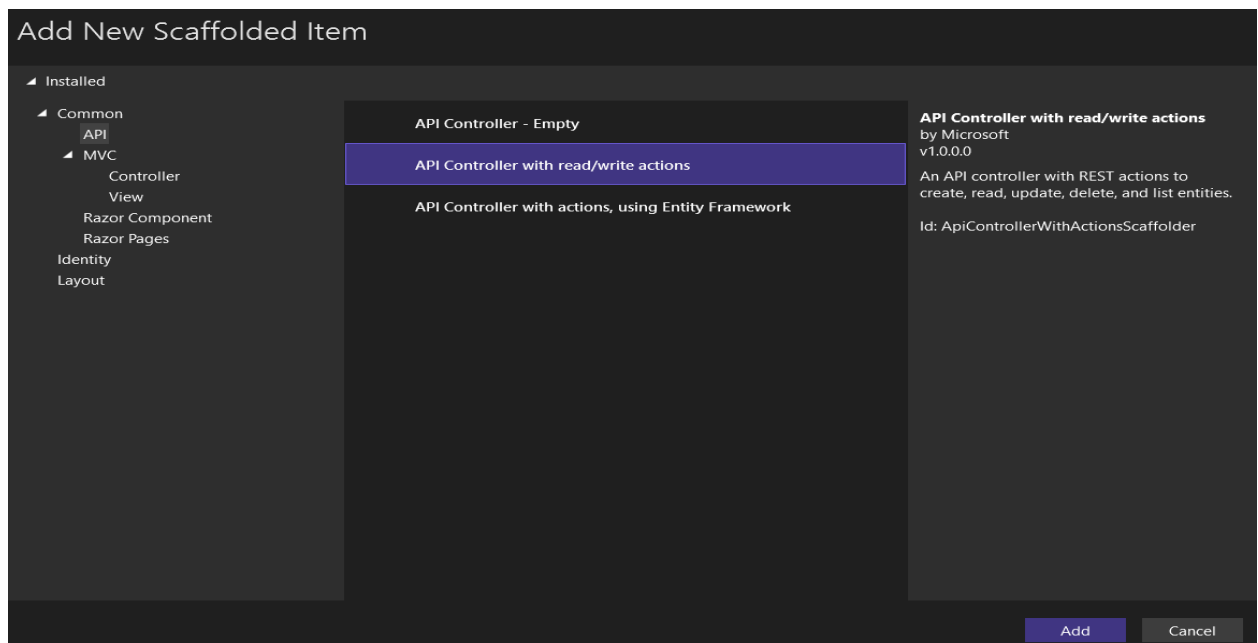
```
namespace StudentAdmissionManagement.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class StudentAdmissionController : ControllerBase
    {
        // GET: api/<StudentAdmissionController>
        [HttpGet]
        public IEnumerable<StudentAdmissionDetailsModel> Get()
        {
            StudentAdmissionDetailsModel admissionobj1 = new
StudentAdmissionDetailsModel();
            StudentAdmissionDetailsModel admissionobj2 = new
StudentAdmissionDetailsModel();
            admissionobj1.StudentID = 1;
            admissionobj1.StudentName = "Adam";
            admissionobj1.StudentClass = "X";
            admissionobj1.DateofJoining = DateTime.Now;
            admissionobj2.StudentID = 2;
            admissionobj2.StudentName = "Brad";
            admissionobj2.StudentClass = "IX";
            admissionobj2.DateofJoining = DateTime.Now;
            List<StudentAdmissionDetailsModel> listofobj = new
List<StudentAdmissionDetailsModel>
{
    admissionobj1,
    admissionobj2
};
            return listofobj;

        }
    }
}
```

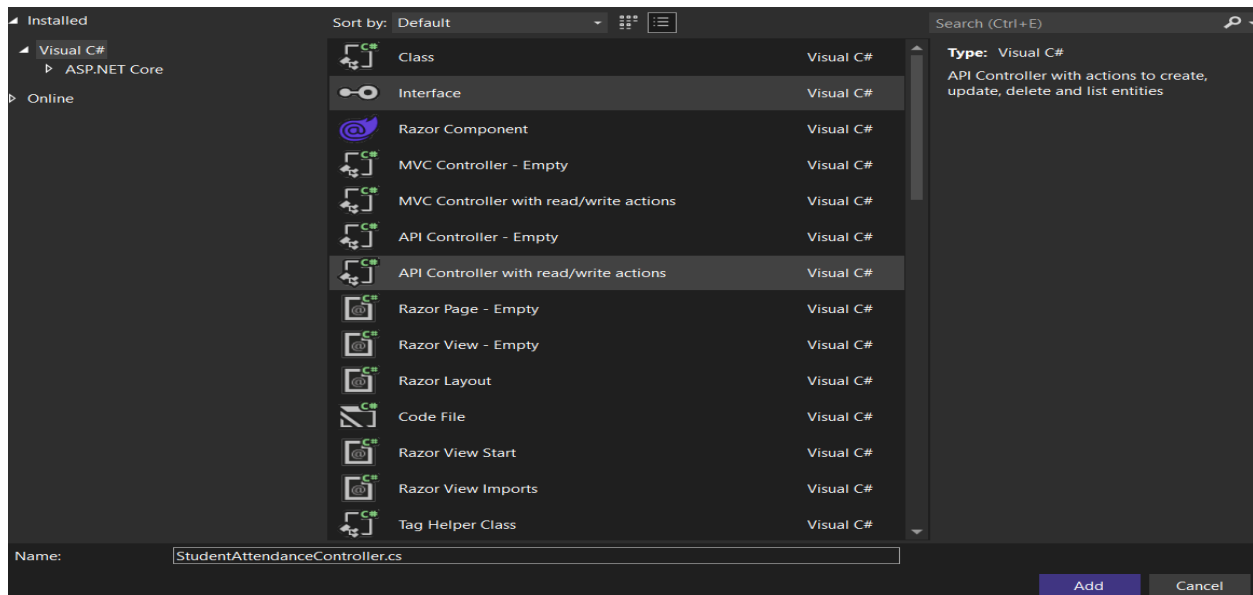
Now, Right-click on the Controllers folder of StudentAttendanceManagement and then select Add => Controller option from the context menu as shown in the below image.



Once you select the Add => Controller option, then it will open the Add New Scaffolded Item window. Here, first, select the API template and then select the Add Controller with read/write actions and click on the Add button as shown in the below image.



Once you click on the Add button, from the next window provide the name for your controller. Here, I am providing the name as StudentAttendanceController and click on the Add button as shown in the below image.

