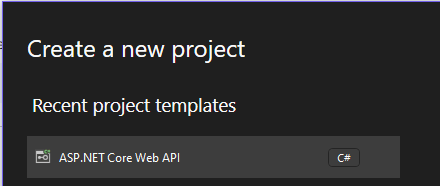
#### **Index**

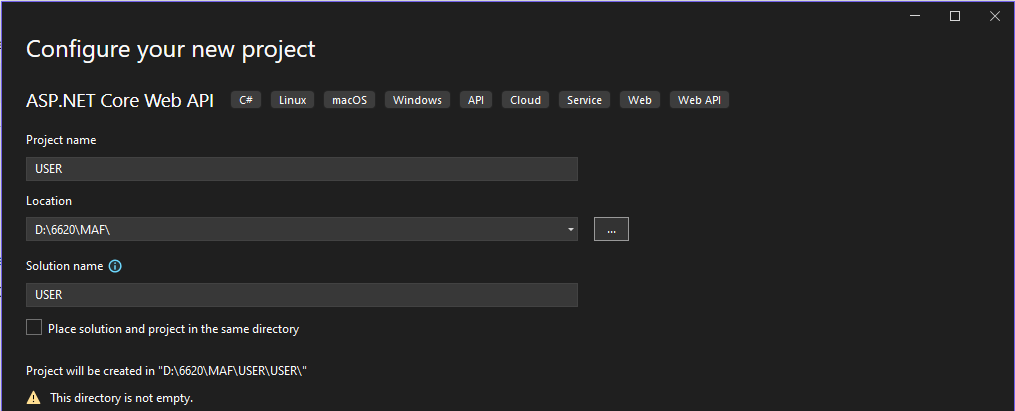
| **Sr.o No** | **Practical** | **Date** | **Remark** |
| --- | --- | --- | --- |
| 1 | Creating console based ASP.NET core applications. |  |  |
| 2 |  |  |  |
| 3 | Creating MVC Project using ASP.NET core. | Jul 15, 2024 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Jul 15, 2024 and Jul 22, 2024

**Step 1 :** Create a Project - ASP.NET Core Web API

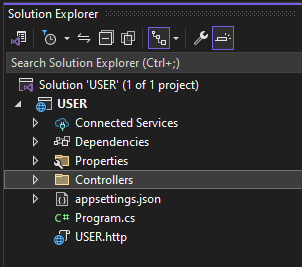


**Step 2 :** Named project as “USER”

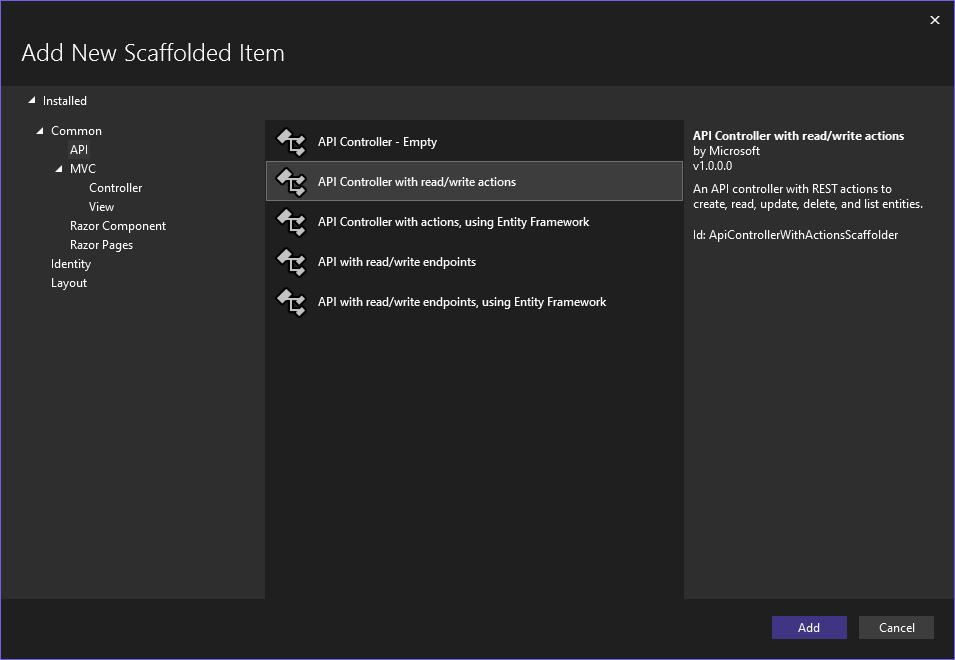


**Step 3 :** Delete default weather controller files

Folder structure will be like



**Step 4 :** create a new Controller of API → API Controller with read/write action → name it as UserController.cs

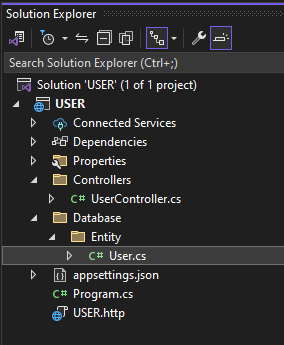


**Step 5 :** Create a new Folder in root Directory named as Database

**Step 6 :** Create a New Folder in Database Dir named as Entity

**Step 7 :** In Entity create a class file named as User.cs

Folder structure will be like



**Step 8 :** place the below code in User.cs file

public class UserEntity

{

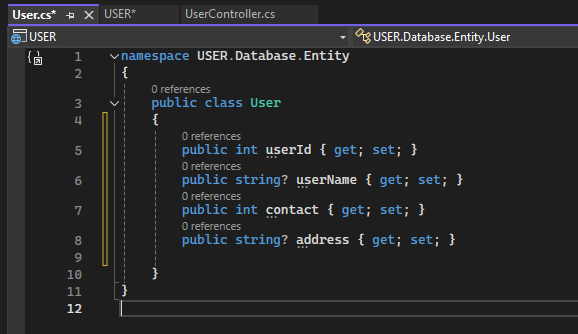
**public int userId { get; set; }**

**public string? userName { get; set; }**

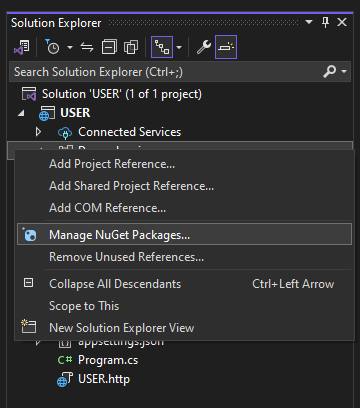
**public int contact { get; set; }**

**public string? address { get; set; }**

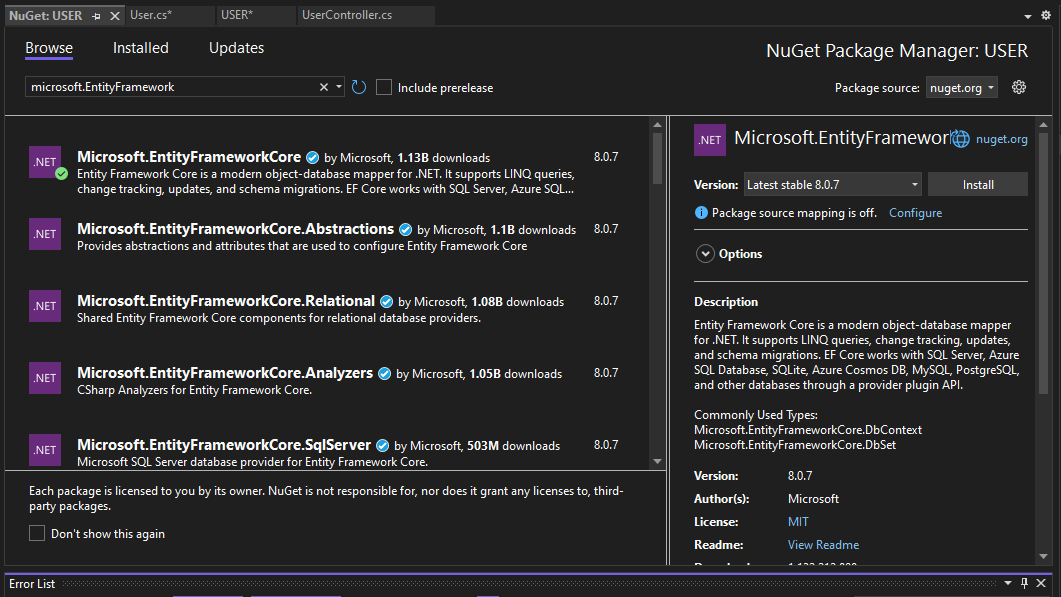
}

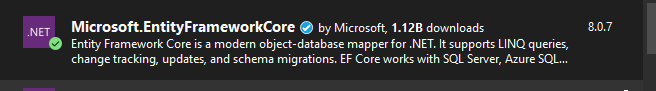


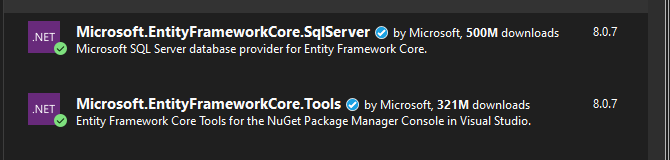
**Step 9 :** Go to dependencies and right click on it and go to ManageNuGet Packages



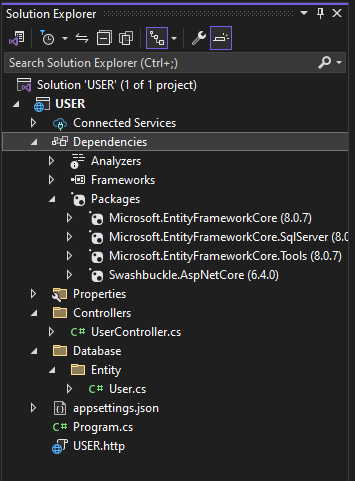
**Step 10 :** Go to browse and Install the below three from browse section of NuGet (Stable version)



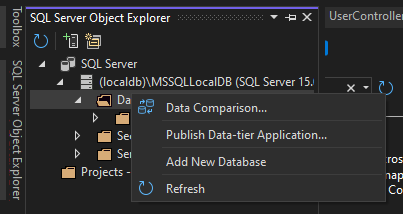




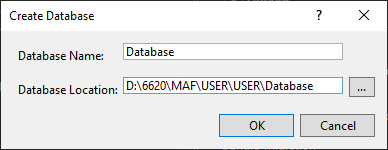
Package will be like this



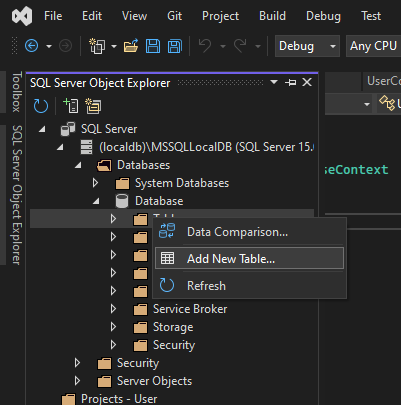
**Step 11 :** Go to SQL Server Object Explorer → go to Database and add new Database name as Database

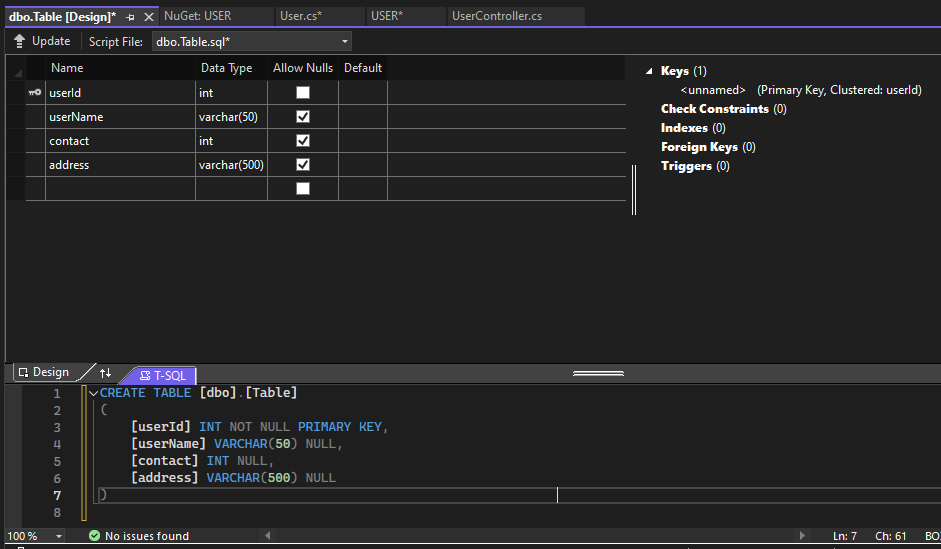


**Step 12 :** Save it to your USER project Database Folder.



**Step 13 :** Add new Table into Database Create field as the same name used in User.cs file





CREATE TABLE [dbo].[Table]

(

[userId] INT NOT NULL PRIMARY KEY,

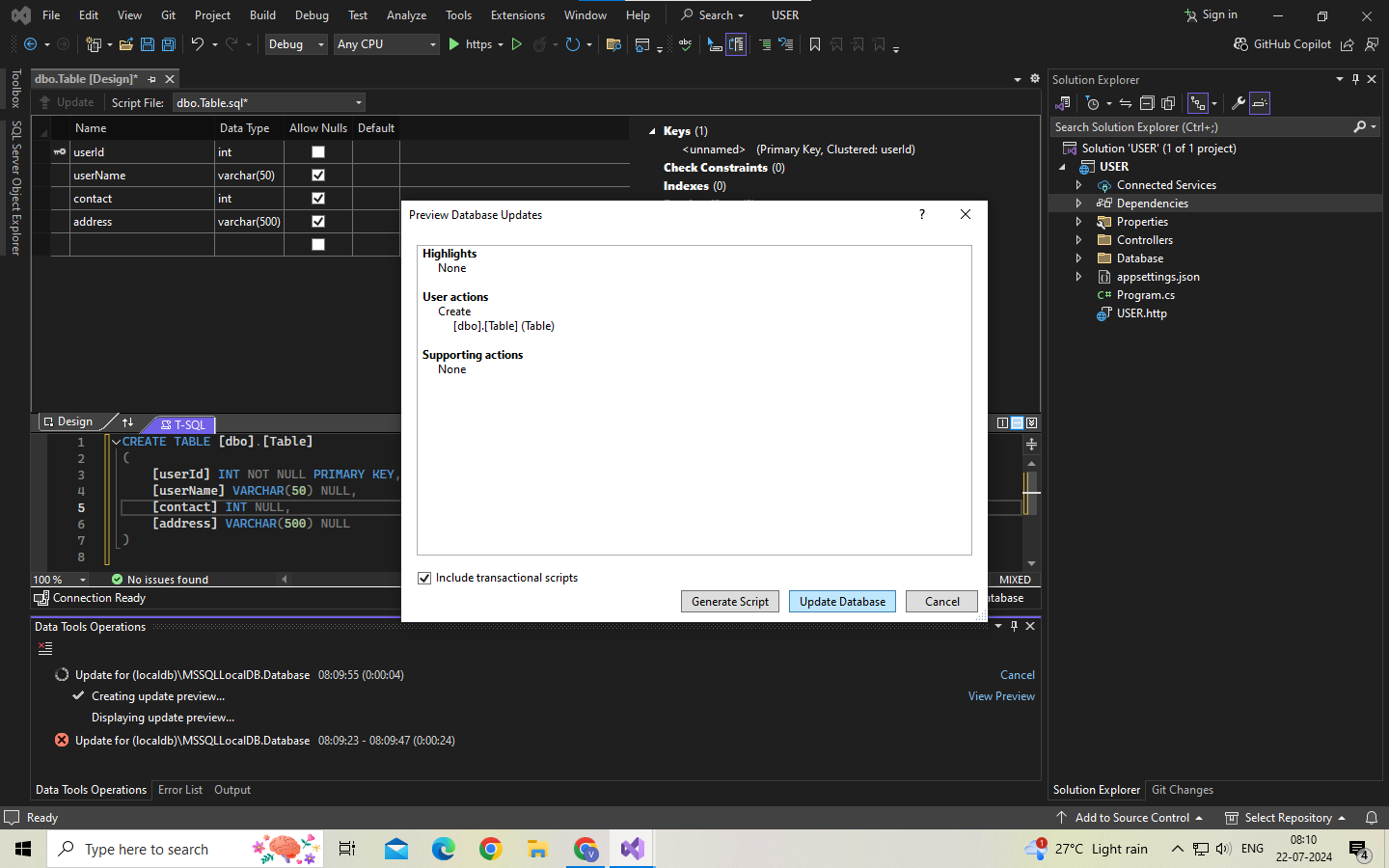
[userName] VARCHAR(50) NULL,

[contact] INT NULL,

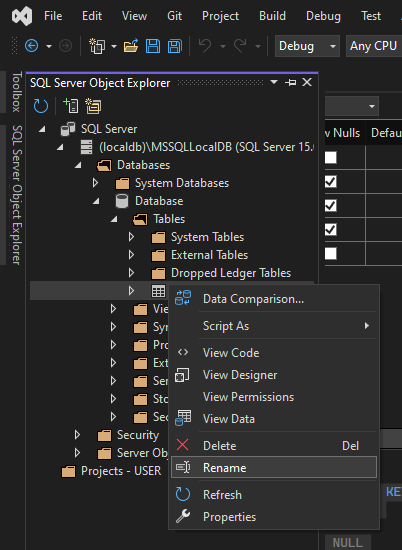
[address] VARCHAR(500) NULL

)

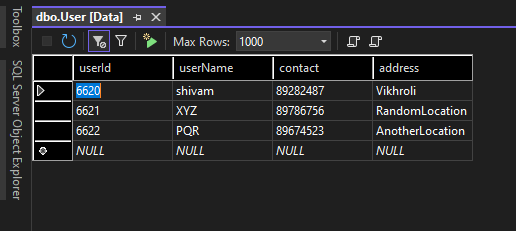
**Step 14 :** Update database



**Step 15 :** Rename the table as User



**Step 16 :** Insert data into Table



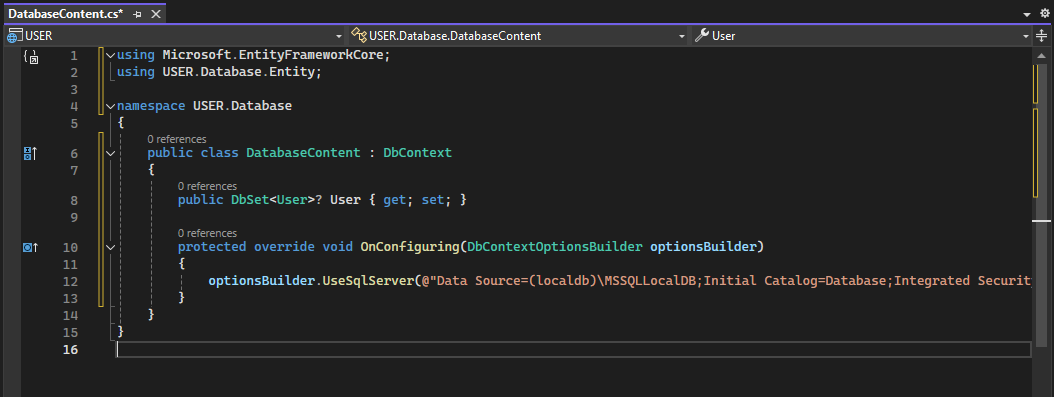
**Step 17 :** Get the connection string from the Database



**Connection String**

Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Database;Integrated Security=True;Connect Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi Subnet Failover=False

**Step 18 :** Create a DatabaseContent.cs file in Database folder and place th copied Connection string in it



**Code**

**using Microsoft.EntityFrameworkCore;**

**using USER.Database.Entity;**

namespace USER.Database

{

public class DatabaseContent : **DbContext**

{

**public DbSet<User>? User { get; set; }**

**protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)**

**{**

**optionsBuilder.UseSqlServer(@"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Database;Integrated Security=True;Connect Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi Subnet Failover=False");**

**}**

}

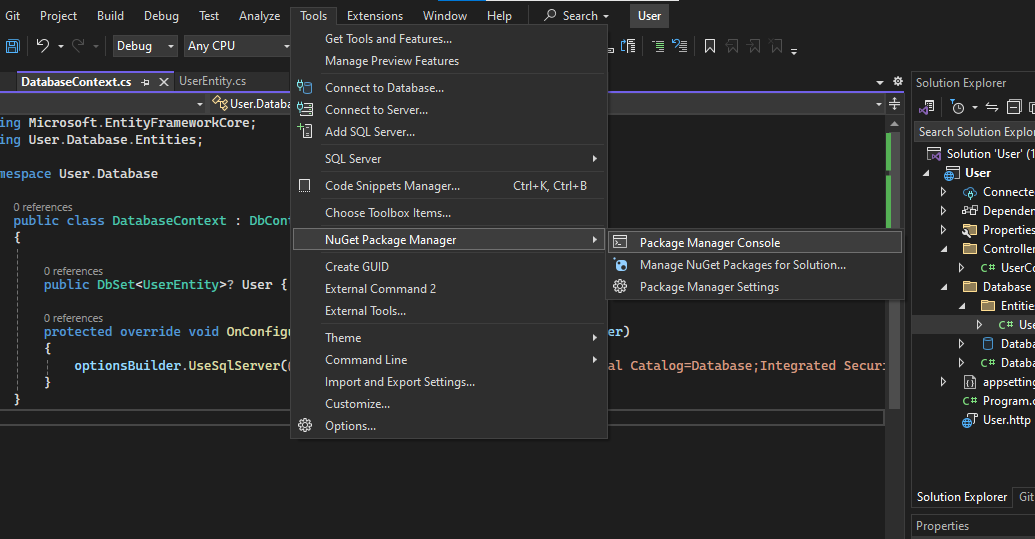
}

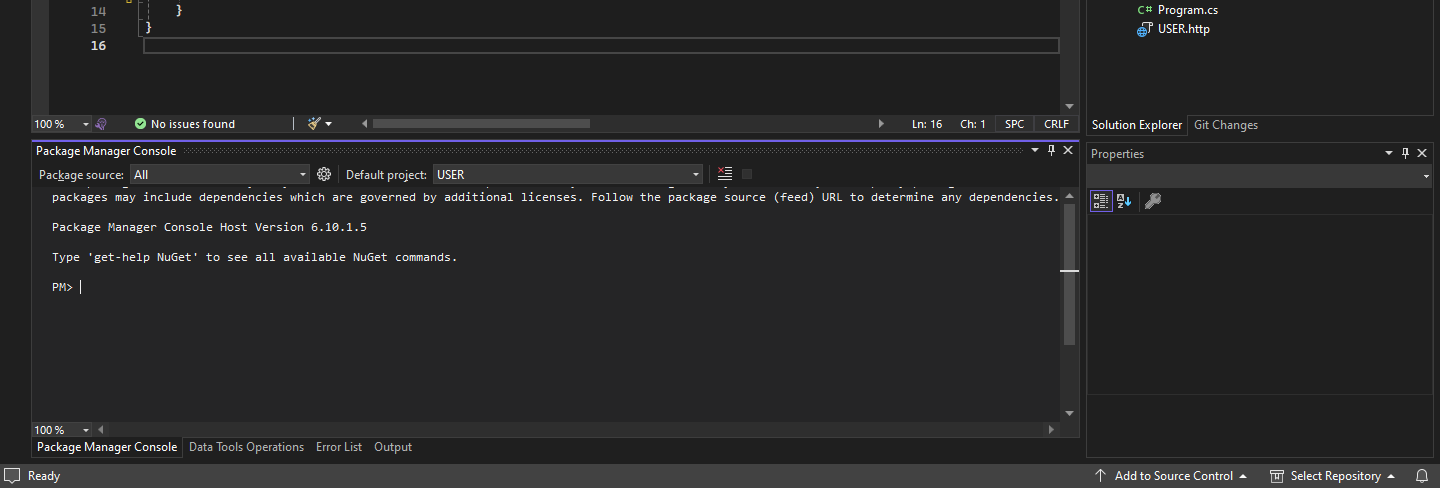
**Step 19 :** Go to Tool → NuGet Package Manager → Package Manager console.