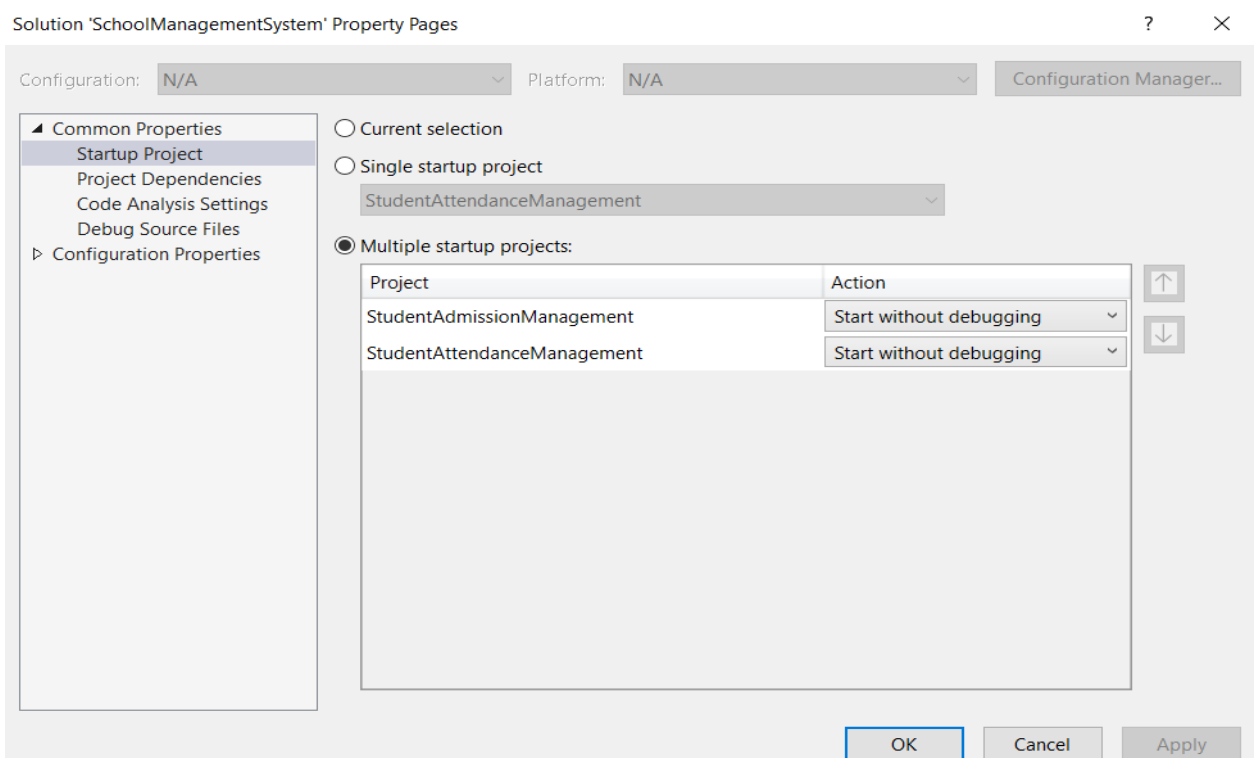
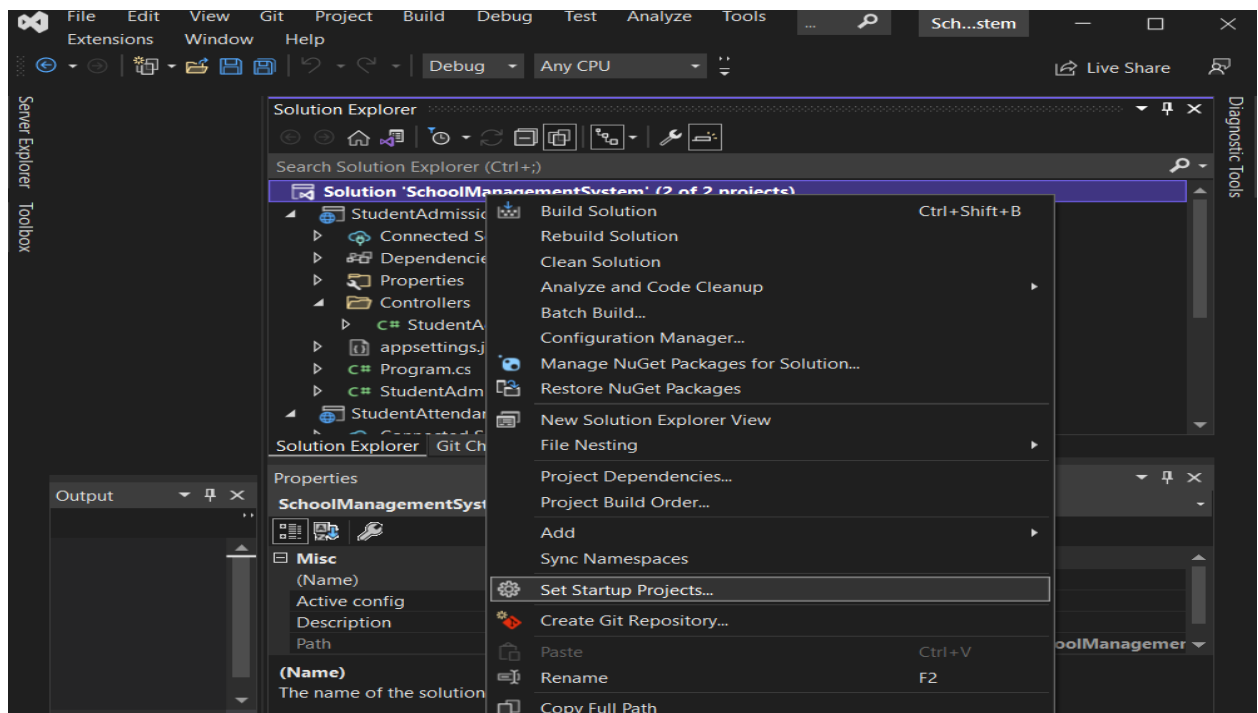


Once you click on the Add button, then it will add the StudentAttendanceController within the Controllers folder of your StudentAttendanceManagement project. For the demonstration purpose, we create a simple Get method for returning the two students' attendance percentage on HTTP GET requests. So, modify the StudentAttendanceController as follows.

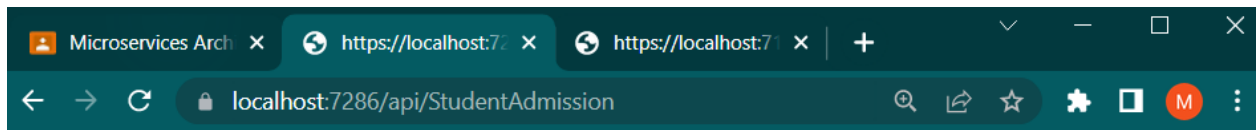
```
using Microsoft.AspNetCore.Mvc;
namespace StudentAttendanceManagement.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class StudentAttendanceController : ControllerBase
    {
        // GET: api/<StudentAttendanceController>
        [HttpGet]
        public IEnumerable<StudentAttendanceController> Get()
        {
            StudentAttendanceDetailsModel attendanceObj1 = new
StudentAttendanceDetailsModel();
            StudentAttendanceDetailsModel attendanceObj2 = new
StudentAttendanceDetailsModel();
            attendanceObj1.StudentID = 1;
            attendanceObj1.StudentName = "Adam";
            attendanceObj1.AttendancePercentage = 83.02;
            attendanceObj2.StudentID = 2;
            attendanceObj2.StudentName = "Brad";
            attendanceObj2.AttendancePercentage = 71.02;
            List<StudentAttendanceDetailsModel> listObj = new
List<StudentAttendanceDetailsModel>
{
    attendanceObj1,
    attendanceObj2
};
            return listObj; } }
```