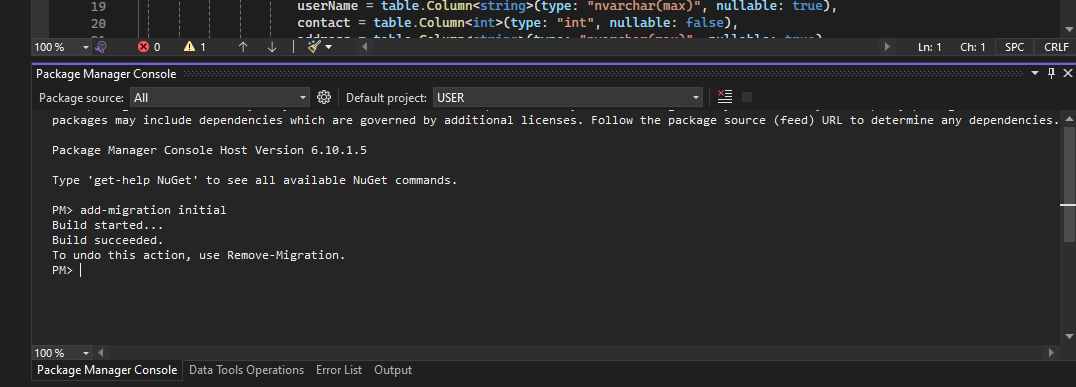
The PM console will open.

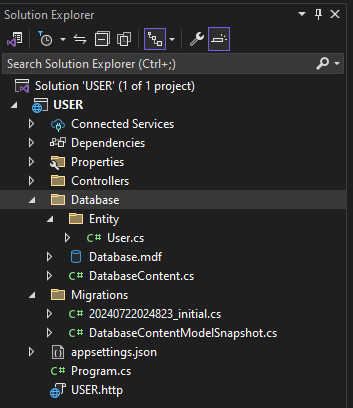
Run command

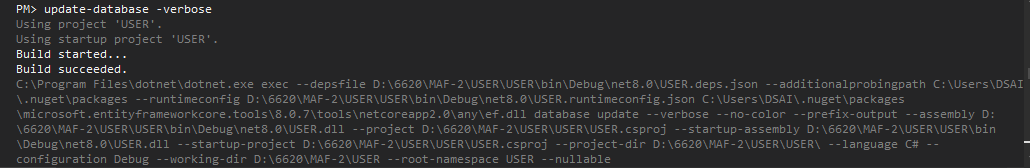
1. Add-migration initial (Migration Folder will be visible to at root Dir (USER))
2. Update-database -verbose

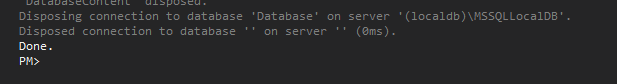












**Step 20 :** Go to userController.cs file and place the below code

**Code**

**using Microsoft.AspNetCore.Mvc;**

**using USER.Database.Entity;**

using USER.Database;

namespace USER.Controllers

{

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

[ApiController]

public class UserController : ControllerBase

{

**DatabaseContent dbContext;**

**public UserController()**

**{**

**dbContext = new DatabaseContent();**

**}**

// GET: api/<UserController>

[HttpGet]

**public IEnumerable<User> Get()**

**{**

**return dbContext.User.ToList();**

**}**

// GET api/<UserController>/5

[HttpGet("{id}")]

**public User Get(int id)**

**{**

**return dbContext.User.Find(id);**

**}**

// POST api/<UserController>

[HttpPost]

**public IActionResult Post([FromBody] User user)**

**{**

**dbContext.User.Add(user);**

**dbContext.SaveChanges();**

**return CreatedAtAction(nameof(Get), new { id = user.userId }, user);**

**}**

// PUT api/<UserController>/5

[HttpPut("{id}")]

**public IActionResult Put(int id, [FromBody] User updatedUser)**

**{**

**if (id != updatedUser.userId)**

**{**

**return BadRequest("User ID mismatch");**

**}**

**var existingUser = dbContext.User.Find(id);**

**if (existingUser == null)**

**{**

**return NotFound();**

**}**

**existingUser.userName = updatedUser.userName;**

**existingUser.contact = updatedUser.contact;**

**existingUser.address = updatedUser.address;**

**dbContext.SaveChanges();**

**return NoContent();**

**}**

// DELETE api/<UserController>/5

[HttpDelete("{id}")]

**public IActionResult Delete(int id)**

**{**

**var user = dbContext.User.Find(id);**

**if (user == null)**

**{**

**return NotFound();**

**}**

**dbContext.User.Remove(user);**

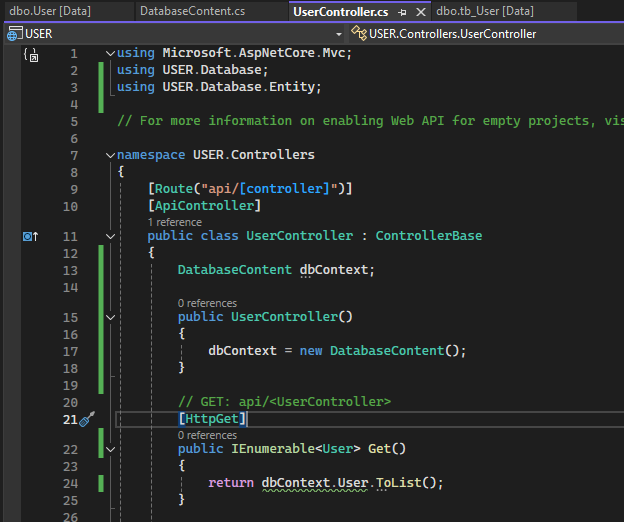
**dbContext.SaveChanges();**

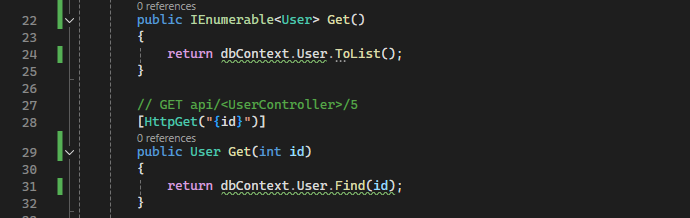
**return NoContent();**

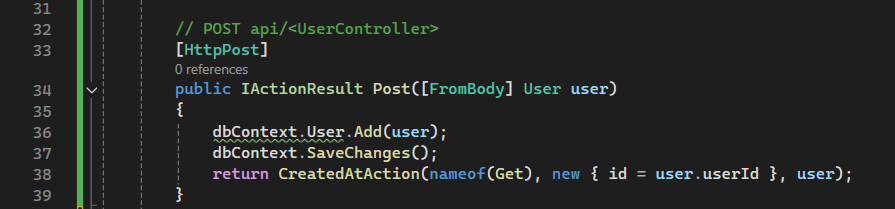
**}**

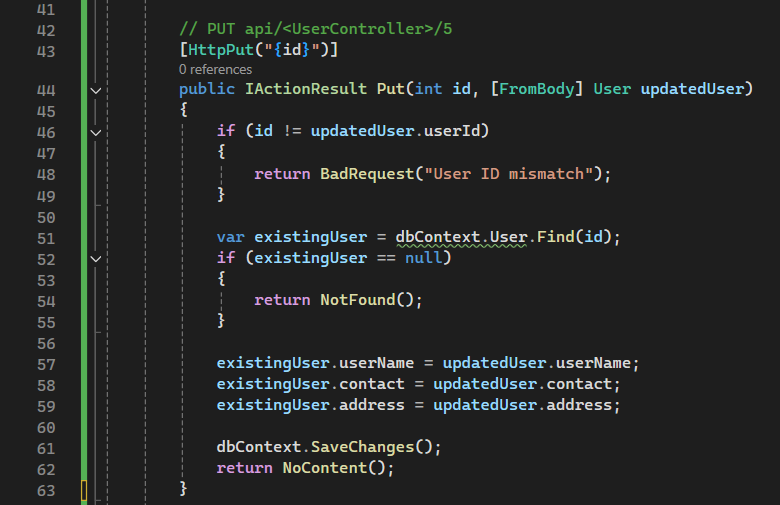
}

}





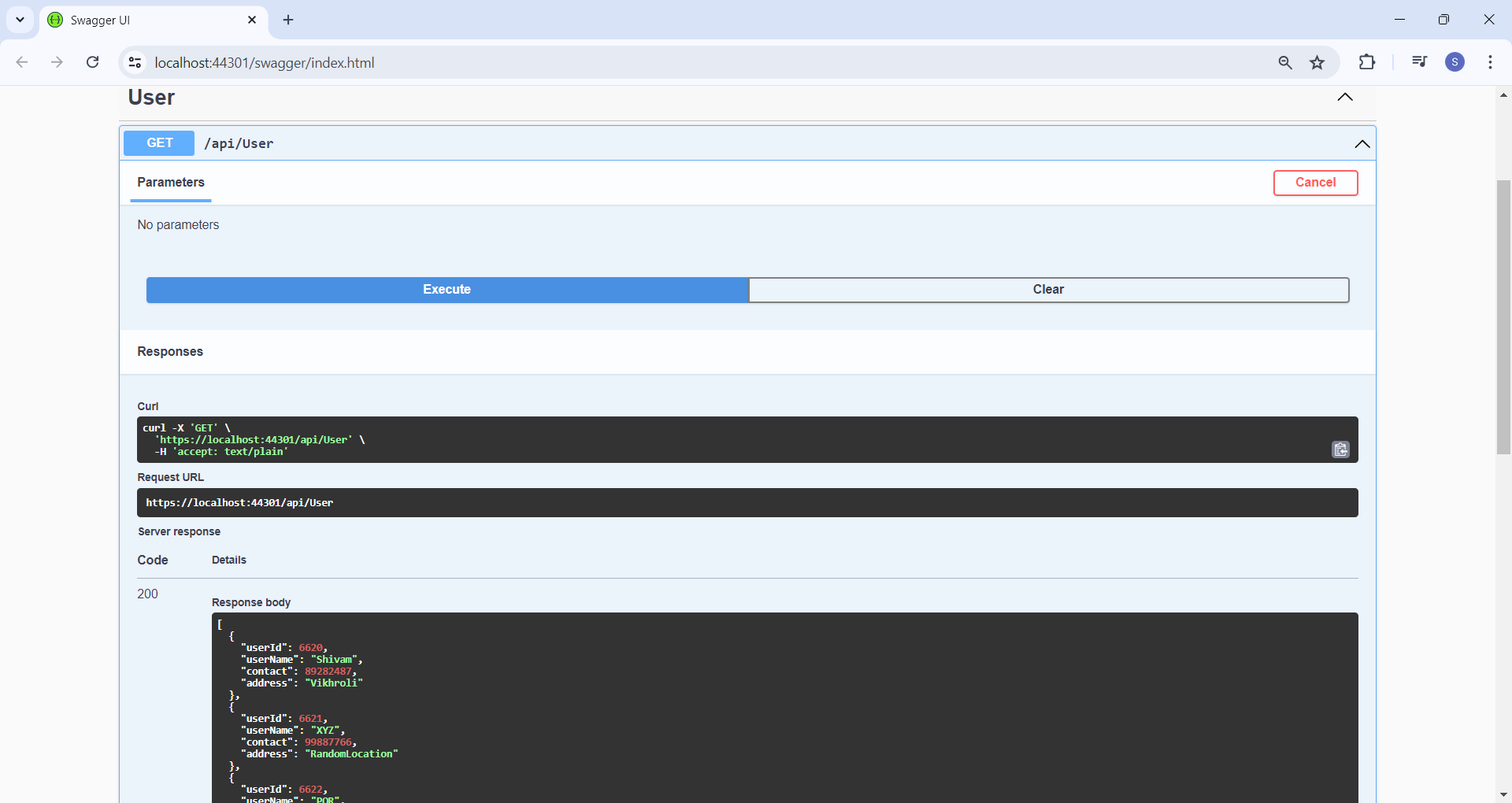




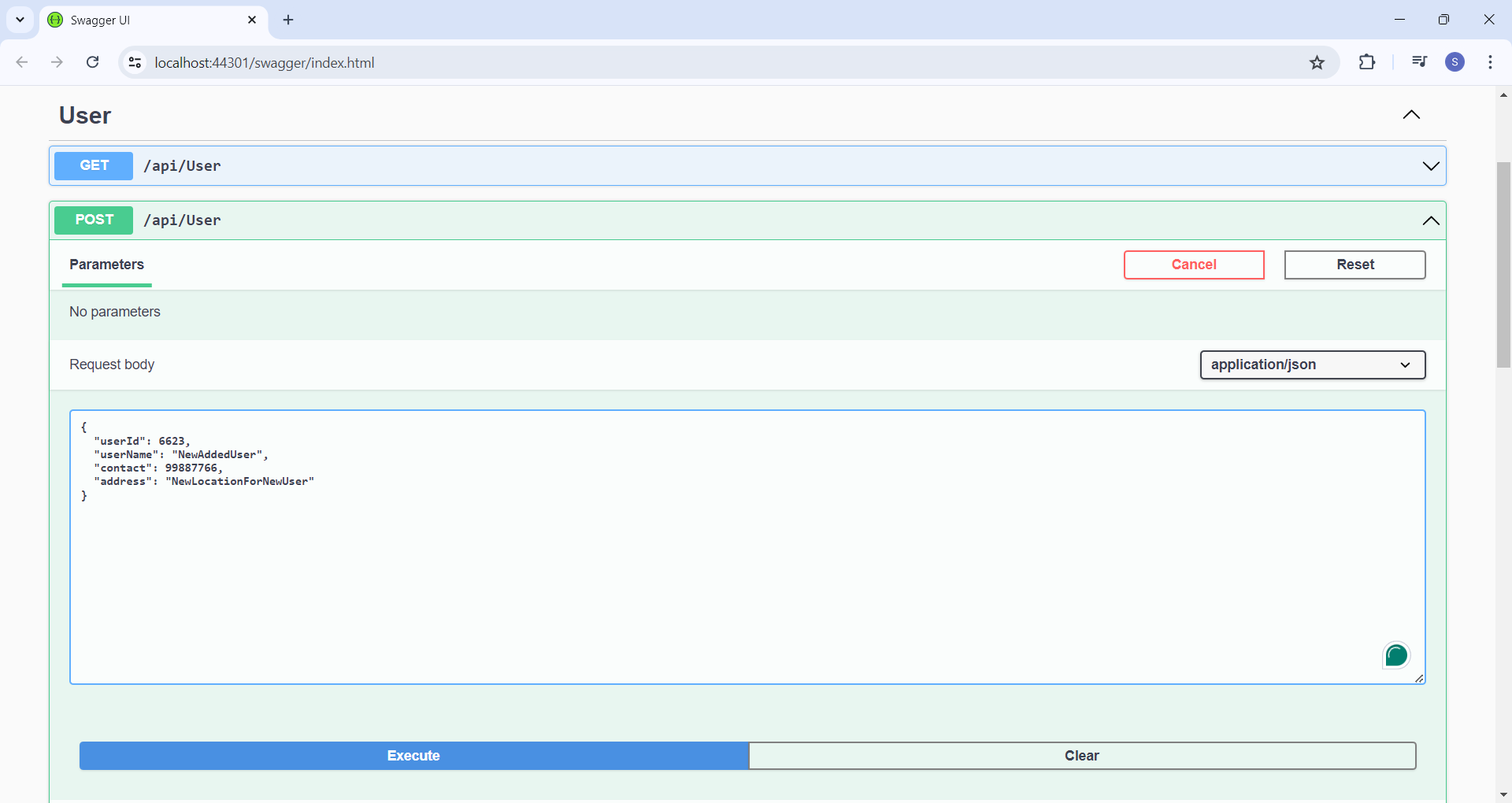


**Step 21 :** Run the project

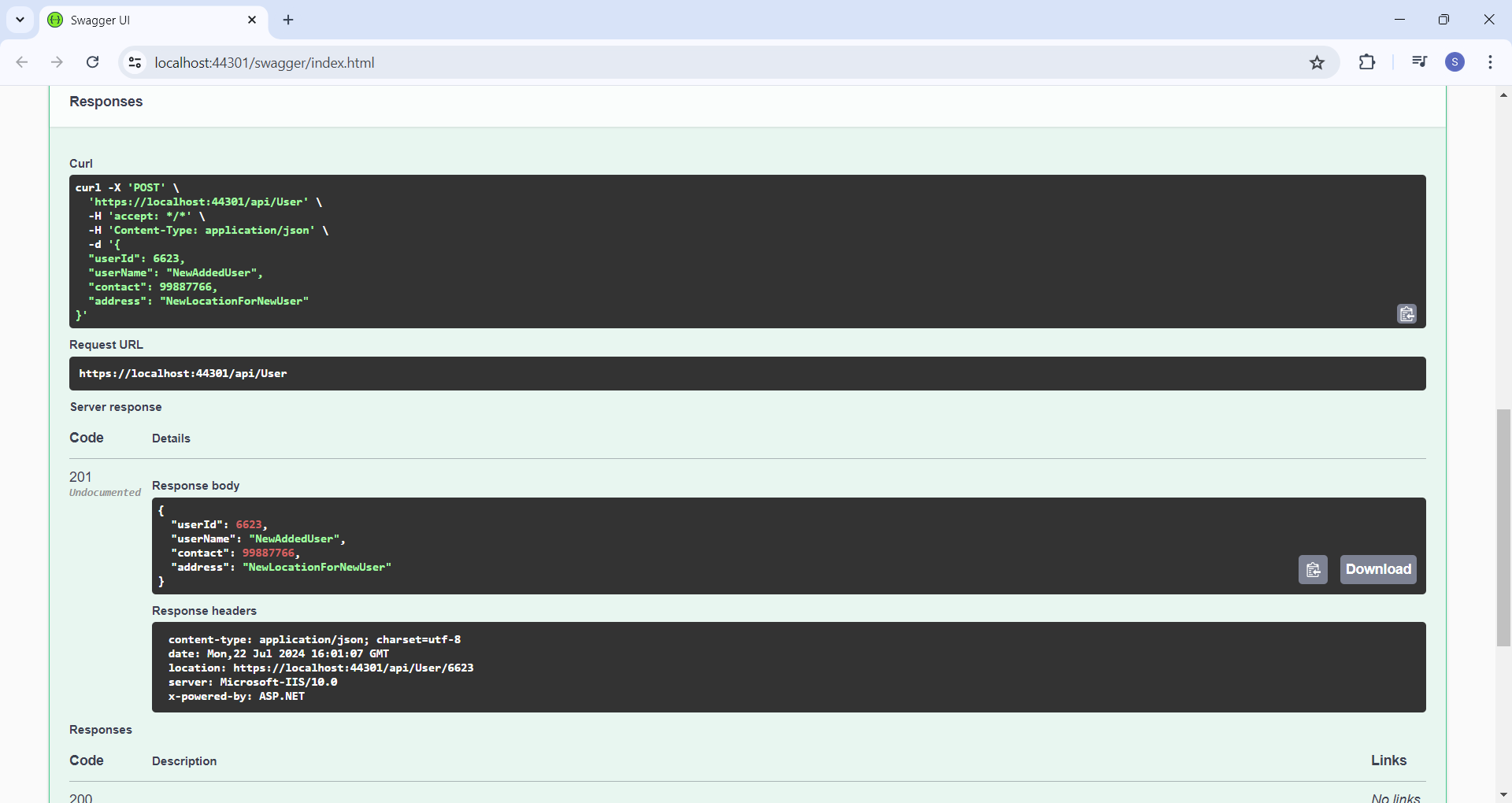
GET Request (Data from the User table is visible in Response body)



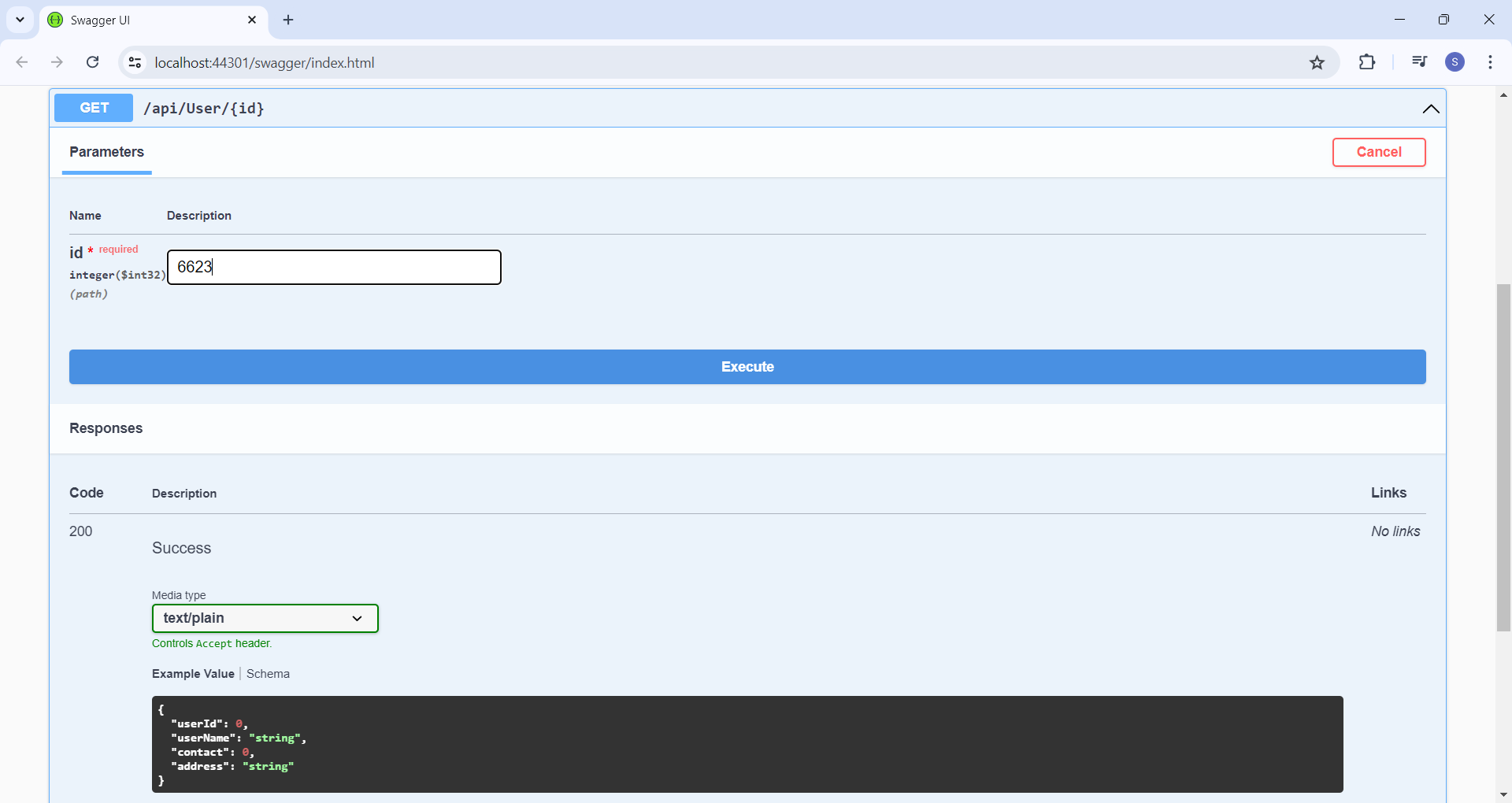
POST Request (Inserting Record fro ID 6623 in to the User table)