Let's build the solution and launch both the Web API services (Microservices) simultaneously

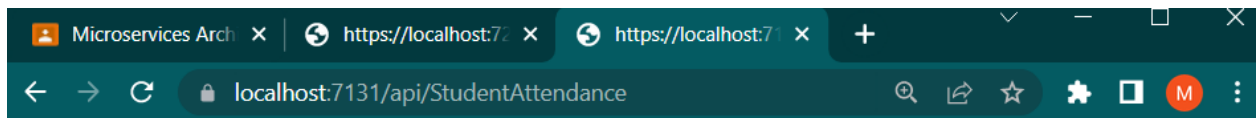
Running Applications and Showing Outputs



Click on the Start button and then we get two browsers opened and point to the controllers from browsers like below. Browser by default sends GET HTTP Verb.



```
[{"studentID":1,"studentName":"Mazhar","studentClass":"X","dateofJoining":"2022-04-20T18:02:47.8642361+05:30"}, {"studentID":2,"studentName":"Brad","studentClass":"IX","dateofJoining":"2022-04-20T18:02:47.8642973+05:30"}]
```



```
[{"studentID":1,"studentName":"Mazhar","attendancePercentage":83.02}, {"studentID":2,"studentName":"Brad","attendancePercentage":71.02}]
```

They returned the details as expected. So, this is how microservices are developed and our two microservices are available to serve the user requests over HTTP request and response. To test Microservices with Postman application please refer to the Testing section provided at end of this learning path.

Till now we have observed the implementation of independent APIs (Microservices) which hold separate concerns of business functionalities, one is to get admission details of all the students and the other is to get the attendance percentage of all the students in examples. Which are directly exposed to the Client.

Practical 6

Aim: Web API versioning using URI.

Why Web API versioning is required?

Once you develop and deploy a Web API service then different clients start consuming your Web API services. As you know, day by day the business grows and once the business grows then the requirement may change, and once the requirement change then you may need to change the services as well, but the important thing you need to keep in mind is that you need to do the changes to the services in such a way that it should not break any existing client applications who already consuming your services. This is the ideal scenario when the Web API versioning plays an important role. You need to keep the existing services as it is so that the existing client applications will not break, they worked as it is, and you need to develop a new version of the Web API service which will start consuming by the new client applications.

What are the Different options available in Web API to maintain the versioning?

The different options that are available to maintain versioning are as follows

1. URI's

URI	Return Value
/api/v1/employees	List of all Employees (Version 1)
/api/v1/employees/102	Specific Employee (Version 1)

2. Query String

URI	Return Value
/api/employees?v=1	List of all Employees (Version 1)
/api/employees?v=2	List of all Employees (Version 1)
/api/employees/102?v=1	Specific Employee (Version 1)
/api/employees/102?v=2	Specific Employee (Version 2)
/api/employees	List of all Employees (Version 1) by Default
/api/employees/102	Specific Employee (Version 1) by Default

3. Version Header

The screenshot shows a REST client interface. At the top, there is a dropdown menu set to 'GET' and a text input field containing the URL 'http://localhost:52418/api/employees'. To the right of the URL is a 'Params' tab and a blue 'Send' button. Below the URL bar, there are several tabs: 'Authorization', 'Headers (1)', 'Body', 'Pre-request Script', and 'Tests'. The 'Headers (1)' tab is selected and highlighted with a red box. Below the tabs, there is a table with three columns: 'Key', 'Value', and 'Description'. The first row in the table has a checked checkbox in the 'Key' column, the text 'X-EmployeesService-Version' in the 'Key' column, the value '2' in the 'Value' column (which is also highlighted with a red box), and 'Version 2' in the 'Description' column.

Key	Value	Description
<input checked="" type="checkbox"/> X-EmployeesService-Version	2	Version 2

4. Accept Header

5. Media Type

How to version a Web API using URI's?

I am going to discuss how to maintain versioning using URI's?

ASP.NET Web API application with the name **WebAPIVersioning**

Once you create the application, let's create the following **EmployeeV1** model within the Models folder

EmployeeV1.cs

```
namespace WebAPIVersioning.Models
{
    public class EmployeeV1
    {
        public int EmployeeID { get; set; }
        public string EmployeeName { get; set; }
    }
}
```

Version 1 of the above Employee class (EmployeeV1) has just 2 **properties (EmployeeID & EmployeeName)**.

Let's create an empty Web API controller with the name **EmployeesV1Controller** within the Controllers folder which will act as our version 1 controller. Once you create the controller, then please copy and paste the following code in it.

```
using System.Web.Http;
using WebAPIVersioning.Models;
using System.Collections.Generic;
using System.Linq;
namespace WebAPIVersioning.Controllers
{
    public class EmployeesV1Controller : ApiController
    {
        List<EmployeeV1> employees = new List<EmployeeV1>()
```



```
{  
new EmployeeV1() { EmployeeID = 101, EmployeeName = "Anurag"},  
new EmployeeV1() { EmployeeID = 102, EmployeeName = "Priyanka"},  
new EmployeeV1() { EmployeeID = 103, EmployeeName = "Sambit"},  
new EmployeeV1() { EmployeeID = 104, EmployeeName = "Preety"},  
};  
  
public IEnumerable<EmployeeV1> Get()  
{  
return employees;  
}  
  
public EmployeeV1 Get(int id)  
{  
return employees.FirstOrDefault(s => s.EmployeeID == id);  
}  
}  
}
```

Finally, modify the WebApiConfig.cs file as shown below.

```
using System.Web.Http;  
  
namespace WebAPIVersioning  
{  
public static class WebApiConfig  
{  
public static void Register(HttpConfiguration config)  
{  

```

```
config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(
    name: "Version1",
    routeTemplate: "api/v1/employees/{id}",
    defaults: new { id = RouteParameter.Optional, controller = "EmployeesV1" }
);
}
```

The clients of our Version1 Web API service can use the following URLs to get either the list of all employees or a specific employee by using the EmployeeID. At the moment, as part of the employee objects, the service returns the Employee Id and Name properties.

URI	Return Value
/api/v1/employees	List of all Employees (Version 1)
/api/v1/employees/102	Specific Employee (Version 1)

Implementing Web API Versioning using URI

Let's say the business grows and as a result, the requirements have changed and now some of the new clients want the FirstName and LastName properties instead of the Name property. If we change the Version 1 Web API service, then it will break all the existing client applications. So there is a need to create Version 2 of the Web API service which will be consumed by the new clients who want the FirstName and LastName properties instead of the Name property. If we do so, I mean if we create Version 2 of the Web API Service, then all the existing client applications will not break, they work as it is as before and now they have 2 options. If they want they can make use of the version 2 Web API service by making changes to their application or else they still continue their work as it is without changing their application. So, the important point to keep in mind is that with Web API Versioning we are not breaking any existing client application and at the same time we are also satisfying the new client requirements.

Following are the steps to create Version 2 of the Web API Service

Step1: Add a class file within the Models folder with the name it EmployeeV2 and then copy and paste the following code.

```
namespace WebAPIVersioning.Models
{
    public class EmployeeV2
    {
        public int EmployeeID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
    }
}
```

Notice in **EmployeeV2** class, instead of the Name property, we have the **FirstName** and **LastName** properties.

Step2: Add a new Web API 2 empty controller within the Controllers folder with the name “**EmployeesV2Controller**” and then copy and paste the following code.

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using WebAPIVersioning.Models;
namespace WebAPIVersioning.Controllers
{
    public class EmployeesV2Controller : ApiController
    {
        List<EmployeeV2> employees = new List<EmployeeV2>()
        {
            new EmployeeV2() { EmployeeID = 101, FirstName = "Anurag", LastName = "Mohanty"},
        }
    }
}
```

```
new EmployeeV2() { EmployeeID = 102, FirstName = "Priyanka", LastName =  
"Dewangan"},  
new EmployeeV2() { EmployeeID = 103, FirstName = "Sambit", LastName = "Satapathy"},  
new EmployeeV2() { EmployeeID = 104, FirstName = "Preety", LastName = "Tiwary"},  
};  
  
public IEnumerable<EmployeeV2> Get()  
{  
    return employees;  
}  
  
public EmployeeV2 Get(int id)  
{  
    return employees.FirstOrDefault(s => s.EmployeeID == id);  
}  
  
}  
  
}
```

Now, the “**EmployeesV2Controller**” returns the “**EmployV2**” object that has the **FirstName** and **LastName** properties instead of the **Name** property.

Step3: Modify the **WebApiConfig.cs** file as shown below.

```
using System.Web.Http;  
  
namespace WebAPIVersioning  
{  
    public static class WebApiConfig  
    {  
        public static void Register(HttpConfiguration config)
```

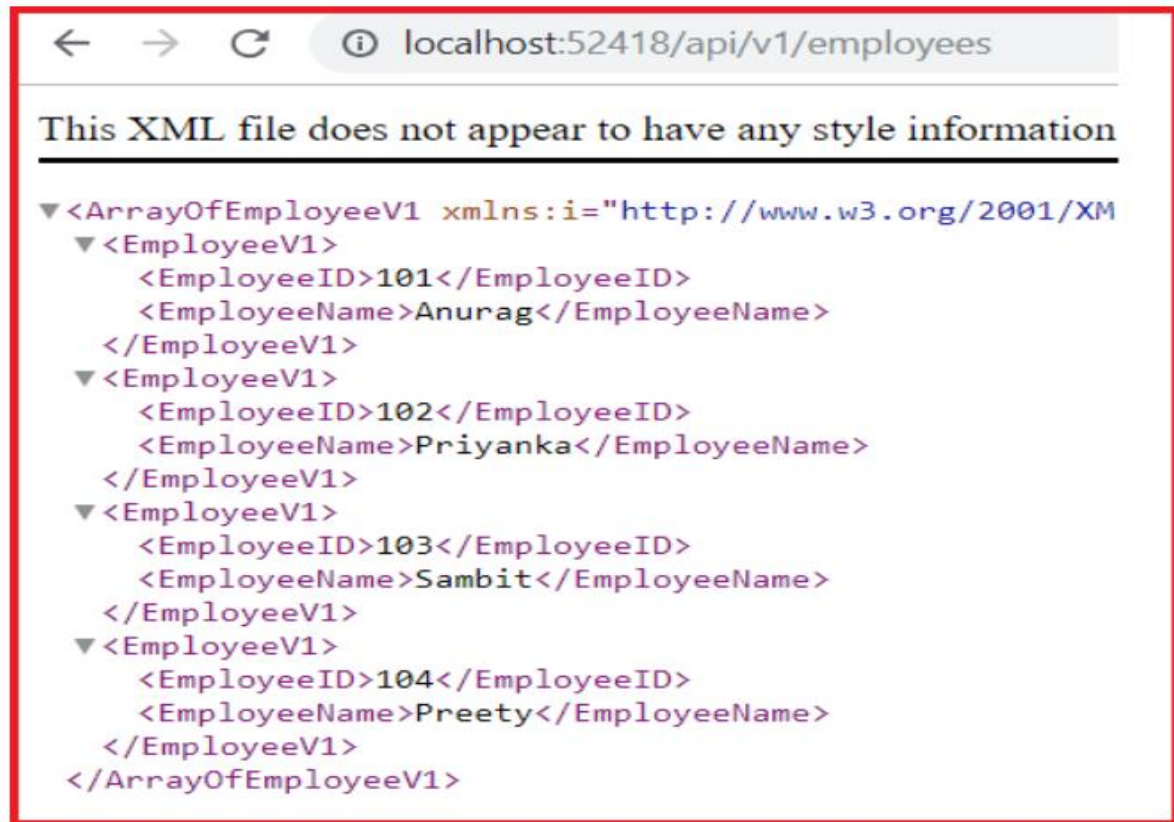
```
{  
    config.MapHttpAttributeRoutes();  
    config.Routes.MapHttpRoute(  
        name: "Version1",  
        routeTemplate: "api/v1/employees/{id}",  
        defaults: new { id = RouteParameter.Optional, controller = "EmployeesV1" }  
    );  
    config.Routes.MapHttpRoute(  
        name: "Version2",  
        routeTemplate: "api/v2/employees/{id}",  
        defaults: new { id = RouteParameter.Optional, controller = "EmployeesV2" }  
    );  
}  
}
```

As shown in the above **WebApiConfig.cs** file, now we have 2 routes. Notice the route template, for each of the routes. The

Version 1 clients use “**/api/v1/employees/{id}**” route while the

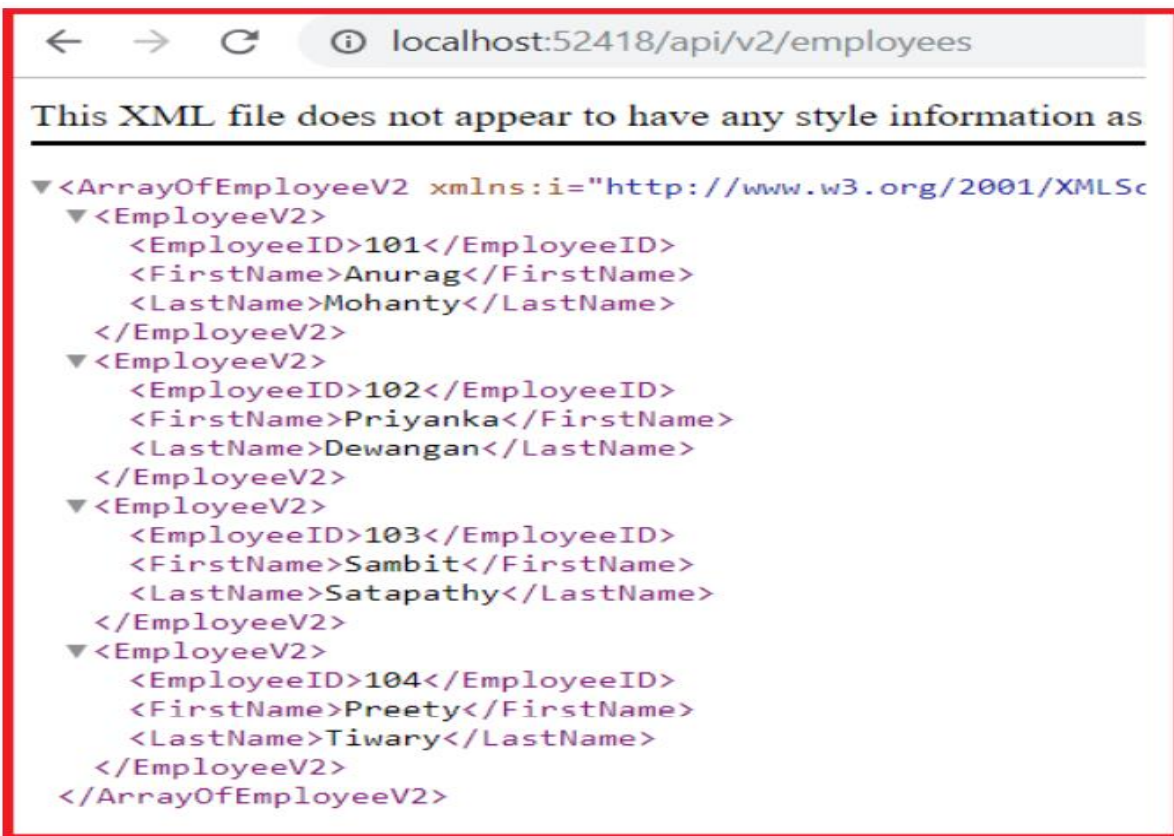
Version 2 clients use “**/api/v2/employees/{id}**” route That’s it.

We have implemented the versioning in our Web API service. So, if we navigate to **/api/v1/employees** – We get the employee with Employee Id and Name properties as shown below