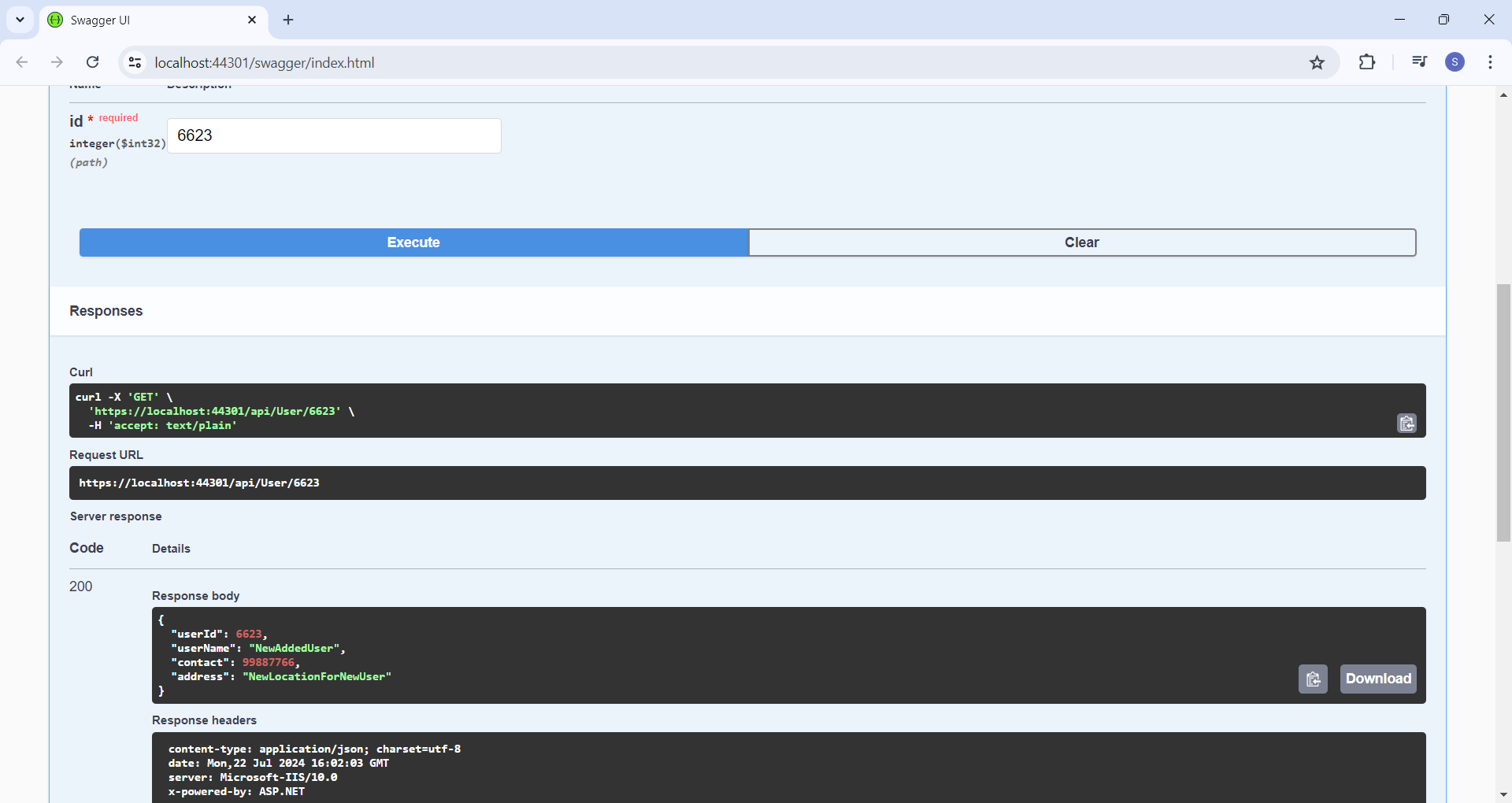
Response code 200



GET Request by ID (Fetching Data for ID 6623)



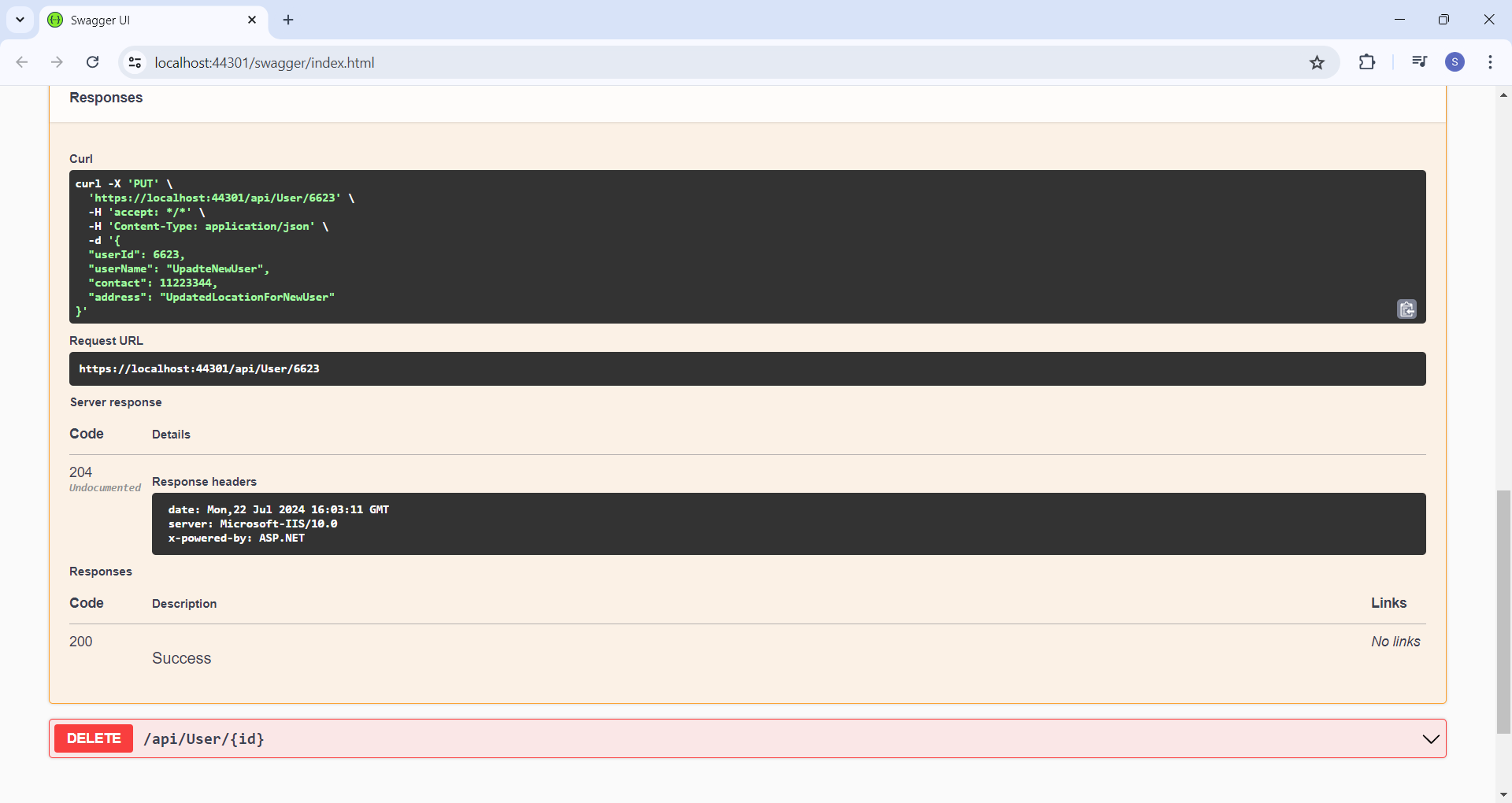
(Data from ID 6623 Visible in Response body)



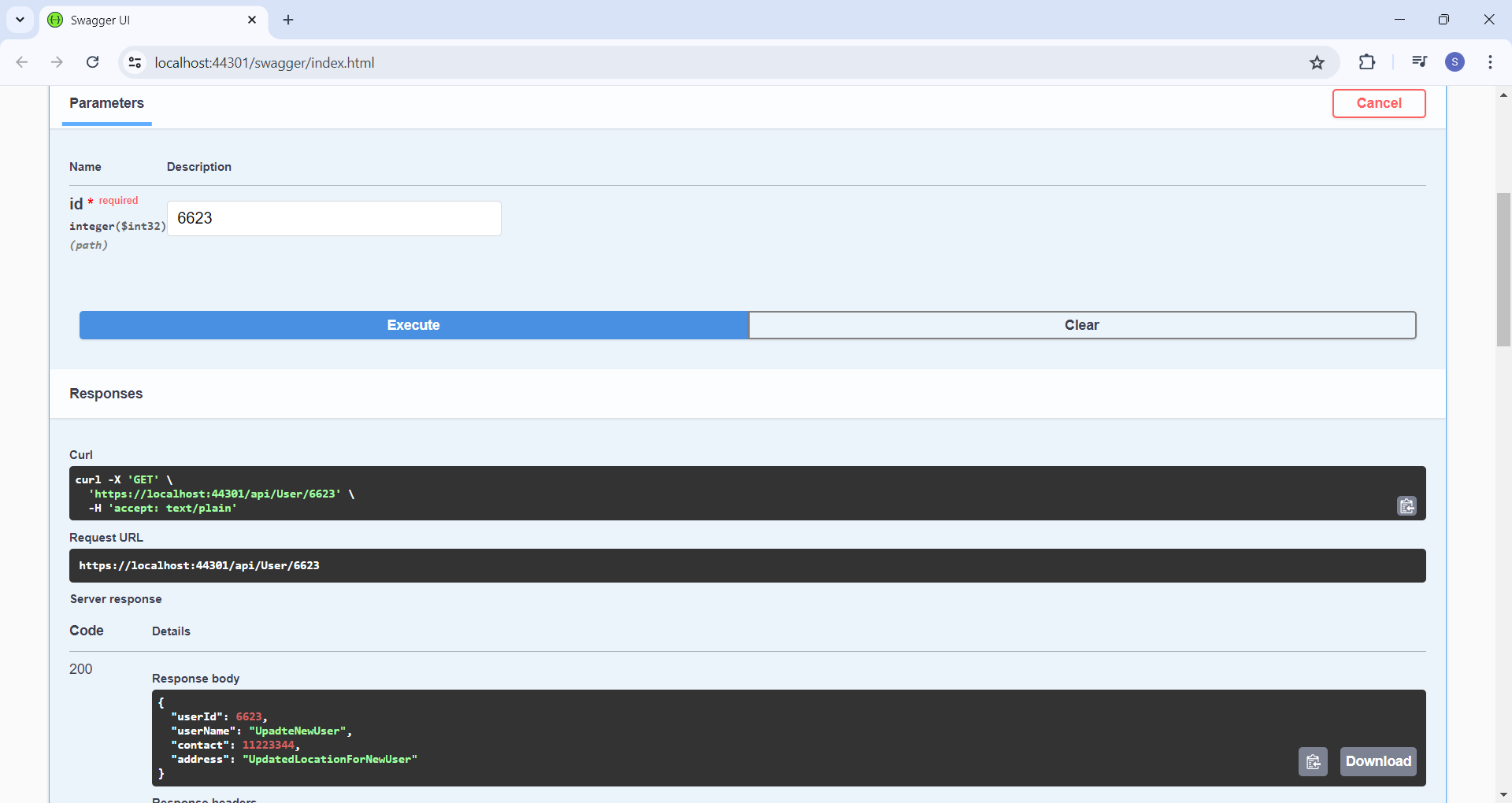
PUT request (Updating record for ID 6623)



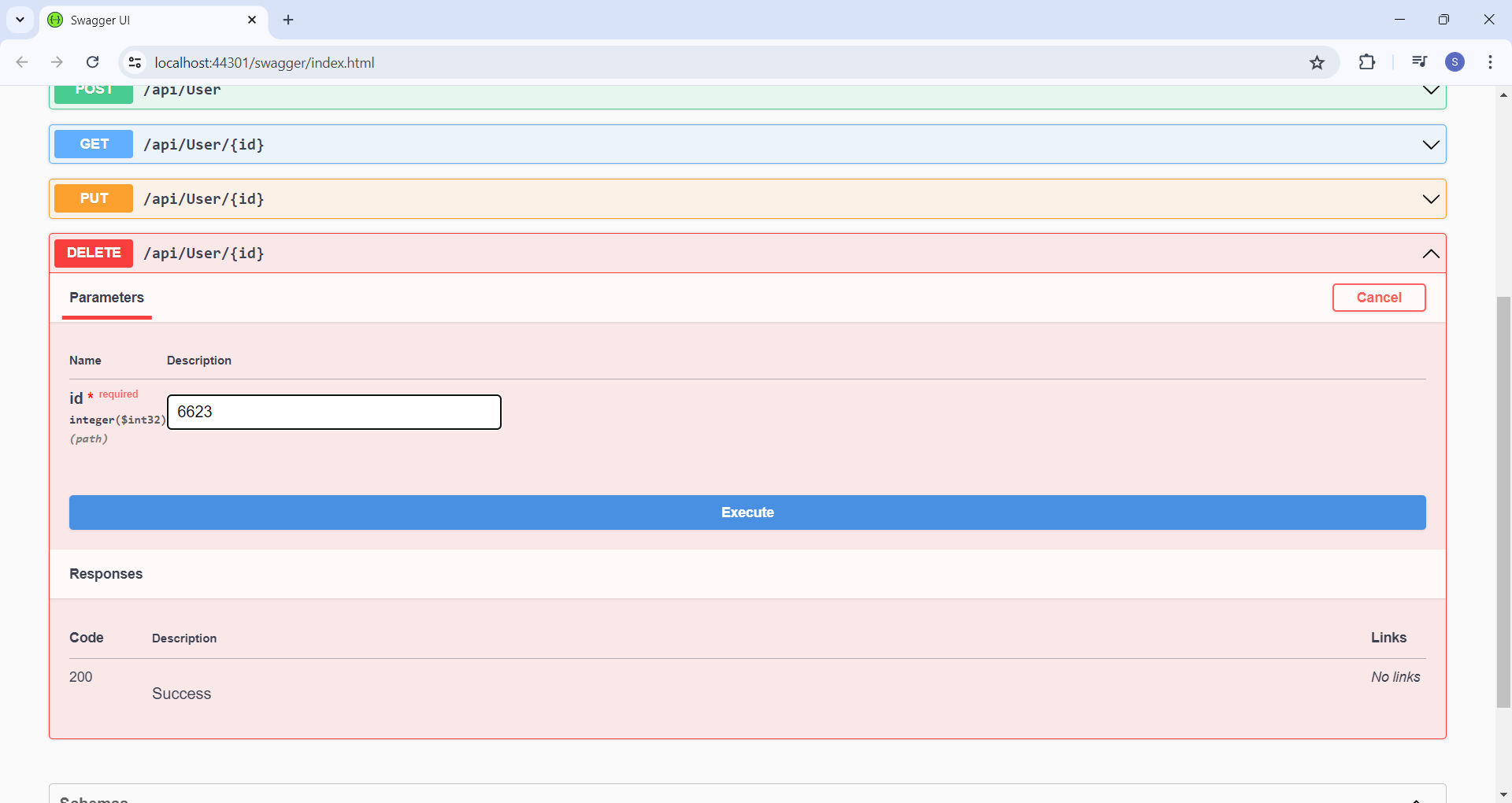
Response code 200



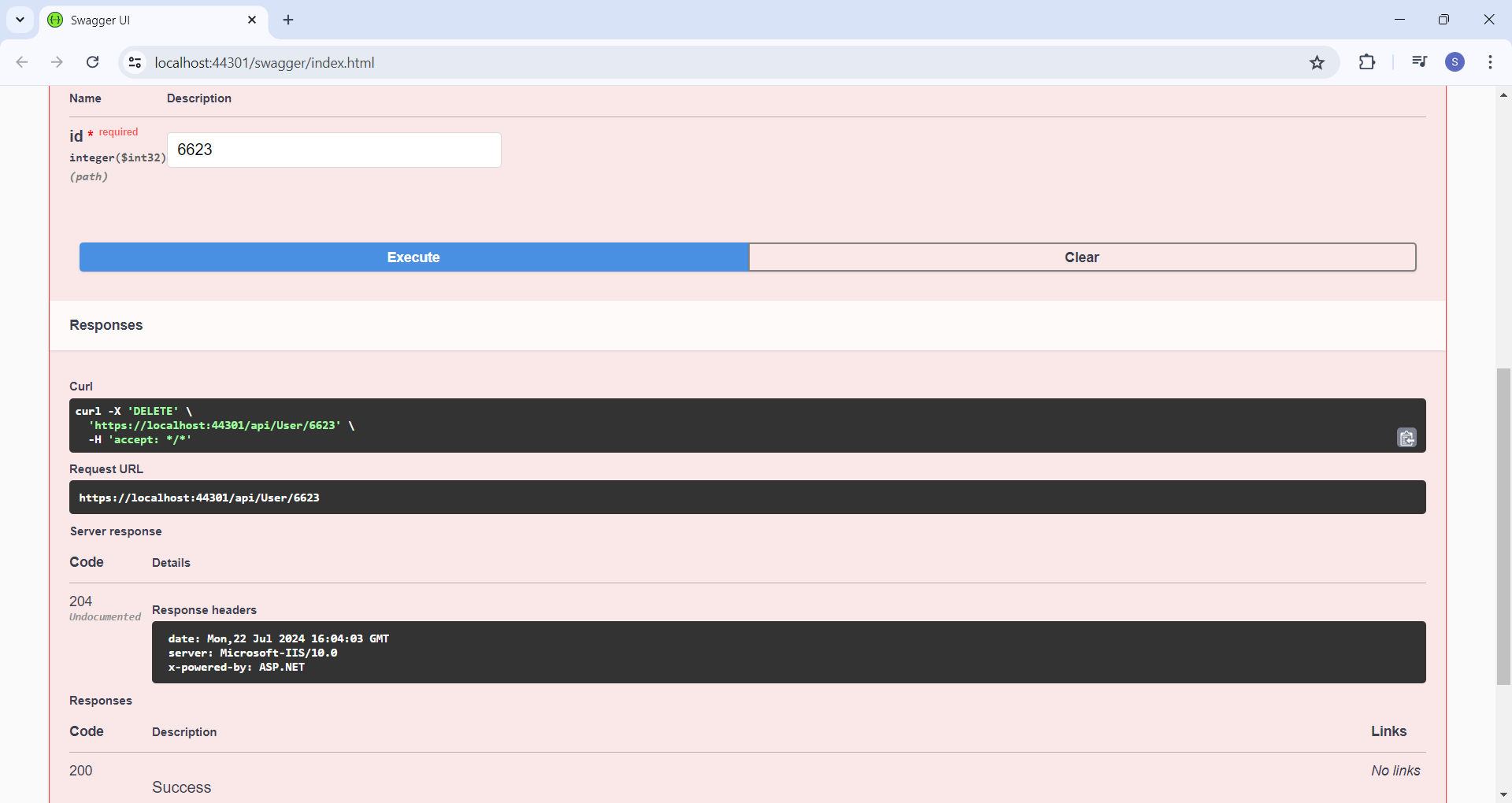
GET request by ID - Verifying Data updated for ID 6623



DELETE Request (Deleting record for ID 6623)

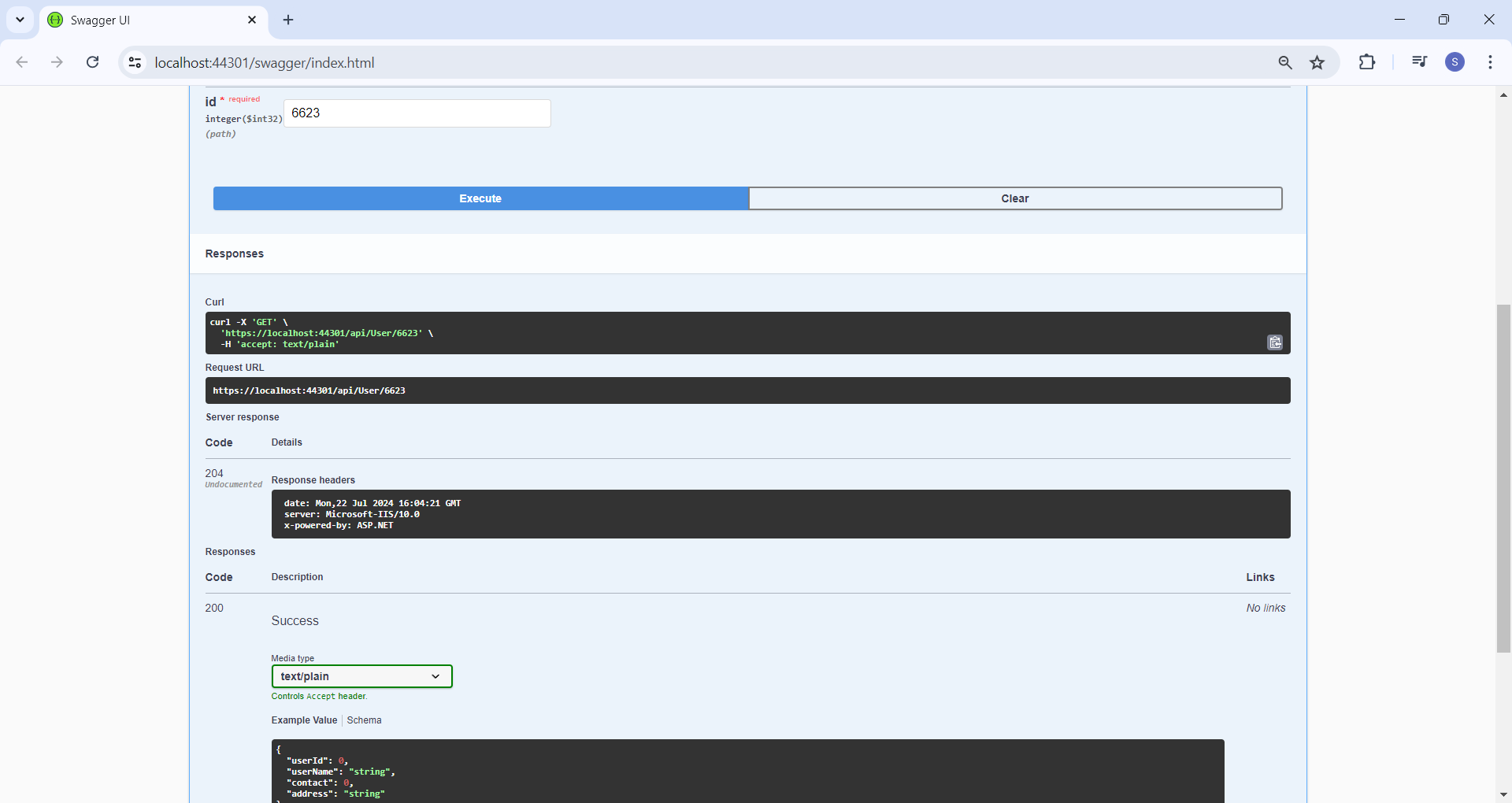


Response code 200



GET request by ID - Verifying Data Deleted for ID 6623

Response code 200

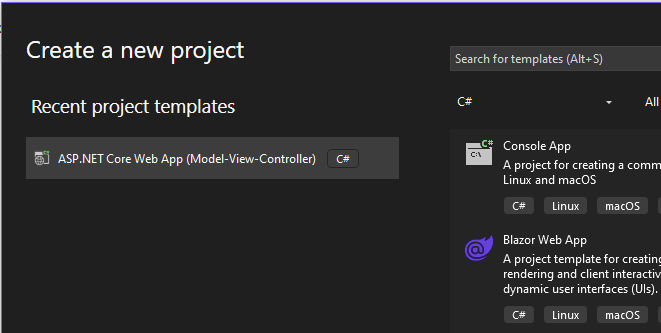


#### 

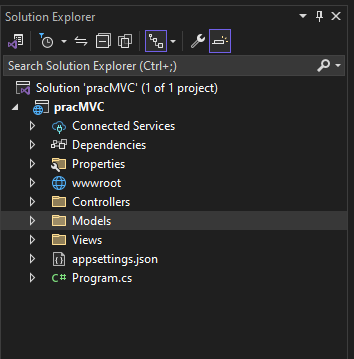
#### **Practical 3 - Creating MVC Project using ASP.NET core.**