```
<?xml version='1.0'>
<ArrayOfEmployeeV1 xmlns:i='http://www.w3.org/2001/XMLSchema-instance'>
  <EmployeeV1>
    <EmployeeID>101</EmployeeID>
    <EmployeeName>Anurag</EmployeeName>
  </EmployeeV1>
  <EmployeeV1>
    <EmployeeID>102</EmployeeID>
    <EmployeeName>Priyanka</EmployeeName>
  </EmployeeV1>
  <EmployeeV1>
    <EmployeeID>103</EmployeeID>
    <EmployeeName>Sambit</EmployeeName>
  </EmployeeV1>
  <EmployeeV1>
    <EmployeeID>104</EmployeeID>
    <EmployeeName>Preety</EmployeeName>
  </EmployeeV1>
</ArrayOfEmployeeV1>
```

/api/v2/employees – We get employees with Employee Id, FirstName and LastName properties as shown below.



```
<?xml version='1.0'>
<ArrayOfEmployeeV2 xmlns:i='http://www.w3.org/2001/XMLSchema-instance'>
  <EmployeeV2>
    <EmployeeID>101</EmployeeID>
    <FirstName>Anurag</FirstName>
    <LastName>Mohanty</LastName>
  </EmployeeV2>
  <EmployeeV2>
    <EmployeeID>102</EmployeeID>
    <FirstName>Priyanka</FirstName>
    <LastName>Dewangan</LastName>
  </EmployeeV2>
  <EmployeeV2>
    <EmployeeID>103</EmployeeID>
    <FirstName>Sambit</FirstName>
    <LastName>Satapathy</LastName>
  </EmployeeV2>
  <EmployeeV2>
    <EmployeeID>104</EmployeeID>
    <FirstName>Preety</FirstName>
    <LastName>Tiwary</LastName>
  </EmployeeV2>
</ArrayOfEmployeeV2>
```

At the moment we are using the convention-based routing to implement the Web API versioning. We can also use the Attribute Routing instead of convention-based routing to implement the Web API versioning. What we need to do is, we need to use the [Route] attribute on methods in EmployeesV1Controller and EmployeesV2Controller as shown below.

EmployeesV1Controller

```
namespace WebAPIVersioning.Controllers
{
    public class EmployeesV1Controller : ApiController
    {
        List<EmployeeV1> employees = new List<EmployeeV1>()
        {
            new EmployeeV1() { EmployeeID = 101, EmployeeName = "Anurag"},
            new EmployeeV1() { EmployeeID = 102, EmployeeName = "Priyanka"},
            new EmployeeV1() { EmployeeID = 103, EmployeeName = "Sambit"},
            new EmployeeV1() { EmployeeID = 104, EmployeeName = "Preety"},
        };

        [Route("api/v1/employees")]
        public IEnumerable<EmployeeV1> Get()
        {
            return employees;
        }

        [Route("api/v1/employees/{id}")]
        public EmployeeV1 Get(int id)
        {
            return employees.FirstOrDefault(s => s.EmployeeID == id);
        }
    }
}
```

EmployeesV2Controller

```
namespace WebAPIVersioning.Controllers
{
    public class EmployeesV2Controller : ApiController
    {
        List<EmployeeV2> employees = new List<EmployeeV2>()
        {
            new EmployeeV2() { EmployeeID = 101, FirstName = "Anurag", LastName = "Mohanty"},
            new EmployeeV2() { EmployeeID = 102, FirstName = "Priyanka", LastName = "Dewangan"},
            new EmployeeV2() { EmployeeID = 103, FirstName = "Sambit", LastName = "Satapathy"},
            new EmployeeV2() { EmployeeID = 104, FirstName = "Preety", LastName = "Tiwary"},
        };
        [Route("api/v2/employees")]
        public IEnumerable<EmployeeV2> Get()
        {
            return employees;
        }
        [Route("api/v2/employees/{id}")]
        public EmployeeV2 Get(int id)
        {
            return employees.FirstOrDefault(s => s.EmployeeID == id);
        }
    }
}
```


As we are using the Attribute Routing, so we can safely comment following 2 route templates in **WebApiConfig.cs** file

```
namespace WebAPIVersioning
```

```
{
```

```
public static class WebApiConfig
```

```
{
```

```
public static void Register(HttpConfiguration config)
```

```
{
```

```
config.MapHttpAttributeRoutes();
```

```
//config.Routes.MapHttpRoute(
```

```
// name: "Version1",
```

```
// routeTemplate: "api/v1/employees/{id}",
```

```
// defaults: new { id = RouteParameter.Optional, controller = "EmployeesV1" }
```

```
//);
```

```
//config.Routes.MapHttpRoute(
```

```
// name: "Version2",
```

```
// routeTemplate: "api/v2/employees/{id}",
```

```
// defaults: new { id = RouteParameter.Optional, controller = "EmployeesV2" }
```