Jul 15, 2024

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



Folder Structure



**STEP 1 -** Go to Models folder and create a class file **StockQuote.cs**

namespace pracMVC.Models

{

public class StockQuote

{

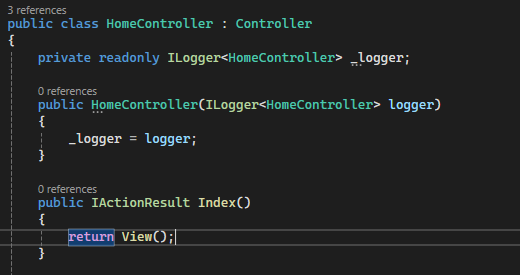
public string? symbol { get; set; }

public int price { get; set; }

}

}

**STEP 2 -** Go to Controllers folder → go to HomeControllers.cs file look for the function name **IActionresult()**

****

**And replace the code**

public IActionResult Index()

{

**var model = new StockQuote { symbol = "Nike", price = 3200 };**

**return View(model);**

}

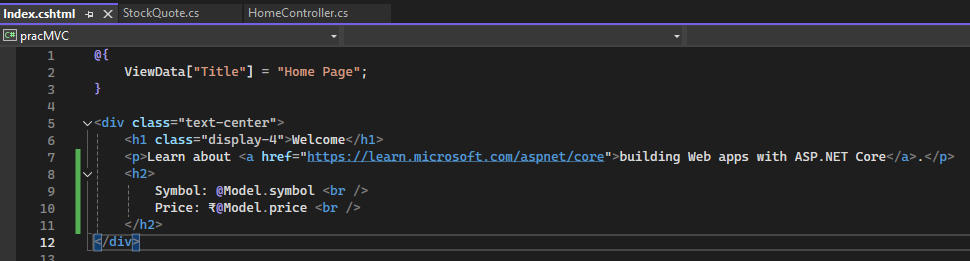
**STEP 3 -** go to Views Folder → Home folder → go to **Index.cshtml** file place the below code in div tag

<h2>

Symbol: @Model.symbol <br />

Price: ₹@Model.price <br />

</h2>

****

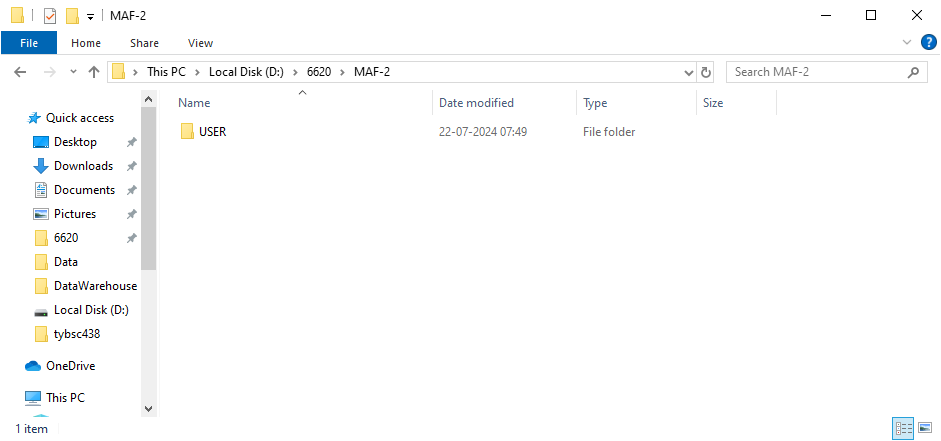
**Output**

****

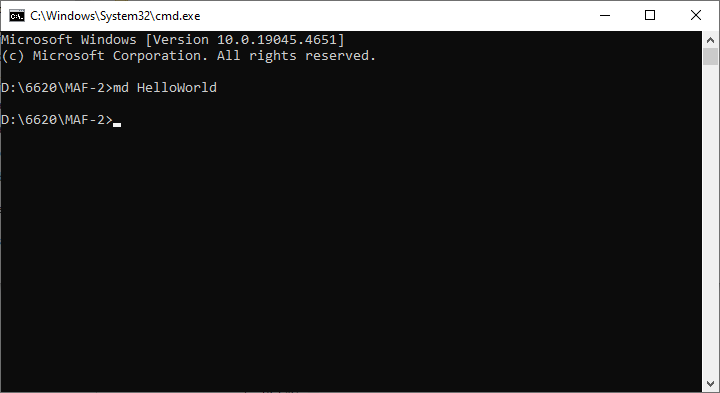
#### Jul 24, 2024 - USER

**Console based app**

**STEP 1 :** Go to your MAF Practical folder and open it n command line



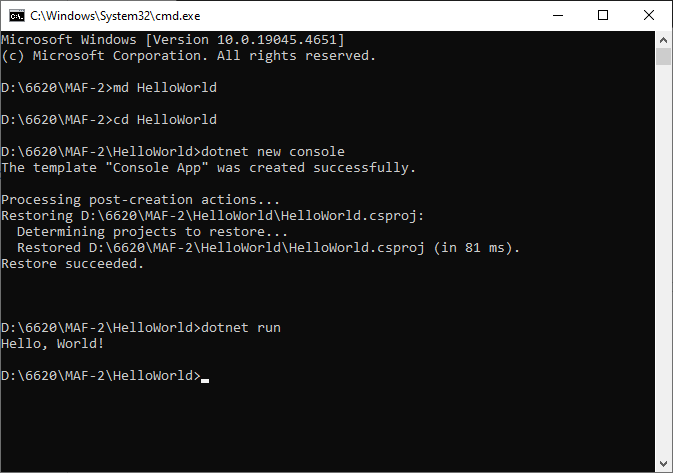
**STEP 2:** Create a directory named as “HelloWorld” using md command



**STEP 3 :** Navigate to “HelloWorld” Directory using cd command

and run cmd dotnet new console for creating console application

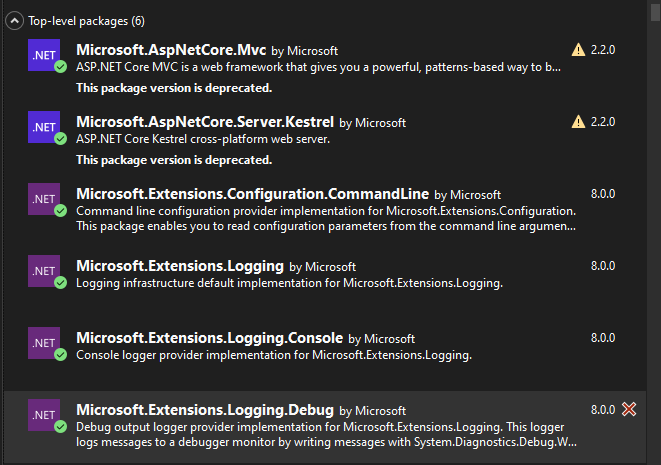
and try running the demo app using dotnet run command

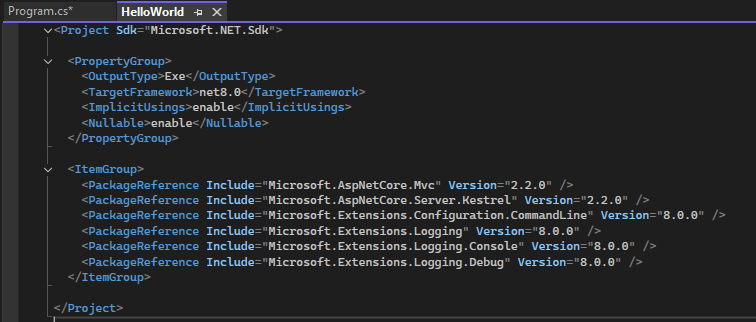


**STEP 4:** Open the same project in Visual Studio 2022

**STEP 5:** Go to Dependencies → Manage NuGet Package → go to browse and install the below package with the latest stable version.







**STEP 6:** Write code in Program.cs file

**Program.cs**

using System;

**using Microsoft.AspNetCore.Builder;**

**using Microsoft.AspNetCore.Hosting;**

**using Microsoft.Extensions.Logging;**

**using Microsoft.Extensions.Configuration;**

**using Microsoft.AspNetCore.Http**;

namespace HelloWorld

{

internal class Program

{

public static void Main(string[] args)

{

**var config = new ConfigurationBuilder().AddCommandLine(args).Build();**

**IWebHost host = new WebHostBuilder().UseKestrel().UseStartup<Startup>().UseConfiguration(config).Build();**

**host.Run();**

}

}

**public class Startup**

**{**

**public Startup(IHostingEnvironment env)**

**{**

**}**

**public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory logger)**

**{**

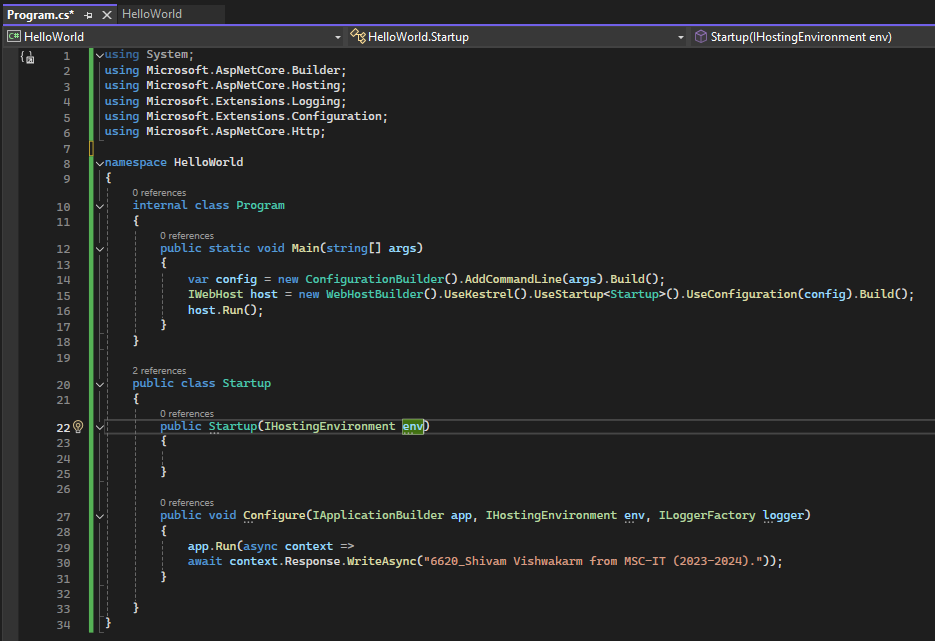
**app.Run(async context =>**

**await context.Response.WriteAsync("6620\_Shivam Vishwakarma from MSC-IT (2023-2024)."));**

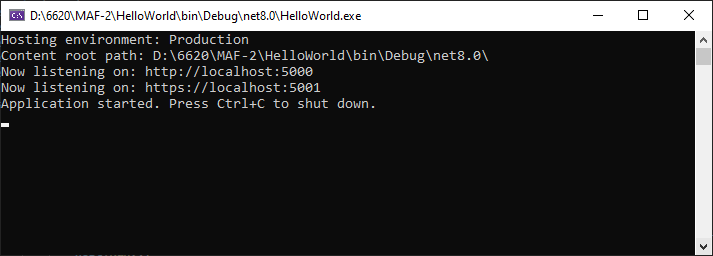
**}**

**}**

}

****

**STEP 7:**  Run the project

****

and open the project on browse on local host

****

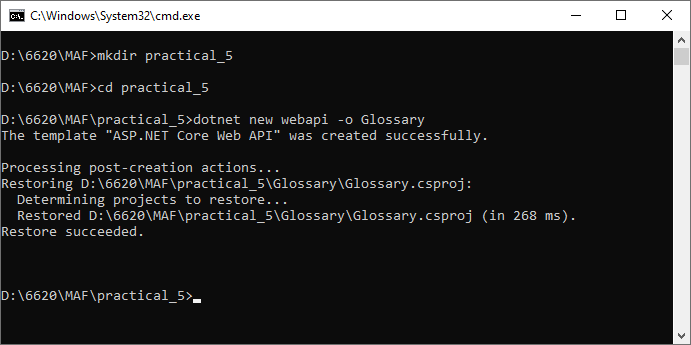
#### **Practical 5 - Building ASP.Net core REST API**

Jul 29, 2024

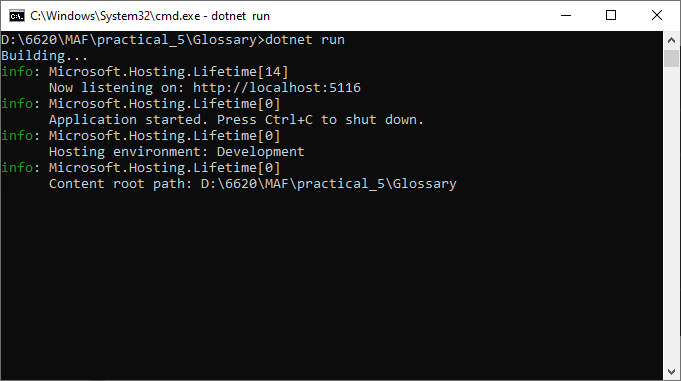
**Step 1**: Create a webAPI

Open command prompt and give the command

dotnet new webapi -o Glossary



Now enter into the glossary folder and then run the project



**Step 2**: Open an other command prompt & give curl command to view the output

curl --insecure http://localhost:5116/weatherforecast



**Step 3**: Delete the weatherforecast.cs from the Glossary Folder i.e root folder and also from the Controller Folder.

**Step 4**: Create a class file in the Glossary folder named “GlossaryItem.cs”

****

namespace Glossary

{

public class GlossaryItem

{

**public string Term { get; set; }**

**public string Definition { get; set; }**

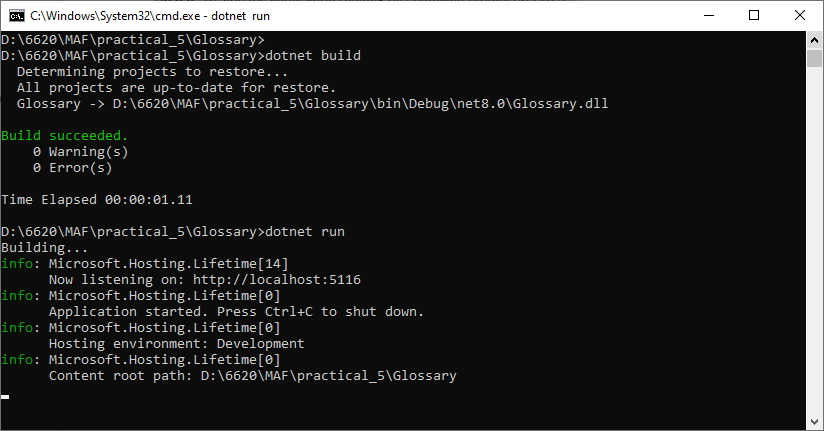
}

}

**Step 5:** Create a class file in the Controllers folder named “GlossaryController.cs”

**Step 6**: To stop the application running on command prompt do Ctrl+c

**Step 7:** Now restore, build and then run the program

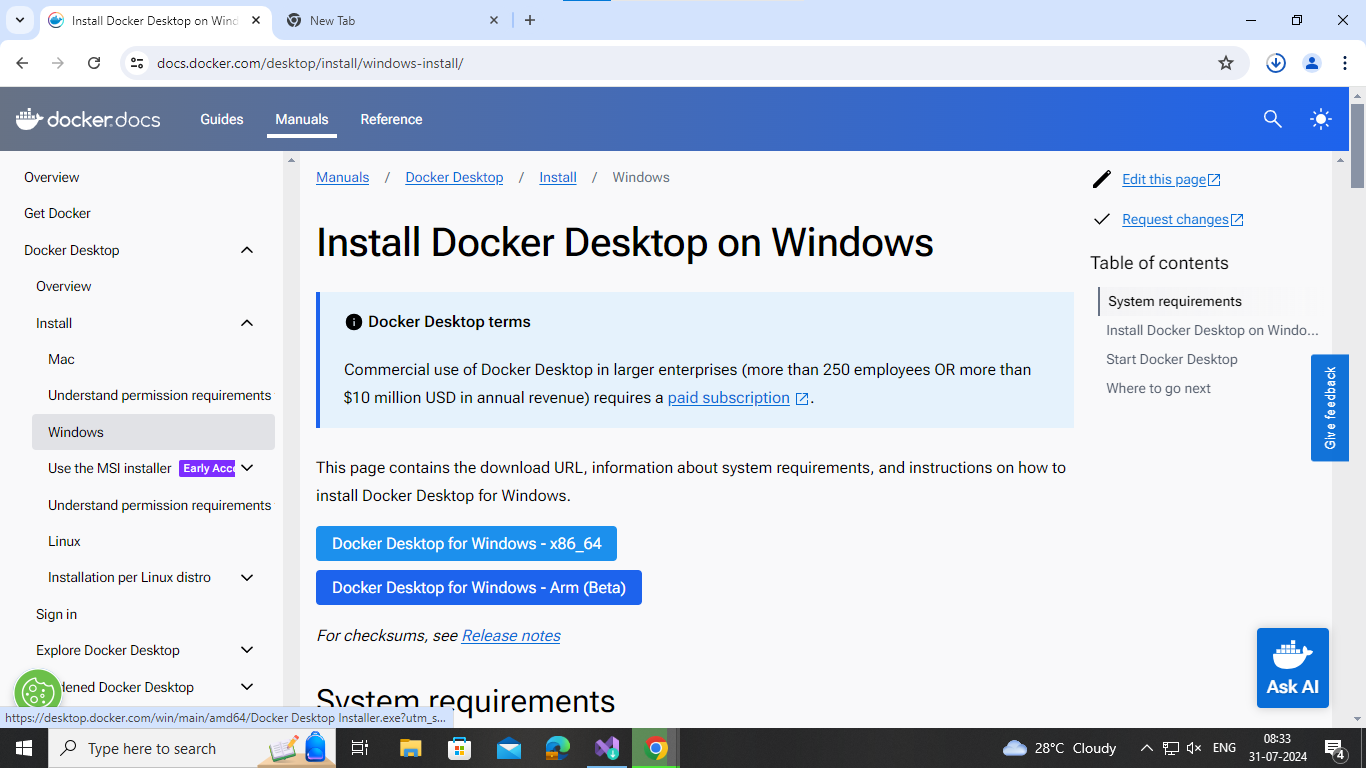


#### 

#### Jul 31, 2024 - Docker

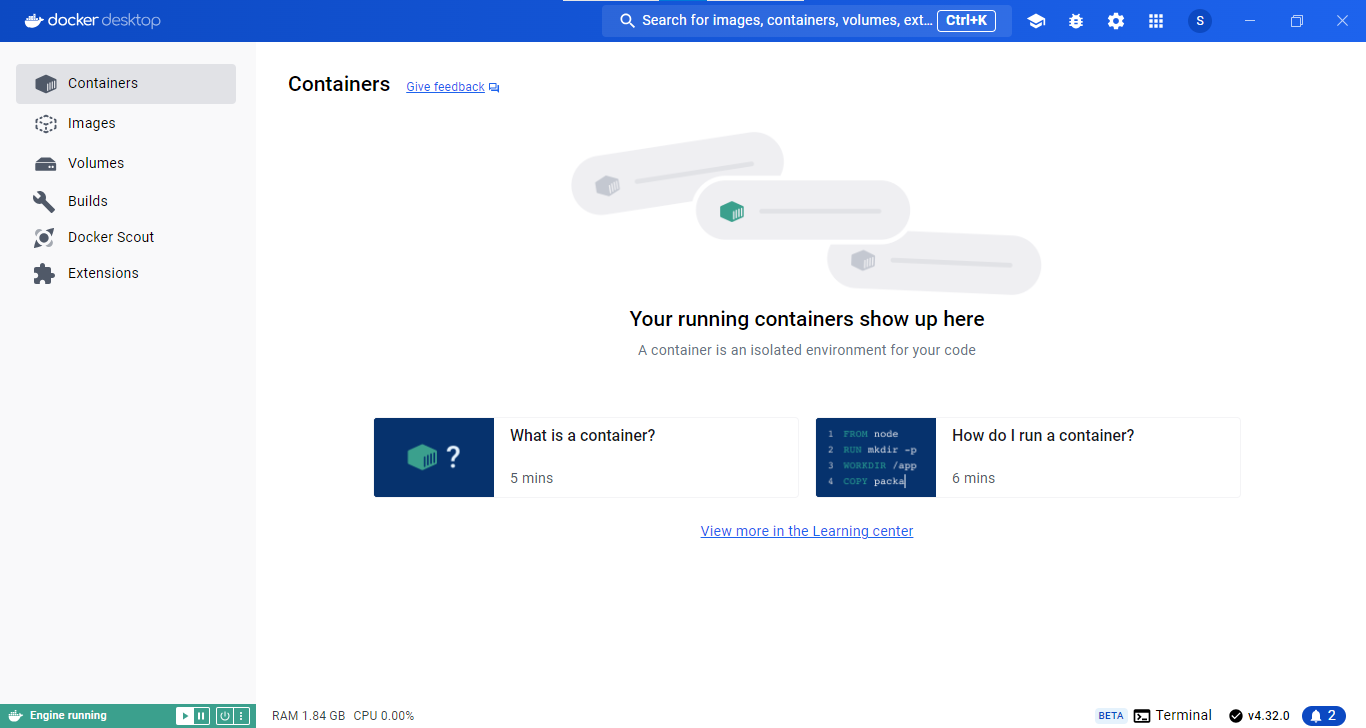
**DOCKER Practical (Push and Pull)**

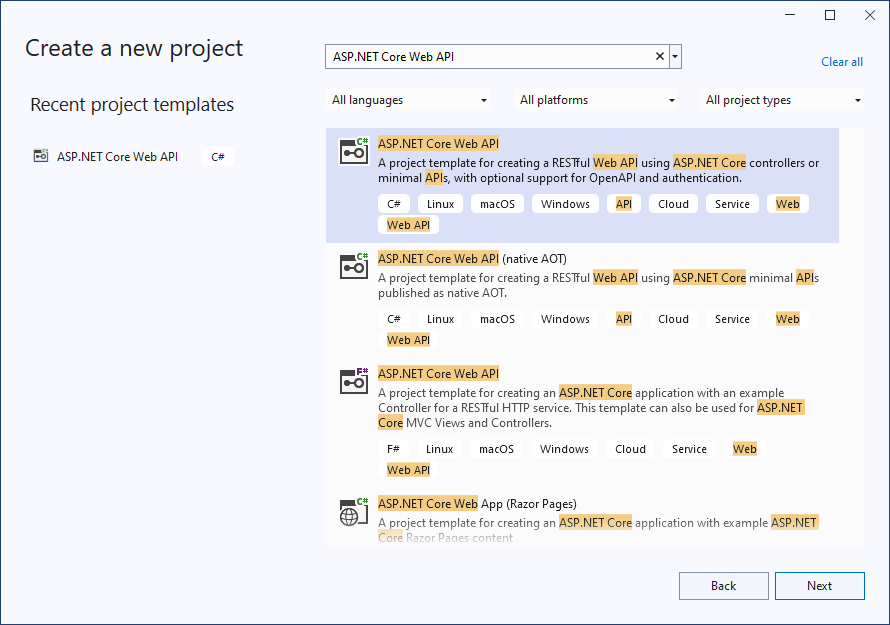
**STEP 1:** [Install Docker Desktop](https://docs.docker.com/desktop/install/windows-install/)

****

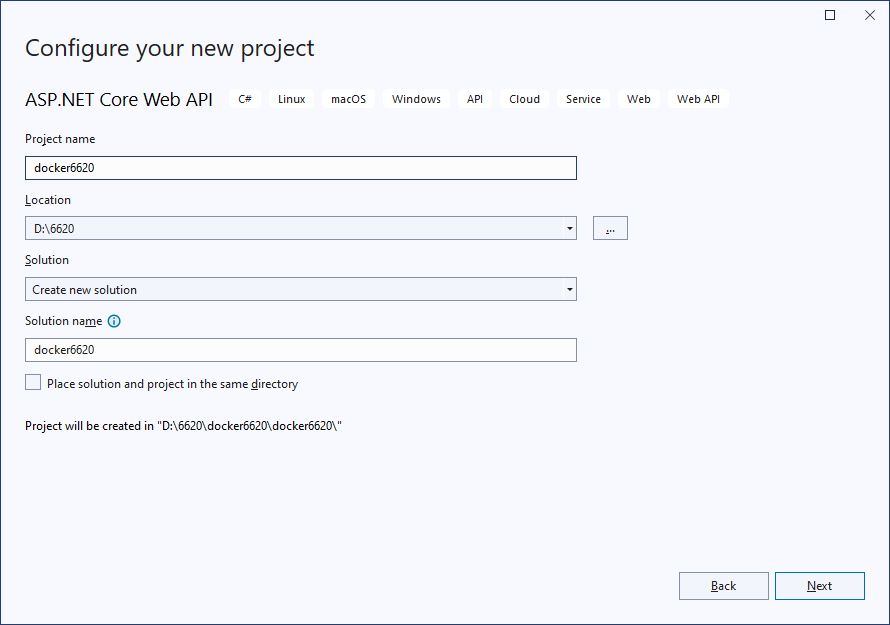
**STEP 2:** Run the exe with recommended settings and create a docker account by signing up and login using that same account.