```
//);
```

```
}
```

```
}
```

```
}
```

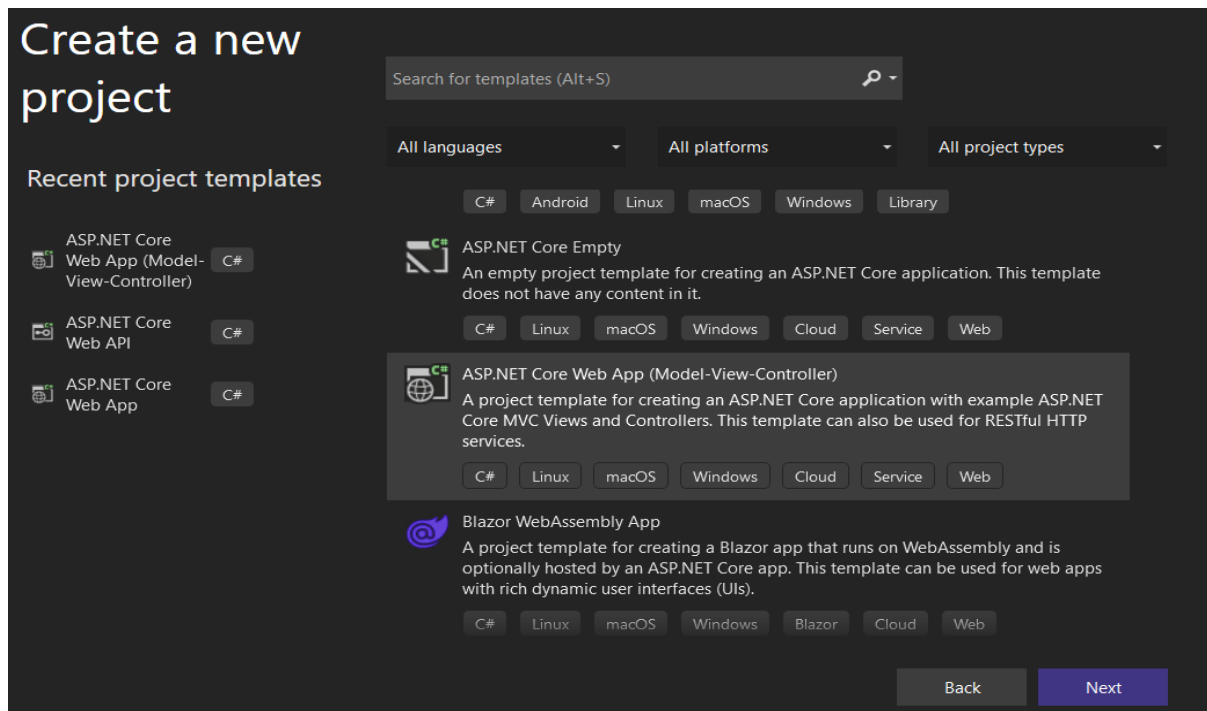
At this point, build the solution and test the application and it should work exactly the same way as before.

Practical 7

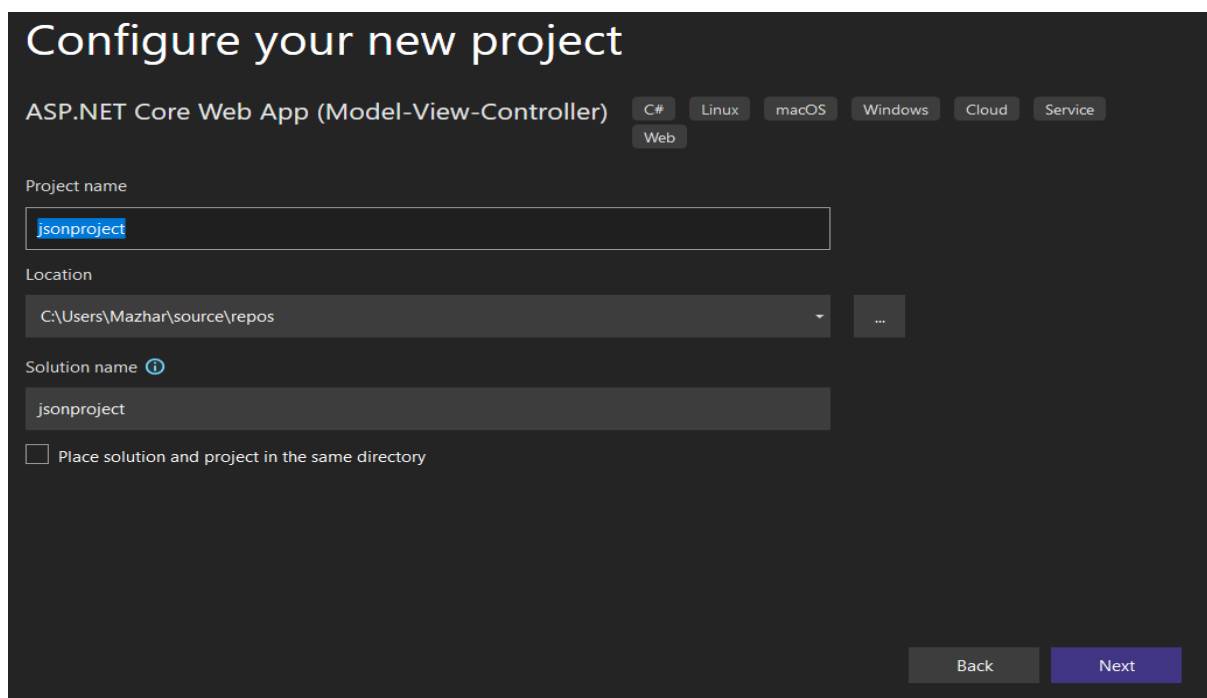
Aim: JSON Result in ASP.NET MVC

The JSON result is one of the most important Action results in the ASP.NET MVC application. This action result returns the data in JSON Format i.e. in the form of key-value pairs.

Step 1 : Create a new project ASP.NET Core Web App(Model-View-Controller)



Step 2:



Additional information

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Framework ⓘ
.NET 6.0 (Long-term support)

Authentication type ⓘ
None

☒ Configure for HTTPS ⓘ
☐ Enable Docker ⓘ

Docker OS ⓘ
Linux

Back Create

Step 3: Go to the controller folder and open HomeController file.

The screenshot shows the Visual Studio IDE with the 'jsonproject' solution open. The 'Solution Explorer' on the right shows the project structure: 'jsonproject' > 'Controllers' > 'HomeController.cs'. The 'HomeController.cs' file is open in the editor, showing the following code:

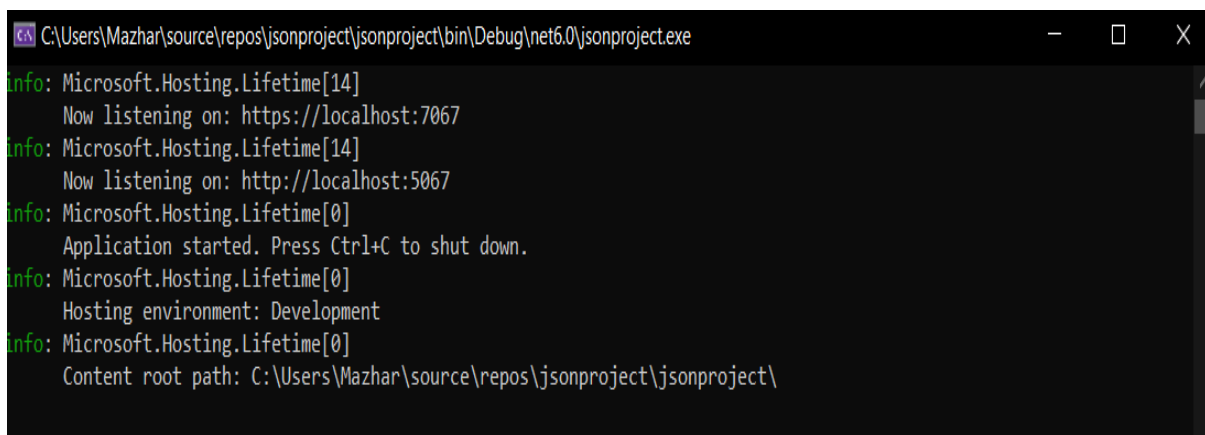
```
7 public class HomeController : Controller
8 {
9     private readonly ILogger<HomeController> _logger;
10
11     0 references
12     public HomeController(ILogger<HomeController> logger)
13     {
14         _logger = logger;
15     }
16
17     0 references
18     public IActionResult Index()
19     {
20         return View();
21     }
22
23     0 references
24     public IActionResult Privacy()
25     {
26         return View();
27     }
28
29     [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
30     0 references
31     public IActionResult Error()
32     {
33         return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
34     }
35 }
```

Step 4: add following code.

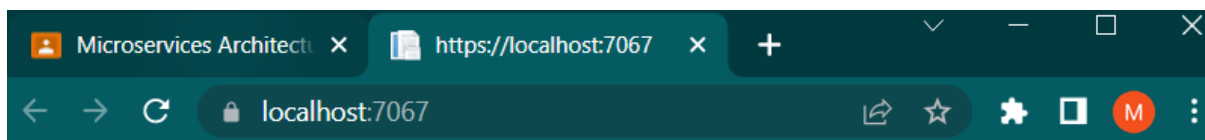
```
[HttpGet]
public JsonResult Index()
{
    return Json(new { Name = "John Smith", ID = 4, DateOfBirth = new
    DateTime(1999, 12, 31) });
}
```

Step 5: Run the application.

Output :-