**Note :** You should be logged in Docker Desktop with Engine running (Like Below)

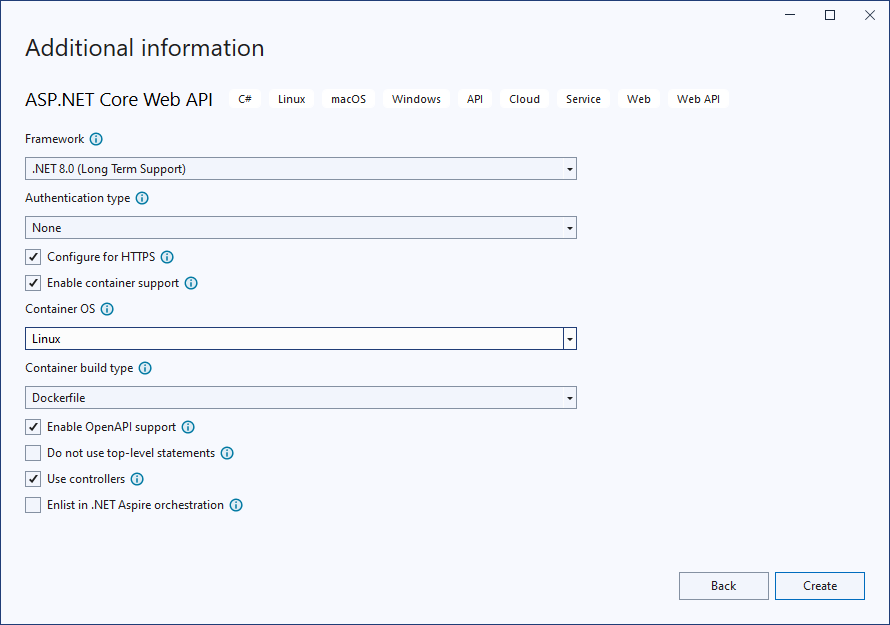
****

**STEP 3:** Create “ASP.NET Core Web API” project

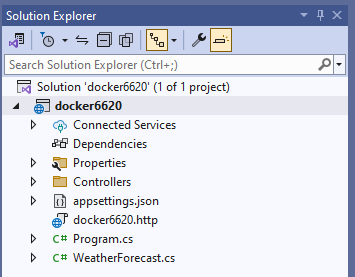
**STEP 4:** name the project as “docker6620” and click on Next



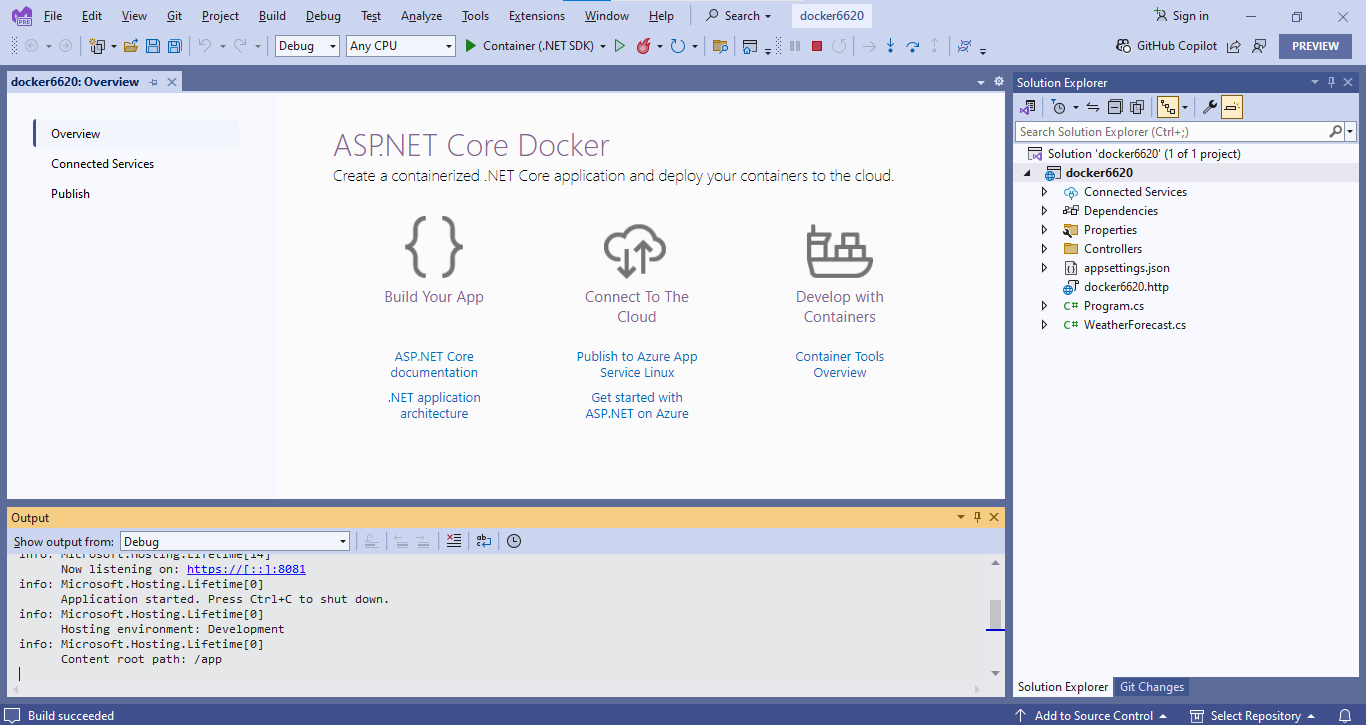
**STEP 5:** Check the “Enable container support” → Select Linux from “Container OS” Dropdown → Select Dockerfile from “Container build type” dropdown



Project structure



**STEP 6:** Run the project using



**STEP 7:** In Docker Desktop you also can see the project running on Docker in Container in and images also.

**STEP 8:** Open Command Prompt and run below following commands

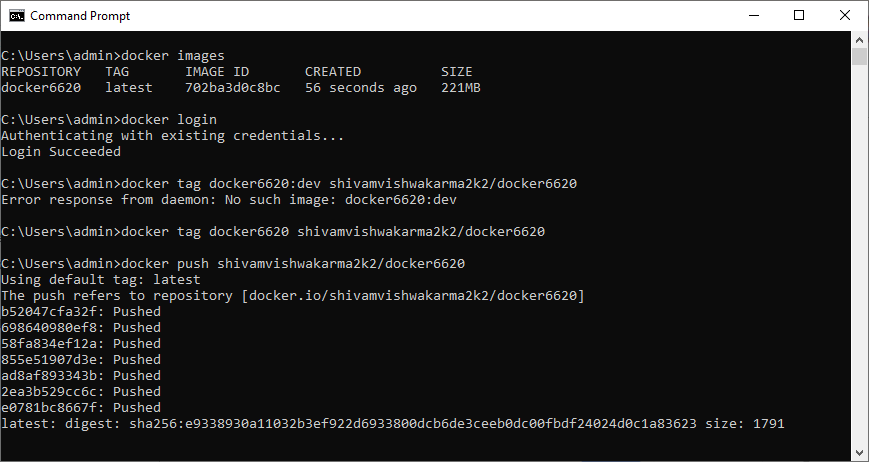
* **docker images**
* **docker login**
* **docker tag <Project Name>:dev <Docker User ID>/<Project name>**

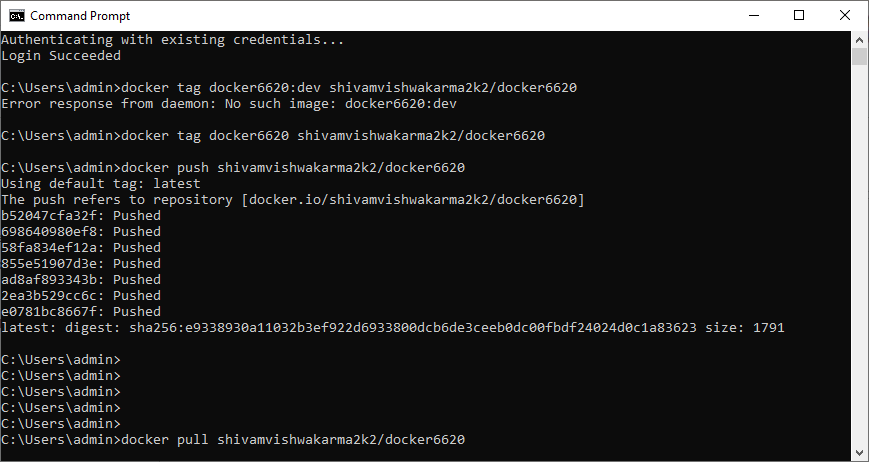
(if getting error in this try removing :dev from command)

* **docker push <DockerUser ID>/<Project name>**

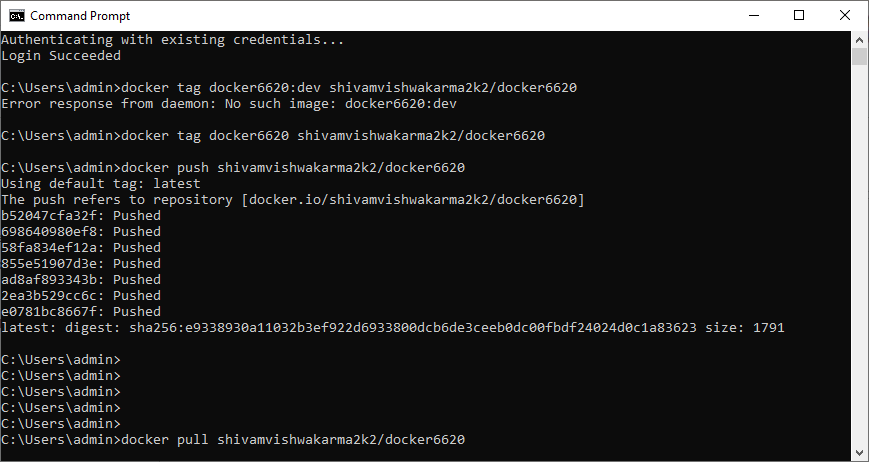
(login to docker hub in browser and you can see your project docker image on the browser)

* **docker pull <DockerUser ID>/<Project name>**

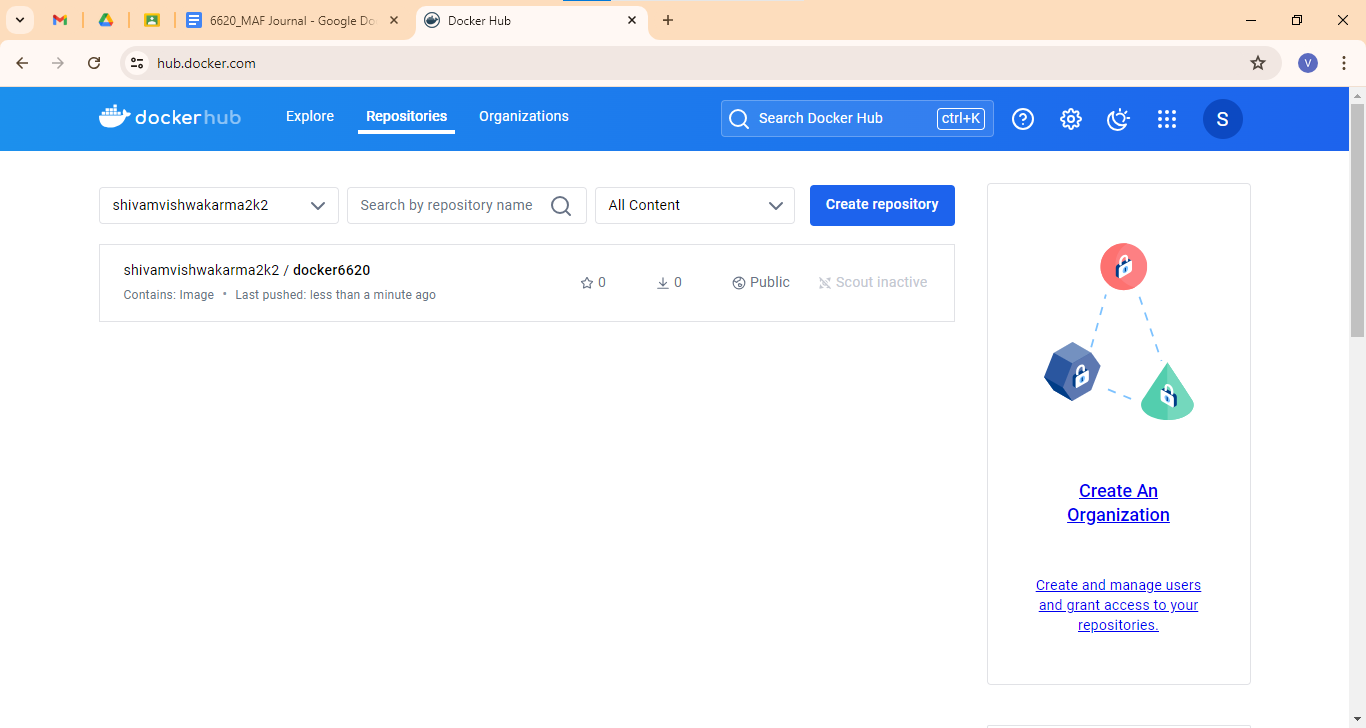


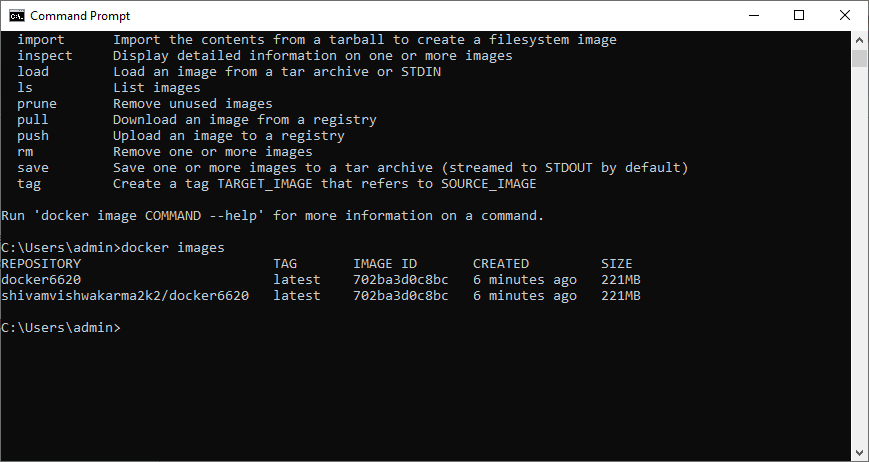


Push your Docker directory into the Docker desktop Wait for the command to be executed.

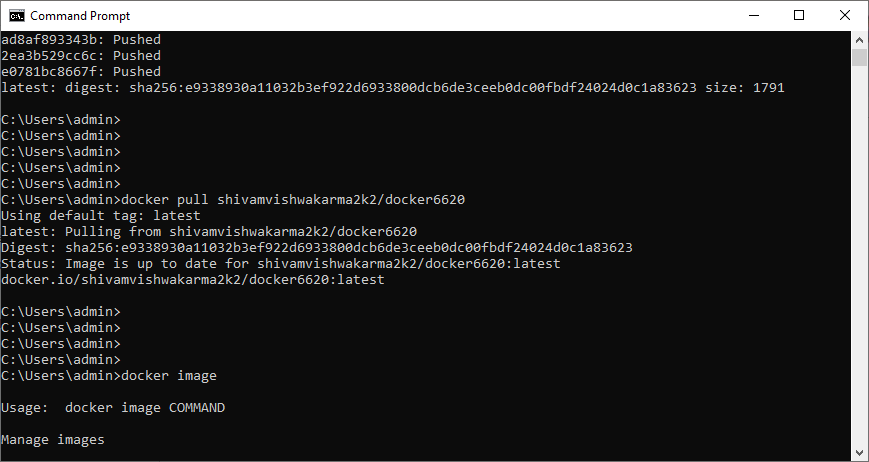


After that you can view your repository in the Repositories section in DocketHub website.

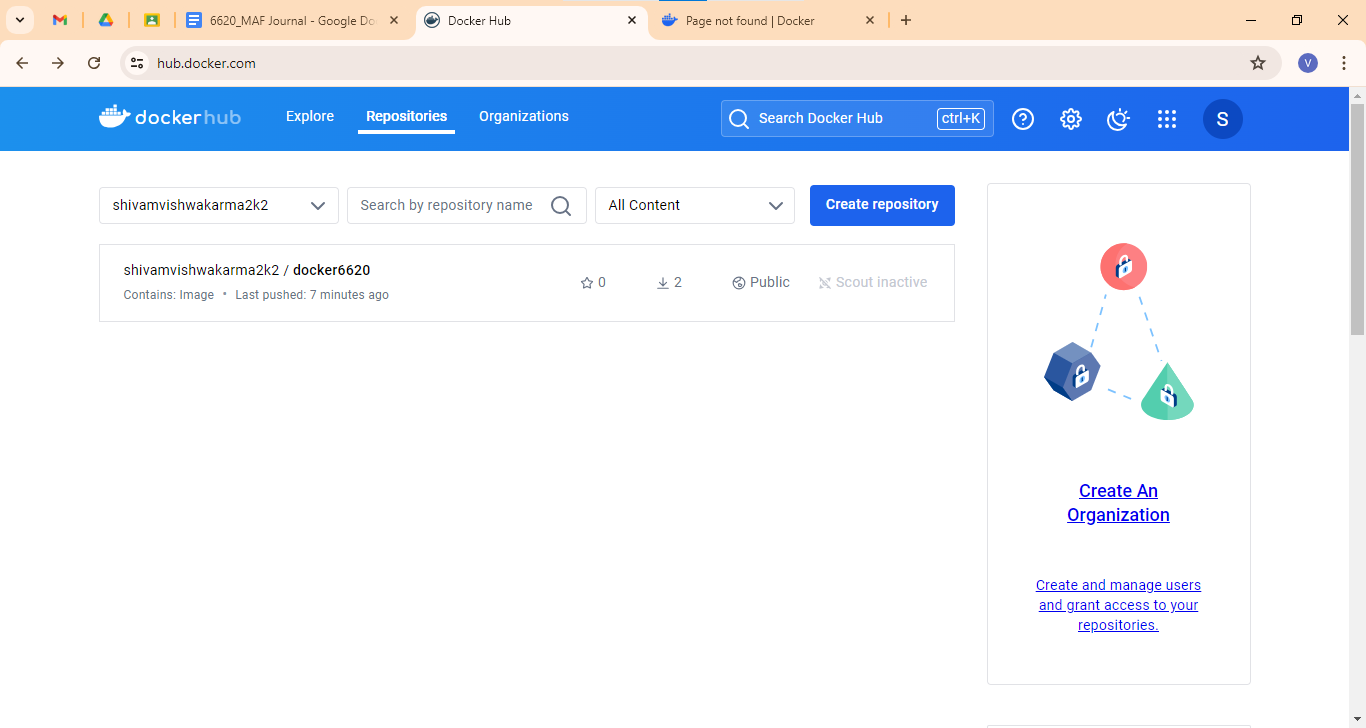




You can also pull the project from docker hub to your own id or another id as well



You can see the numbers of pull on docker hub



**docker run -p 8080:8080 <Docker User ID>/<Project name> (Not used in this practical)**

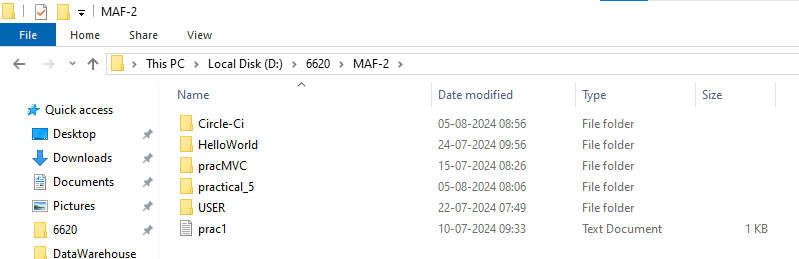
#### Aug 5, 2024 - Circle CI

Requirements

Git bash

Github Desktop

**STEP 1:** Create a folder named Circle-Ci in your MAF Practical Di



**STEP 2:** Create a main.py file in circle-ci folder and write below following code

**main.py**

def to\_upper(name):

return name.upper()

def say\_hello(name):

print(f'Hello, {name}')

if \_\_name\_\_ == '\_\_main\_\_':

name = 'trainwithShivam'

say\_hello(name)

up = to\_upper(name)

print(up)

**STEP 3:** Create a another file named “test.py” in root dir “Circle-Ci” and write below following code

**test.py**

import unittest

from main import to\_upper

class MyTestCase(unittest.TestCase):

def test\_to\_upper(self):

name = 'Shivam'