```
C:\Users\Mazhar\source\repos\jsonproject\jsonproject\bin\Debug\net6.0\jsonproject.exe
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7067
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5067
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: C:\Users\Mazhar\source\repos\jsonproject\jsonproject\
```

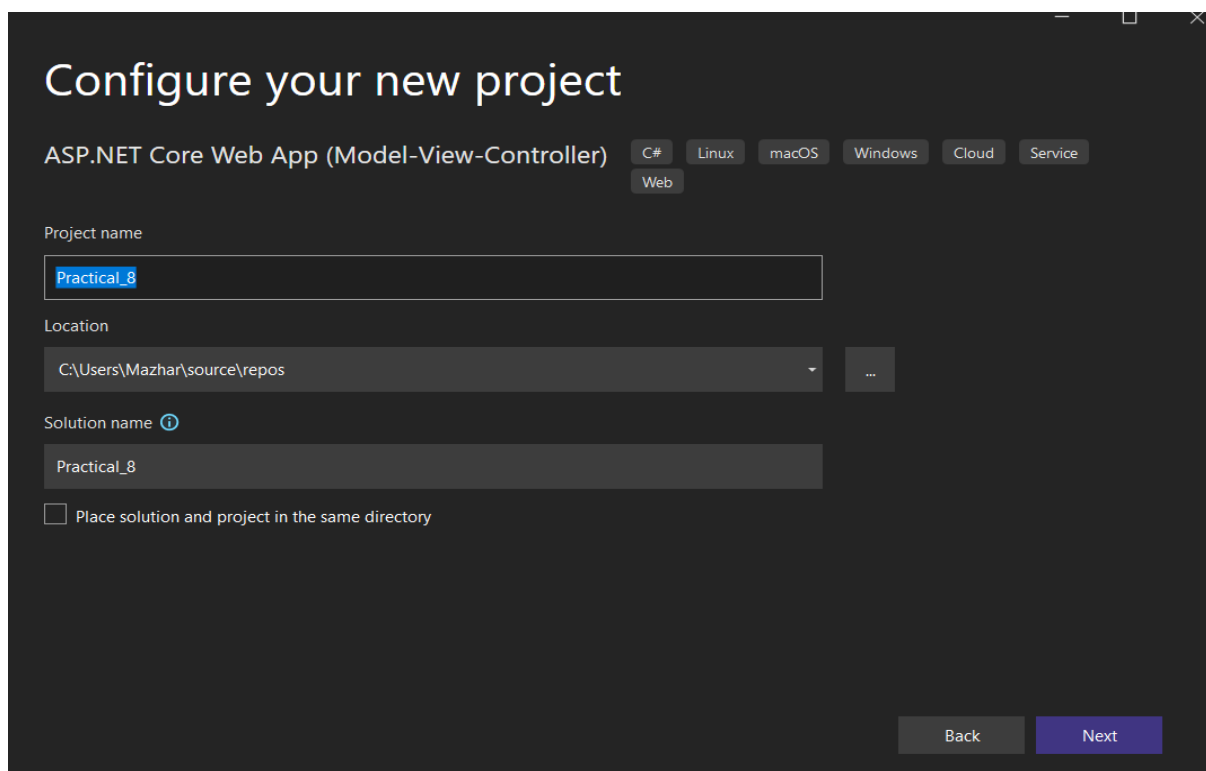
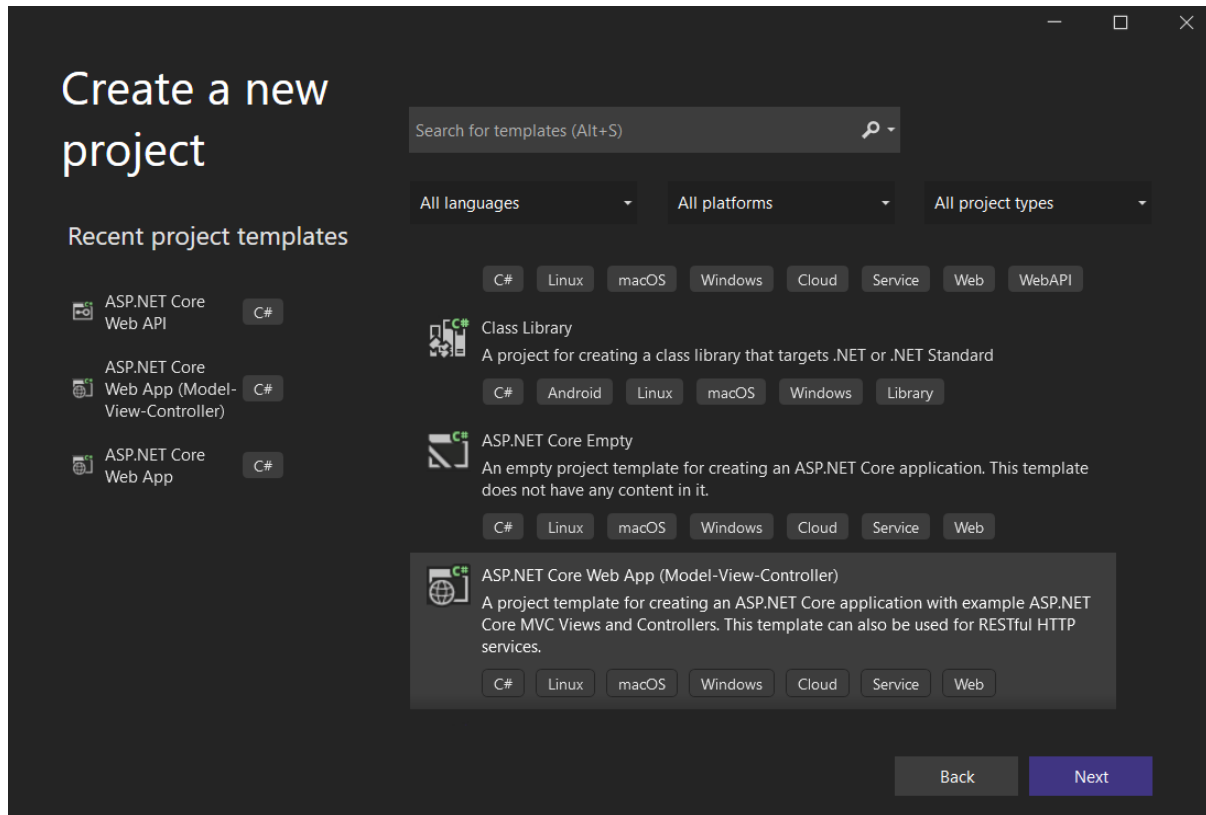