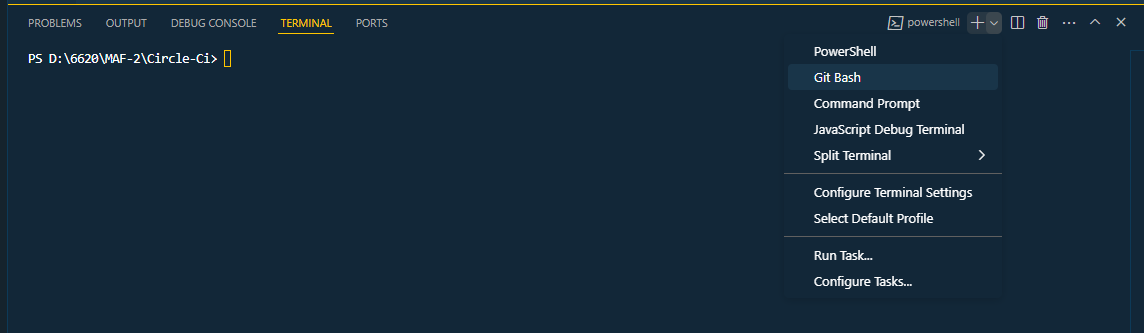
upper\_name = to\_upper(name)

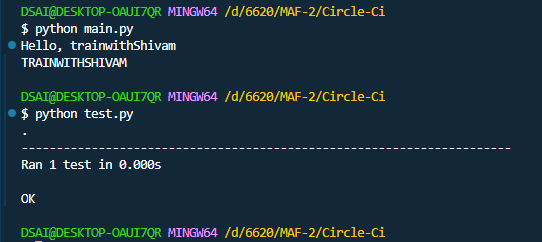
self.assertEqual(upper\_name, "SHIVAM")

if \_\_name\_\_ == '\_\_main\_\_':

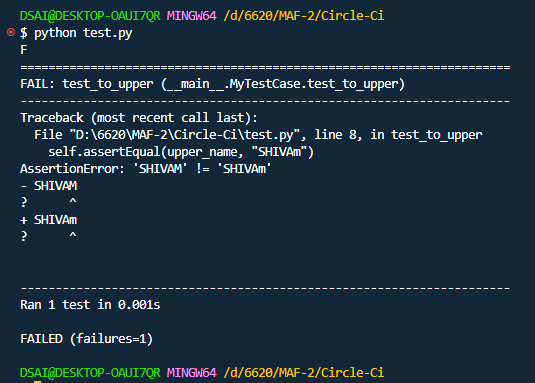
unittest.main()

Open terminal in git bash

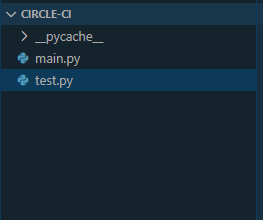




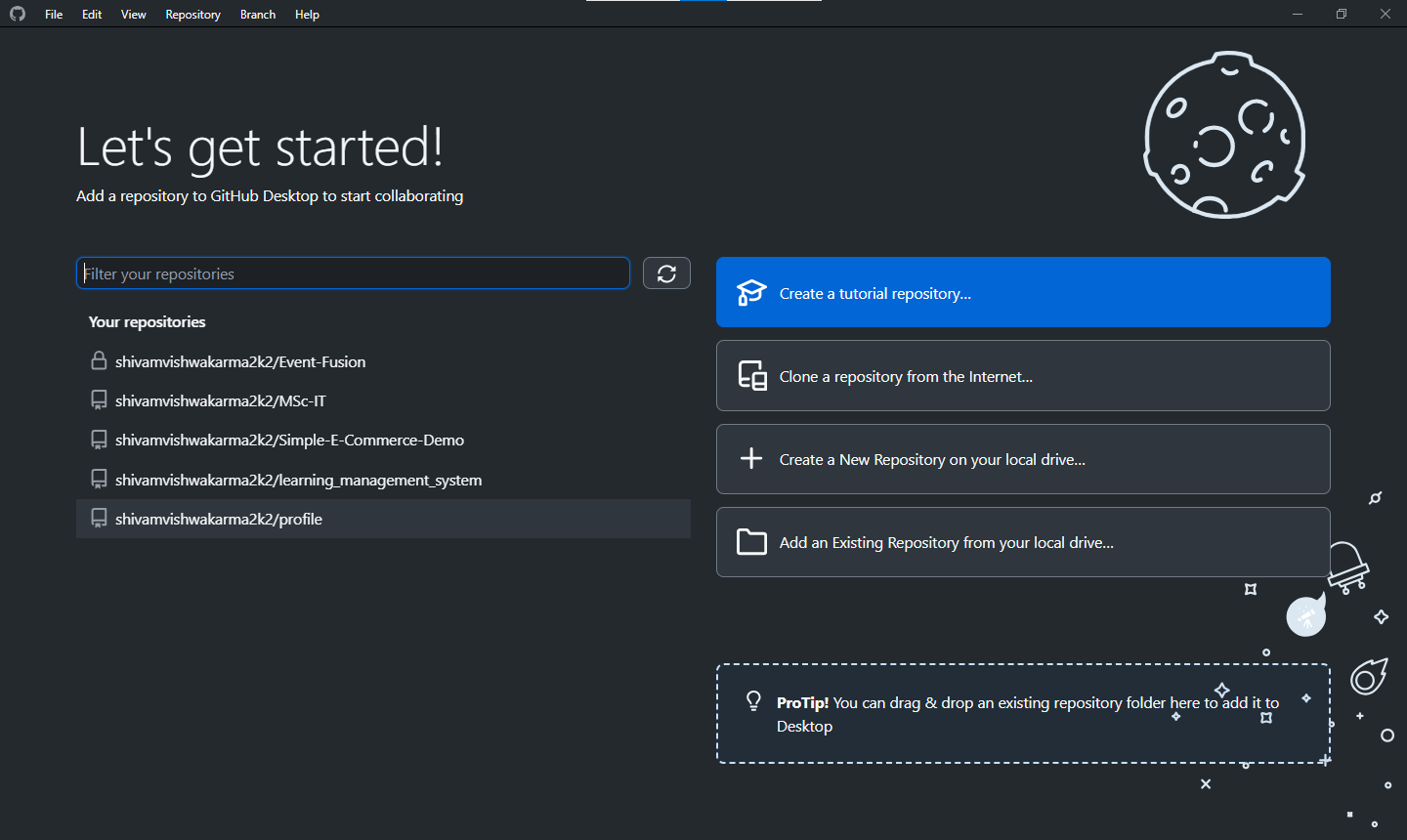
If changes any single letter will get error



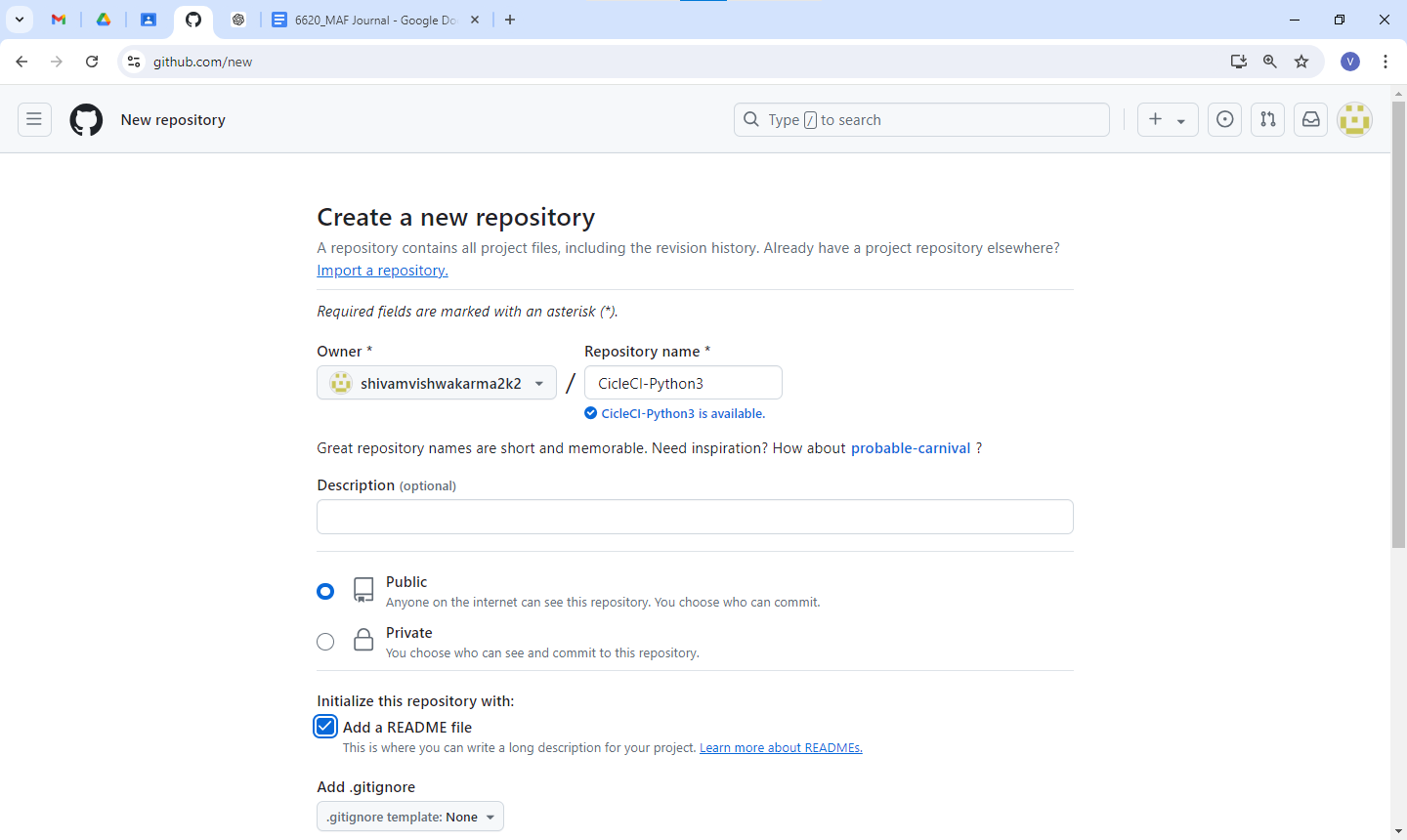
Folder structures of now

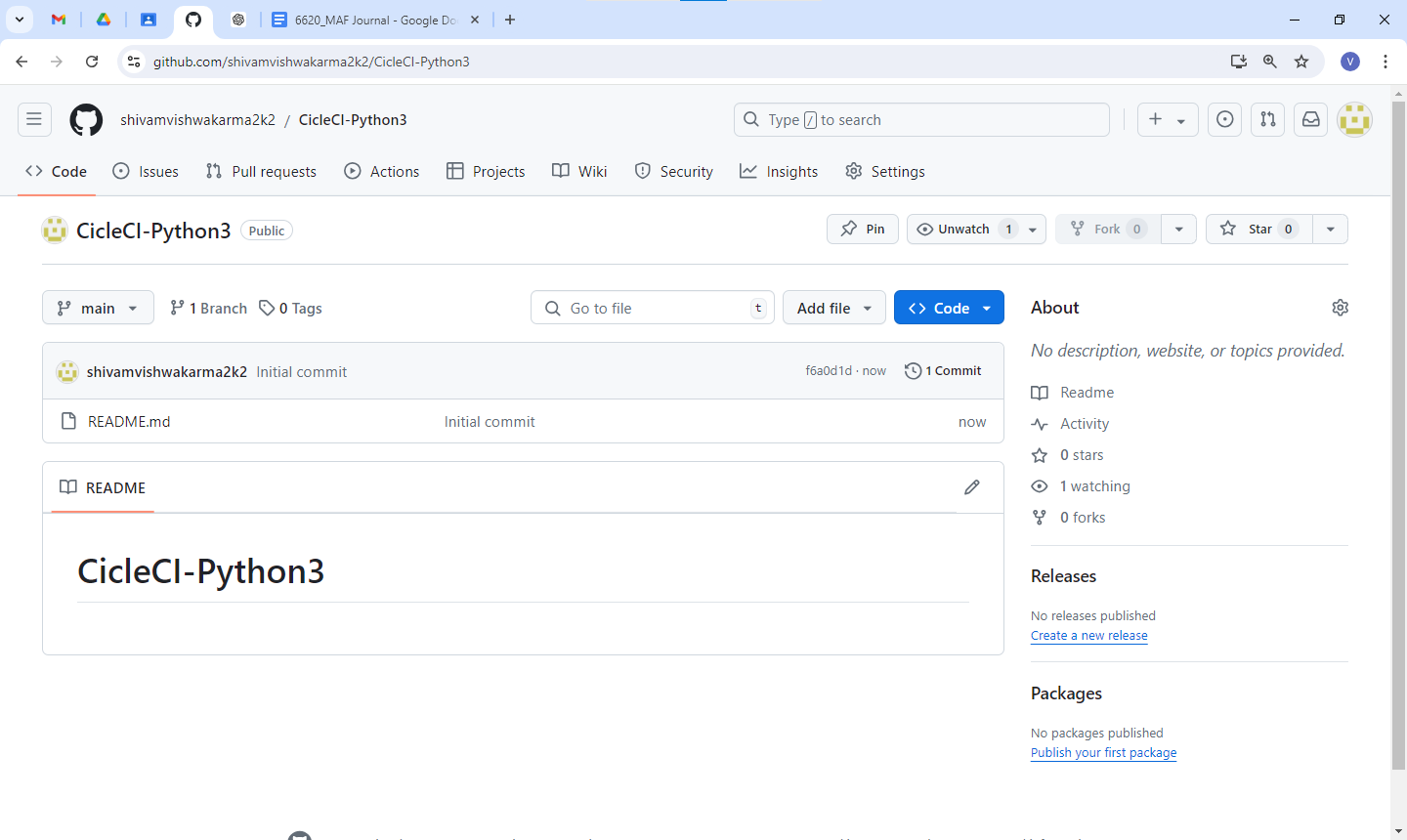


Now open github desktop and login using your github account

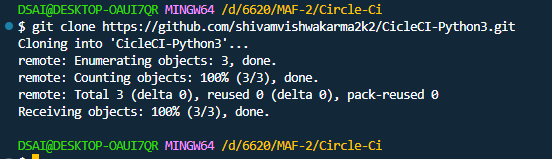


Login to github.com and creta anew respo

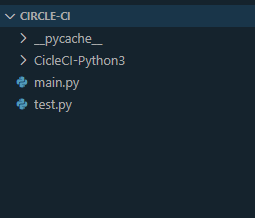




<https://github.com/shivamvishwakarma2k2/CicleCI-Python3.git>

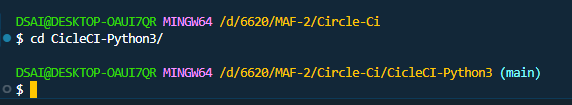


Folder structure after git clone



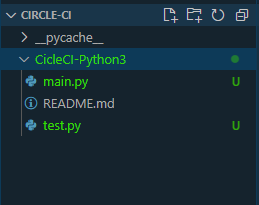
Move to CicleCI-Python3 dir

cd CicleCI-Python3/



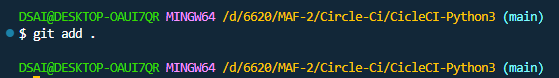
Move main.py and test.py to dir CicleCI-Python3

Folder structure after it

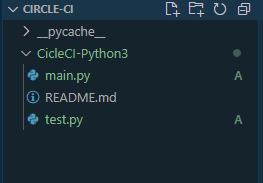


Now type command to push main.py and test.py to github repo using

git add .

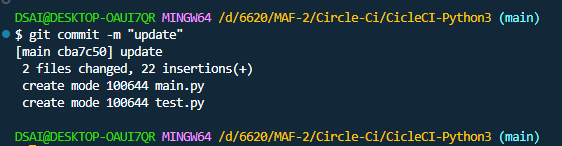


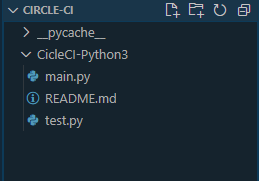
Folder structure



Now commit to repo using command

Git commit -m “message”