```
{"name":"Mazhar Solkar","id":4,"dateOfBirth":"1999-12-31T00:00:00"}
```

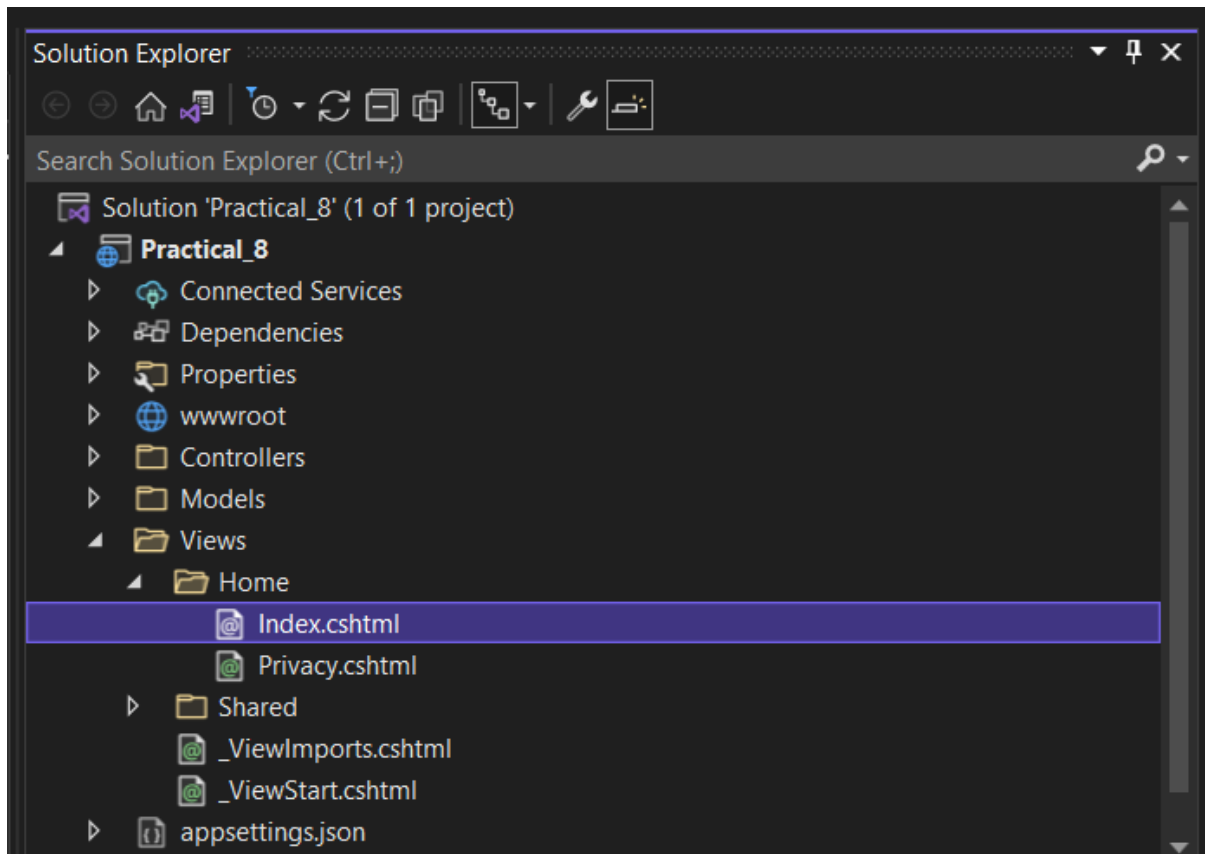
Practical 8

Aim: JavaScript Result in ASP.NET MVC

Step 1:



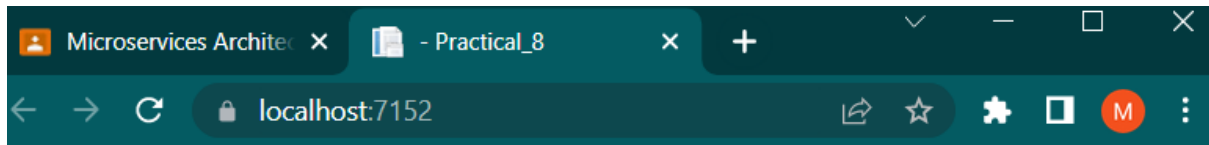
Step 2:



Index.cshtml

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script>
      $(document).ready(function(){
        $("#hide").click(function(){
          $("p").hide();
        });
        $("#show").click(function(){
          $("p").show();
        });
      });
    </script>
  </head>
  <body>
    <p><b>Msc-IT Part I Mazhar Solkar</b><br />
    <b>Sathaye College</b></p>
    <button id="hide">Hide</button>
    <button id="show">Show</button>
  </body>
</html>
```

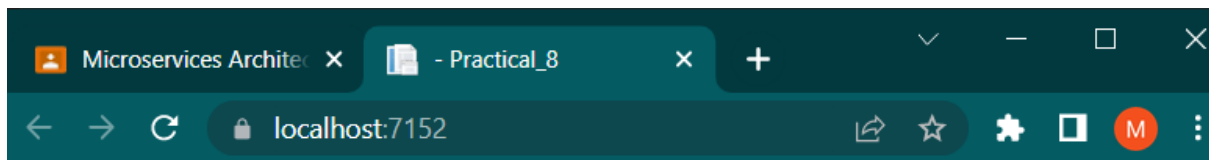
Output :



Practical_8 Home Privacy

Msc-IT Part I Mazhar Solkar
Sathaye College

Hide Show



Practical_8 Home Privacy

Hide Show