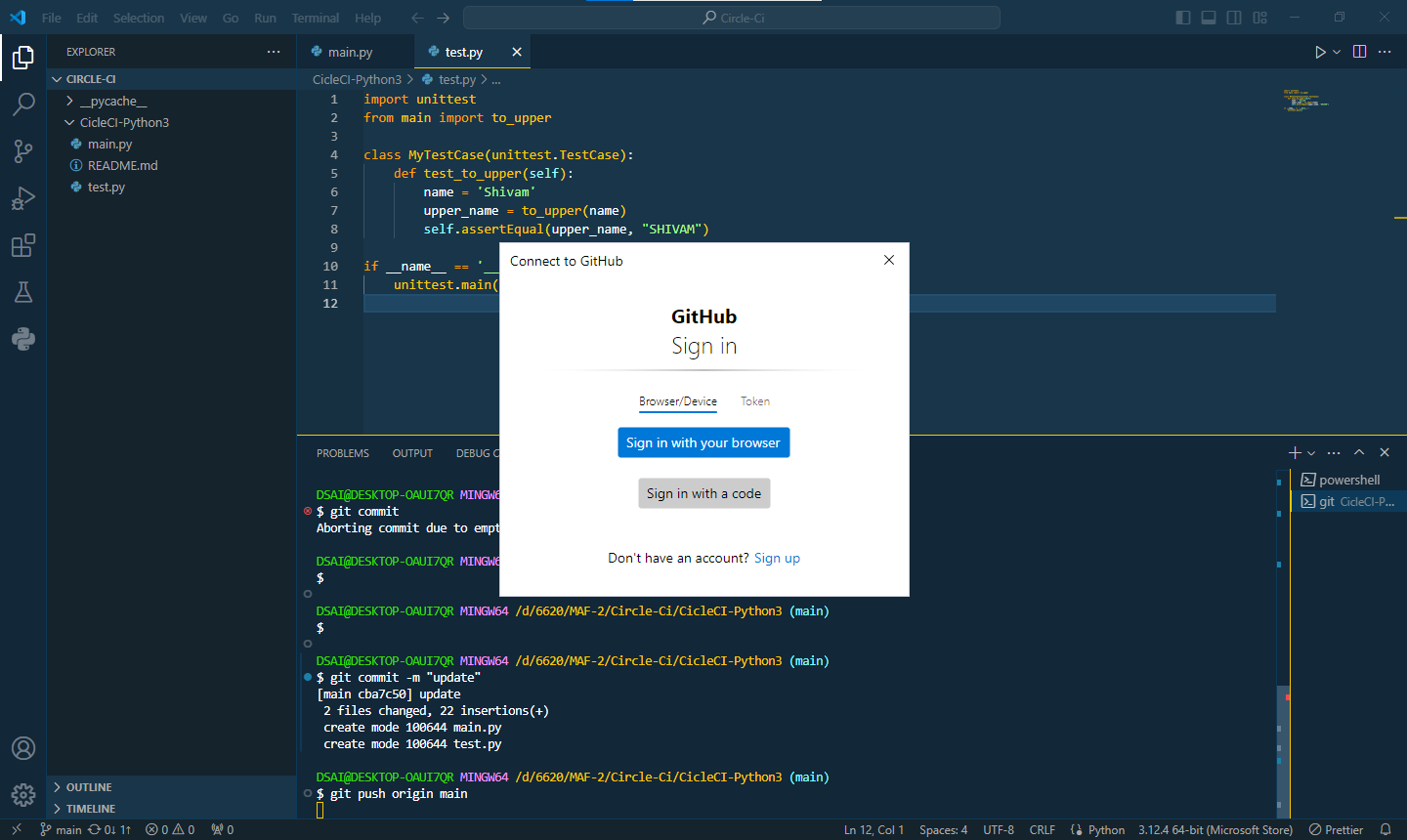




Now push code to repo using command

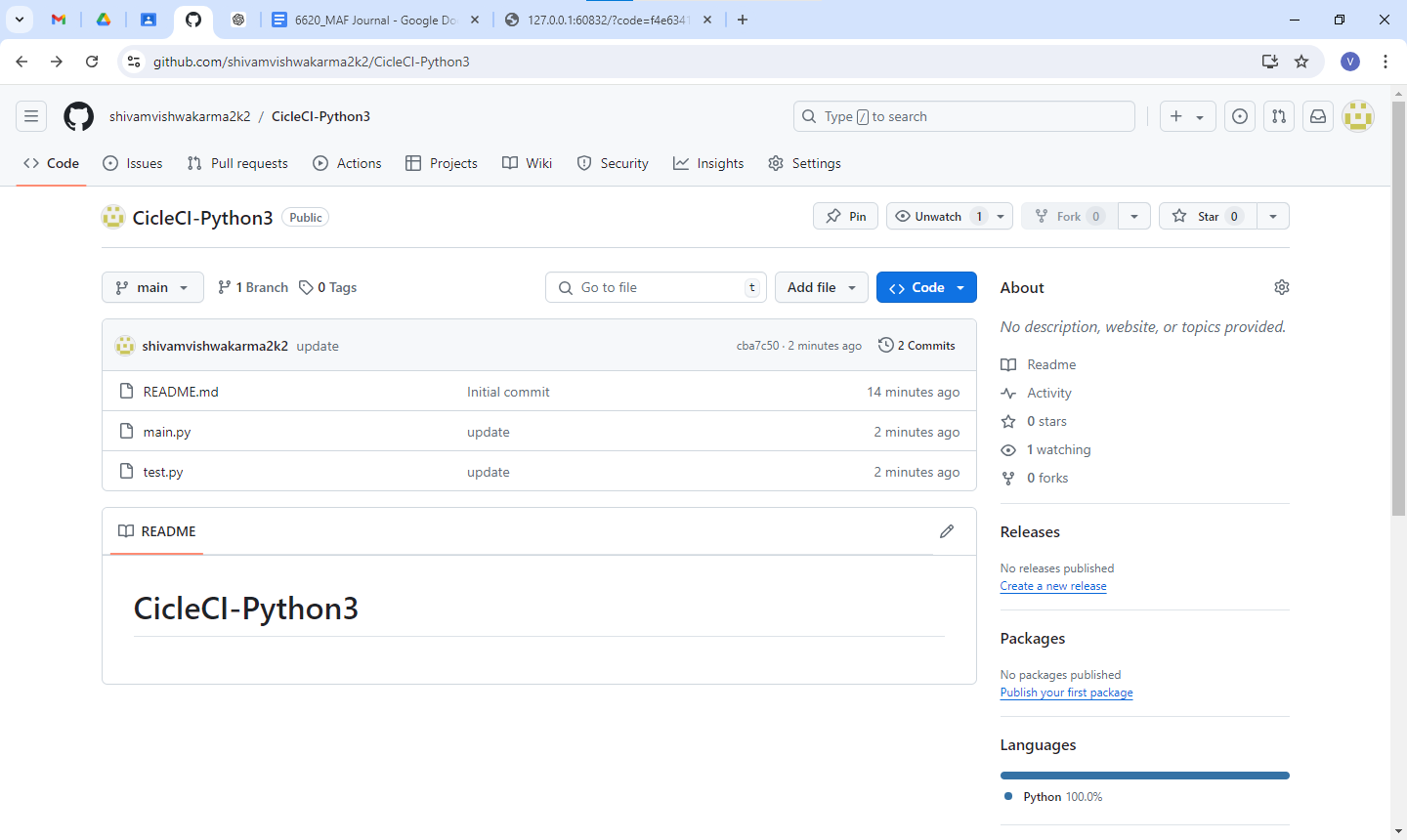
Git push origin main

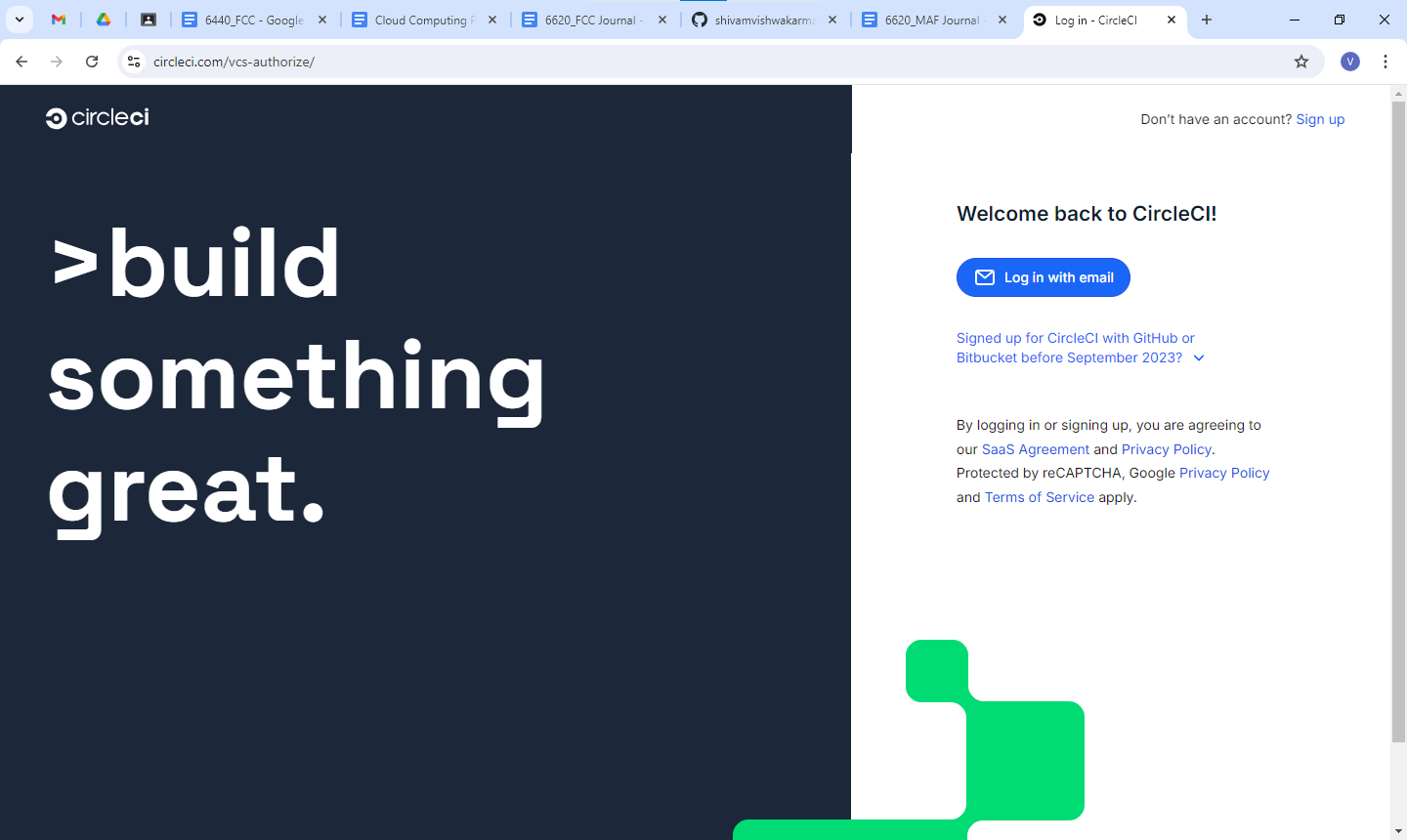
(ask for github login proceed)

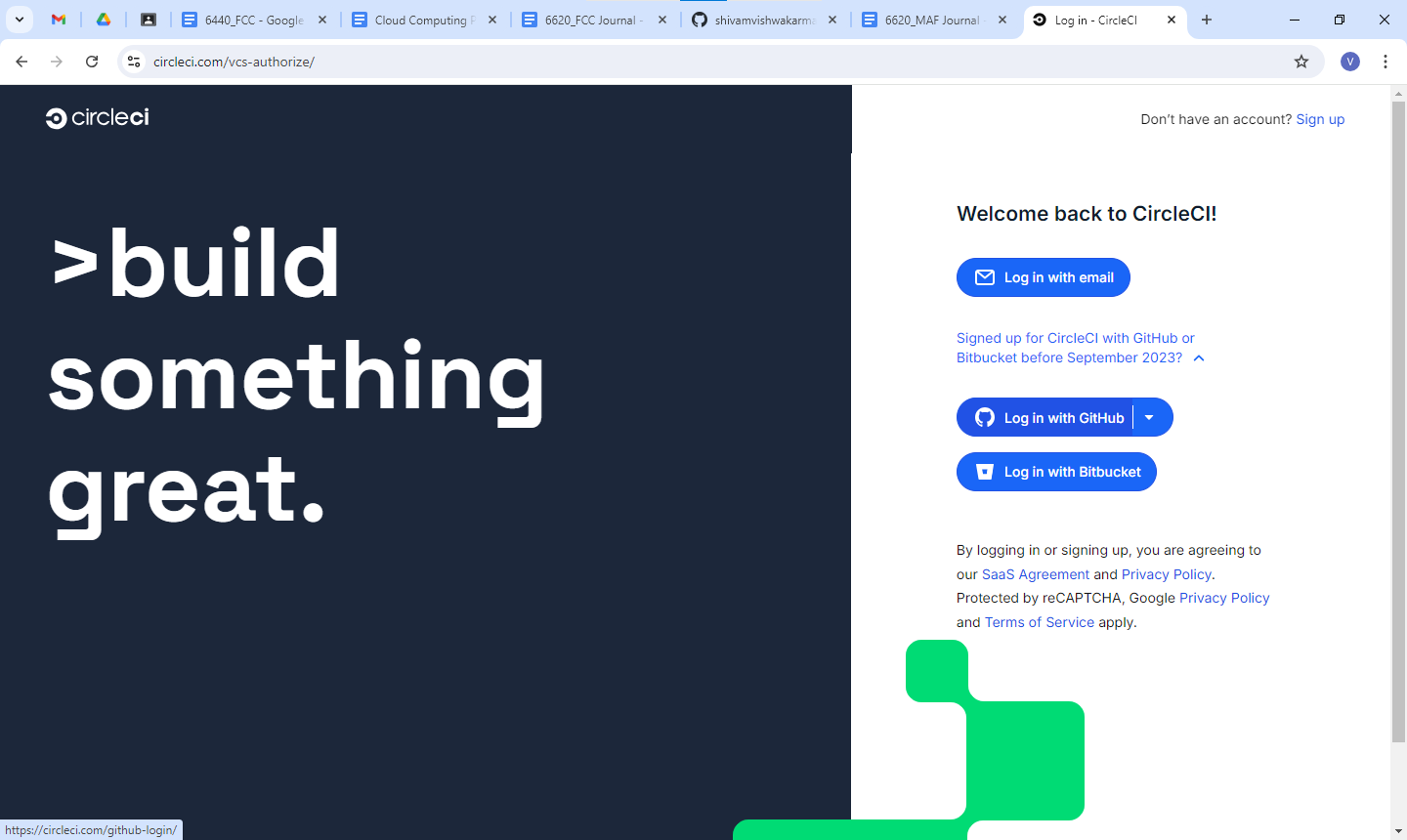


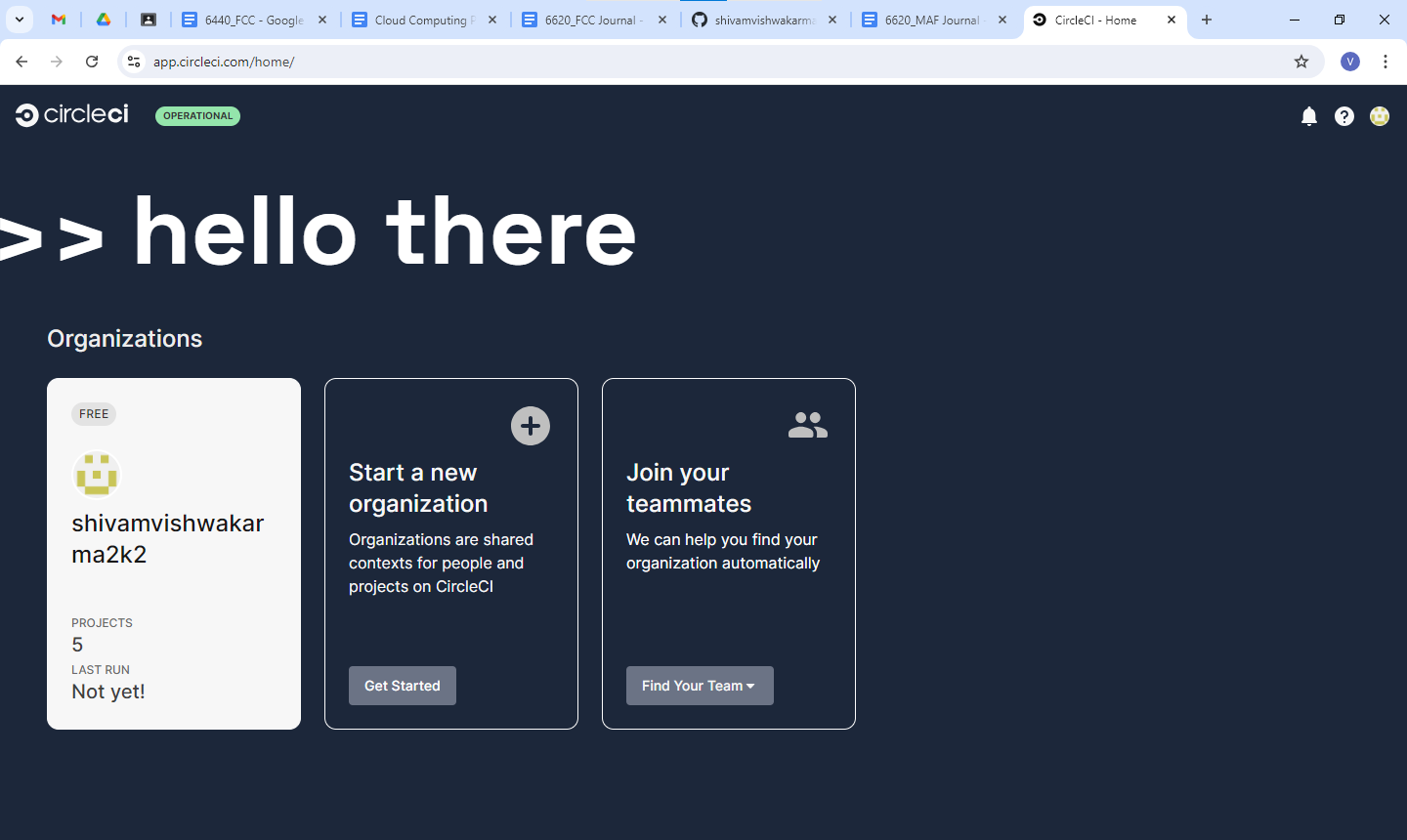


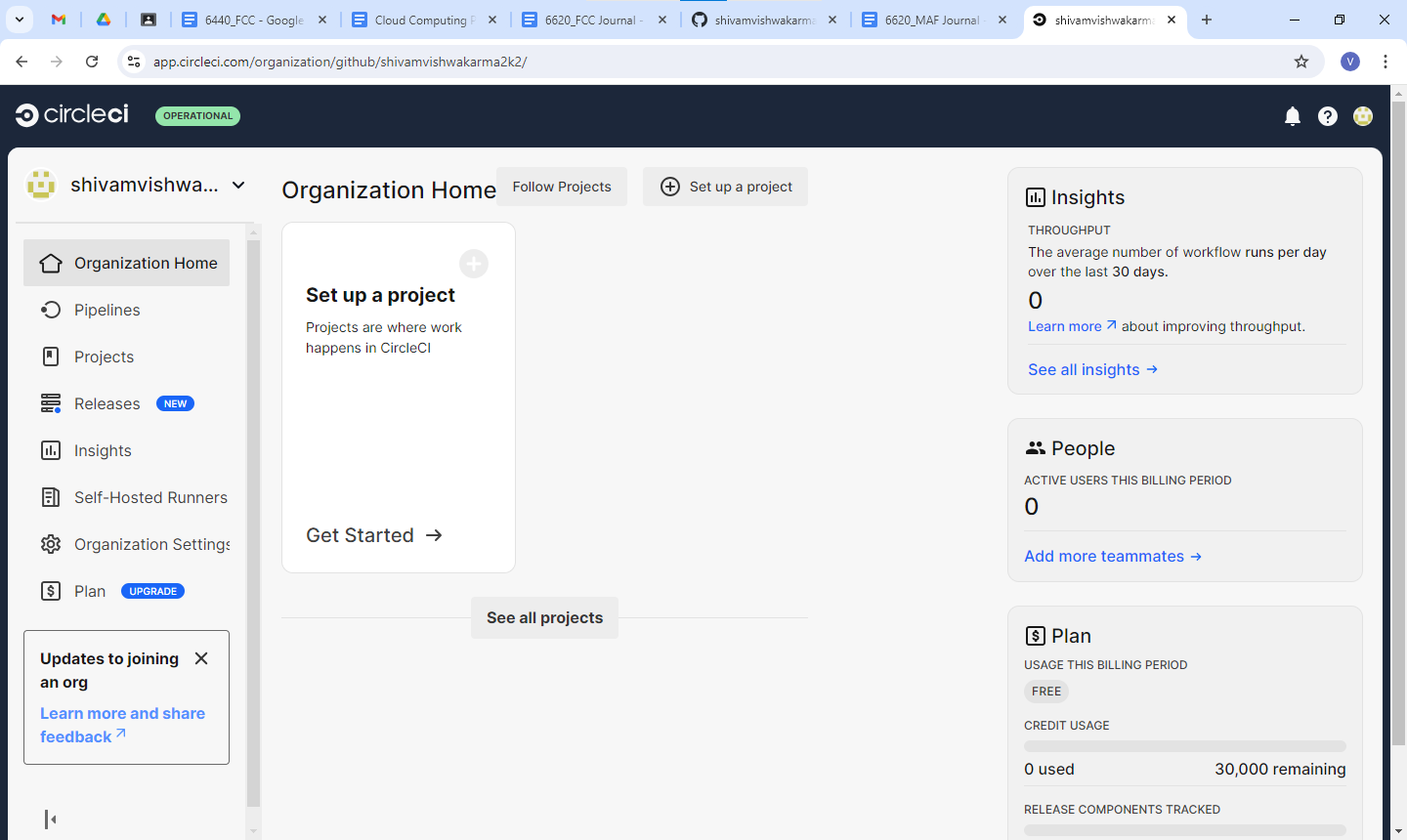
Now main.py and test.py are update to github as well

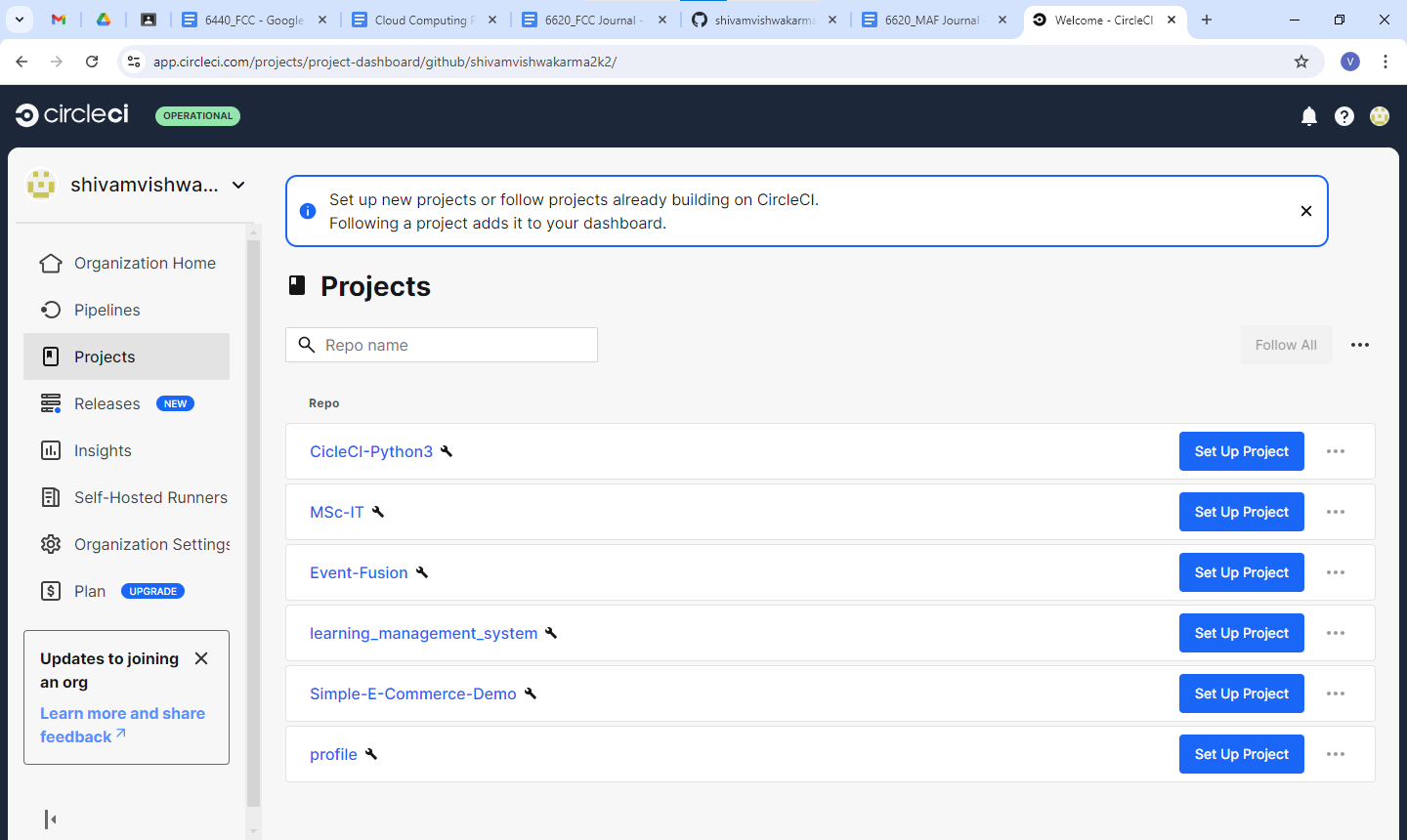


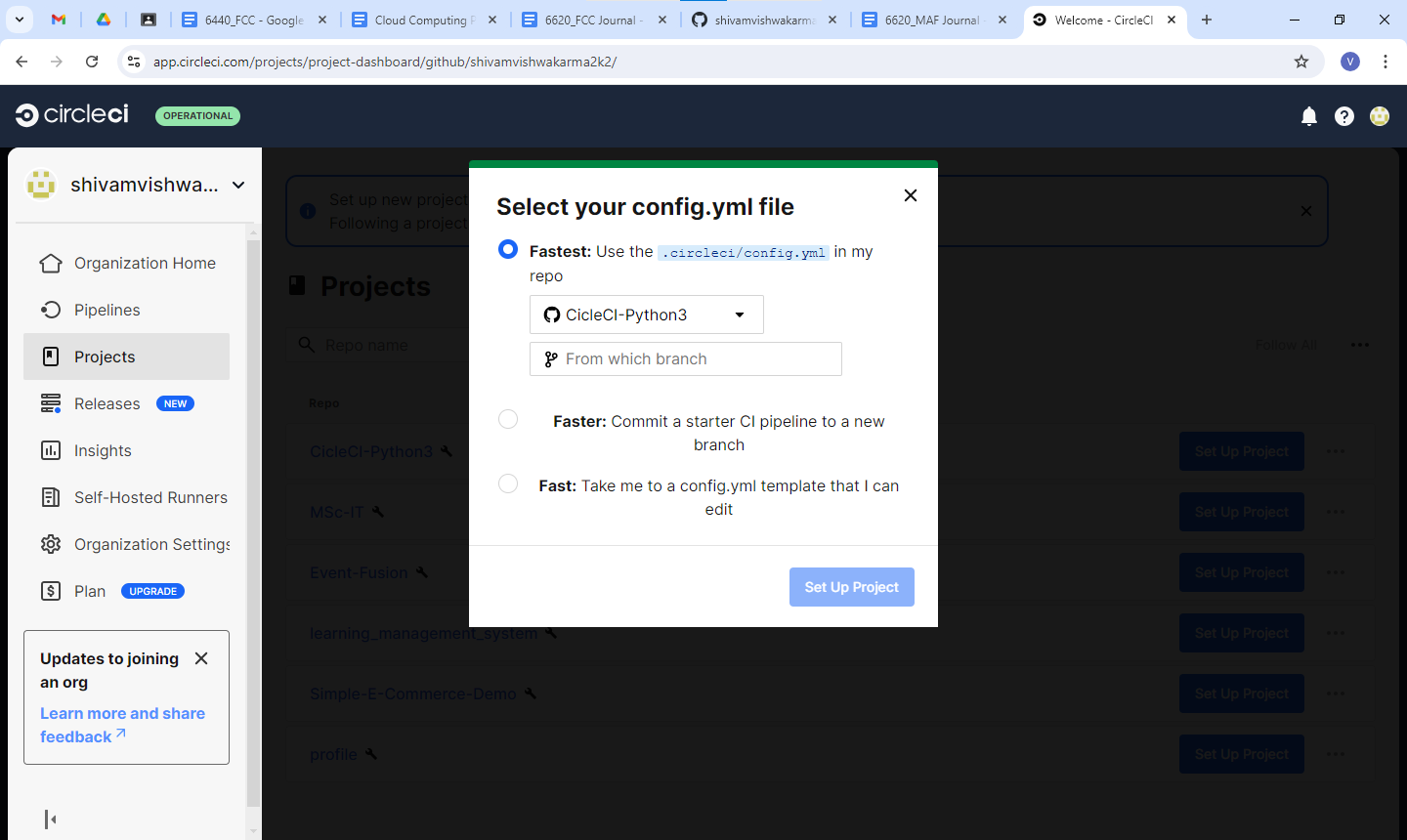




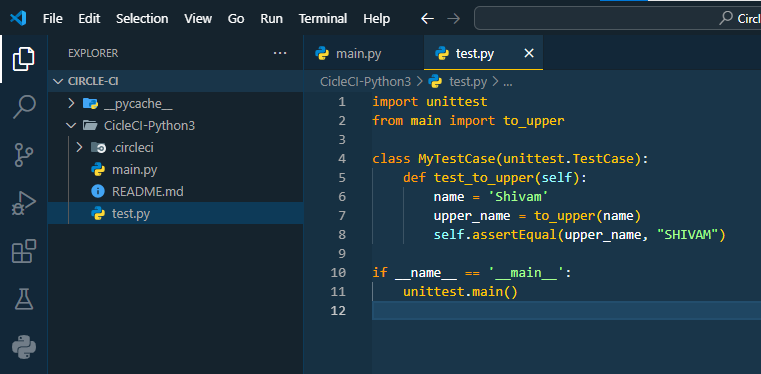


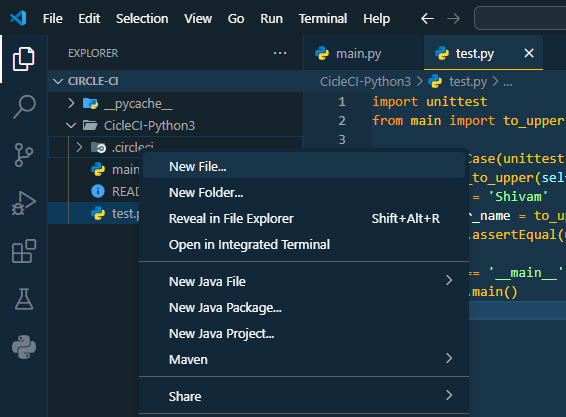


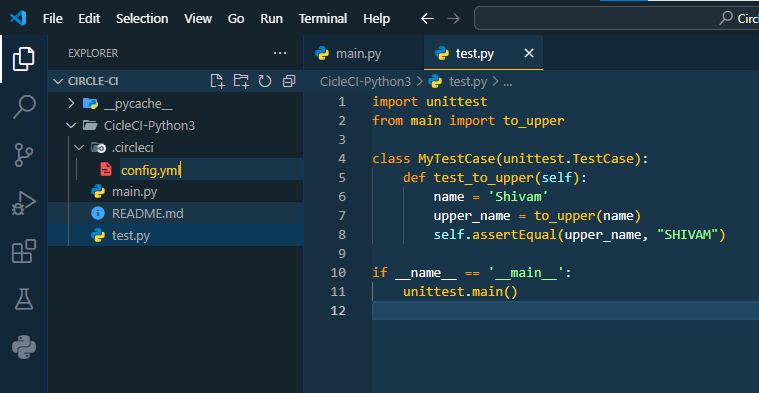




Create .circleci name folder in parent circle-CI-python3







**Write below following code in config.yml**

version: 2.1

jobs:

build:

docker:

- image: cimg/python:3.12

steps:

- checkout

- run:

name: Run build script

command: python main.py

test:

docker:

- image: cimg/python:3.12

steps:

- checkout

- run:

name: Run test script

command: python test.py

deploy:

docker:

- image: cimg/python:3.12

steps:

- run:

name: Deploy

command: echo 'practical done'

workflows:

build\_and\_test\_deploy:

jobs:

- build

- test:

requires:

- build

- deploy:

requires:

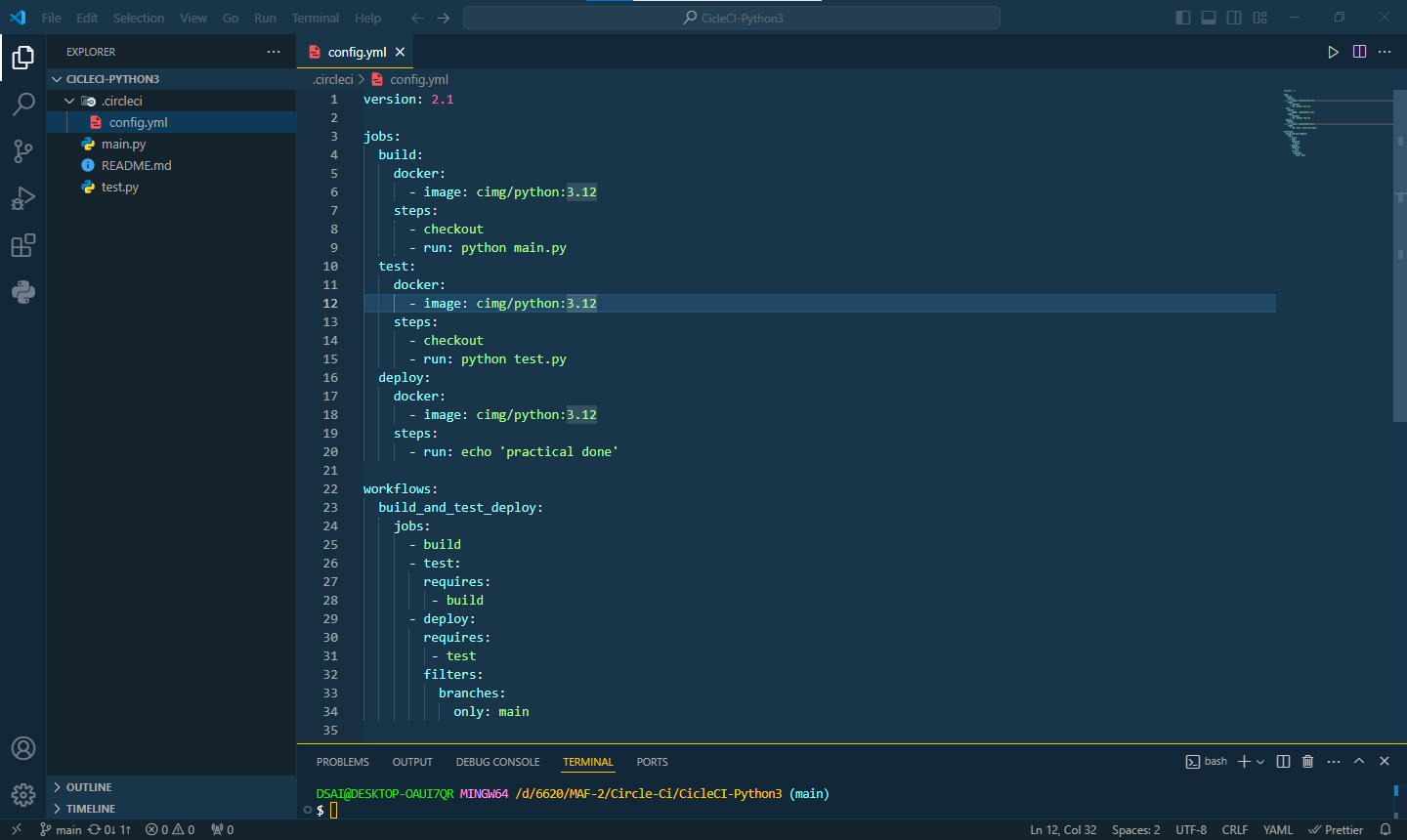
- test

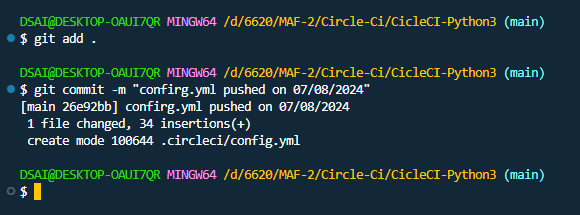
filters:

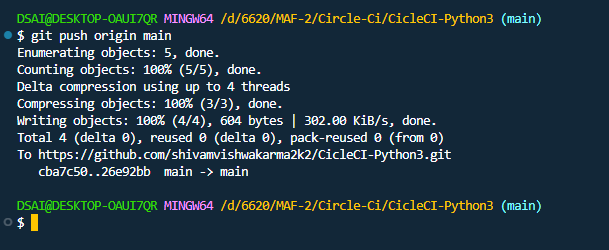
branches:

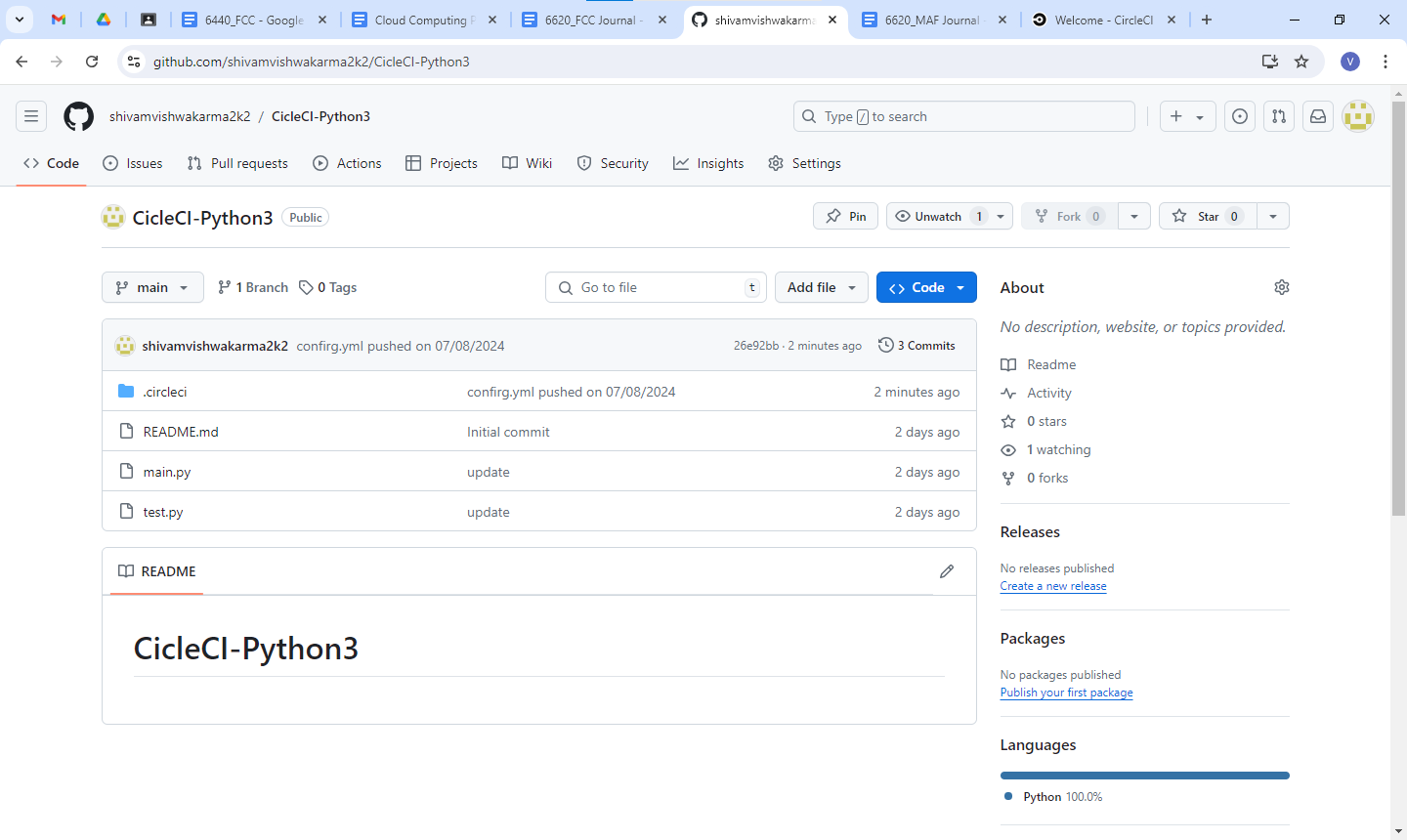
only:

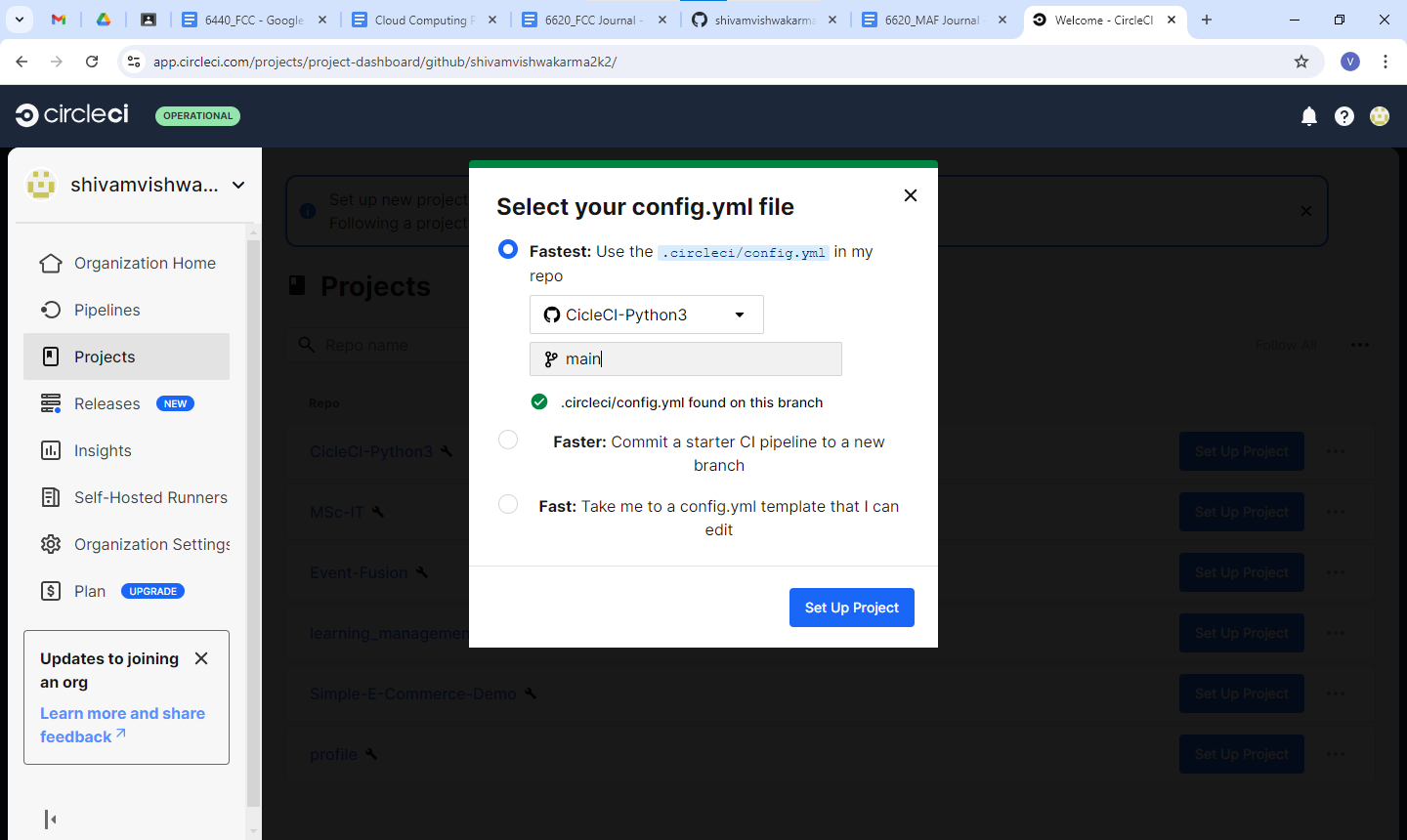
- main

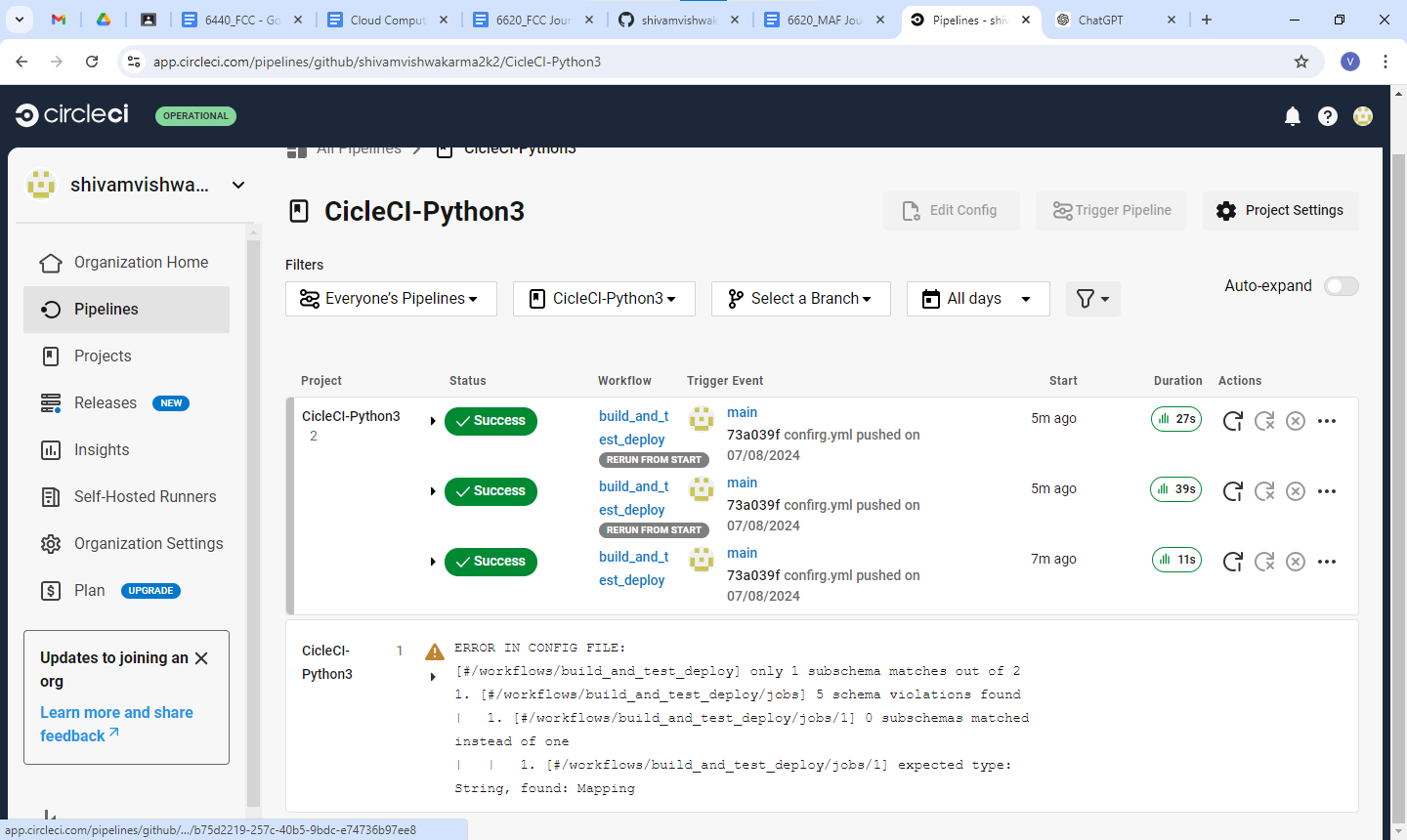








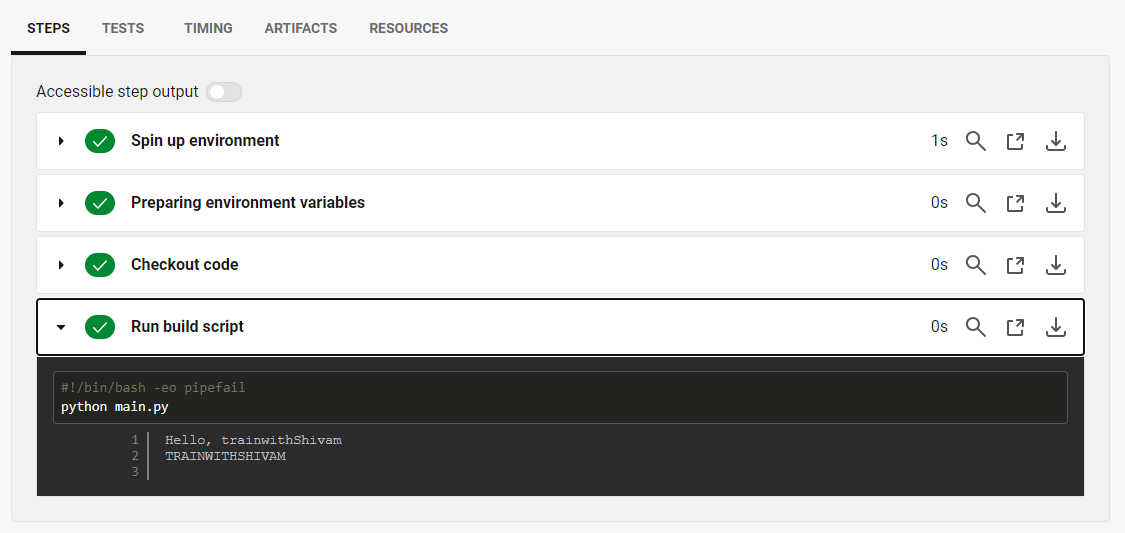




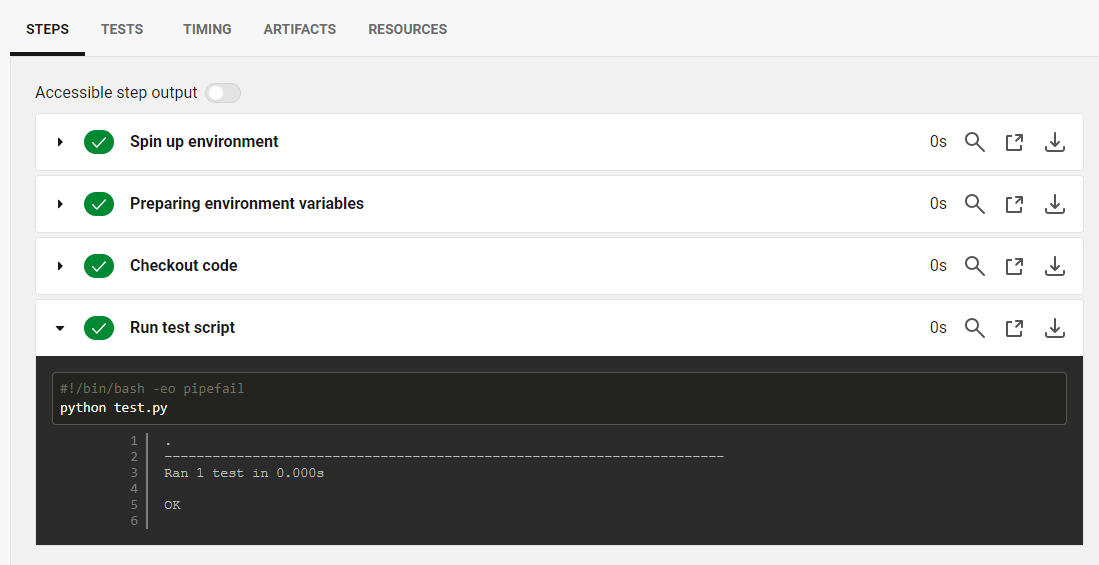
Expand the success node



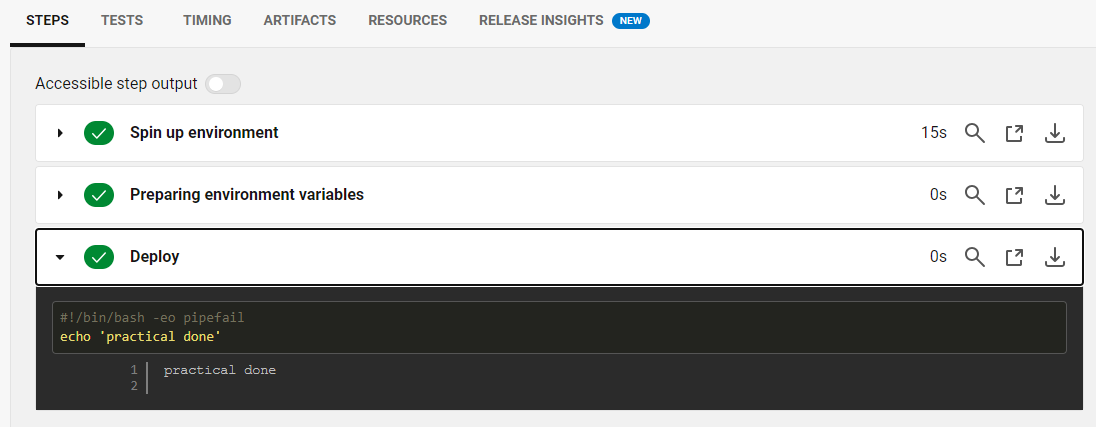
build



test

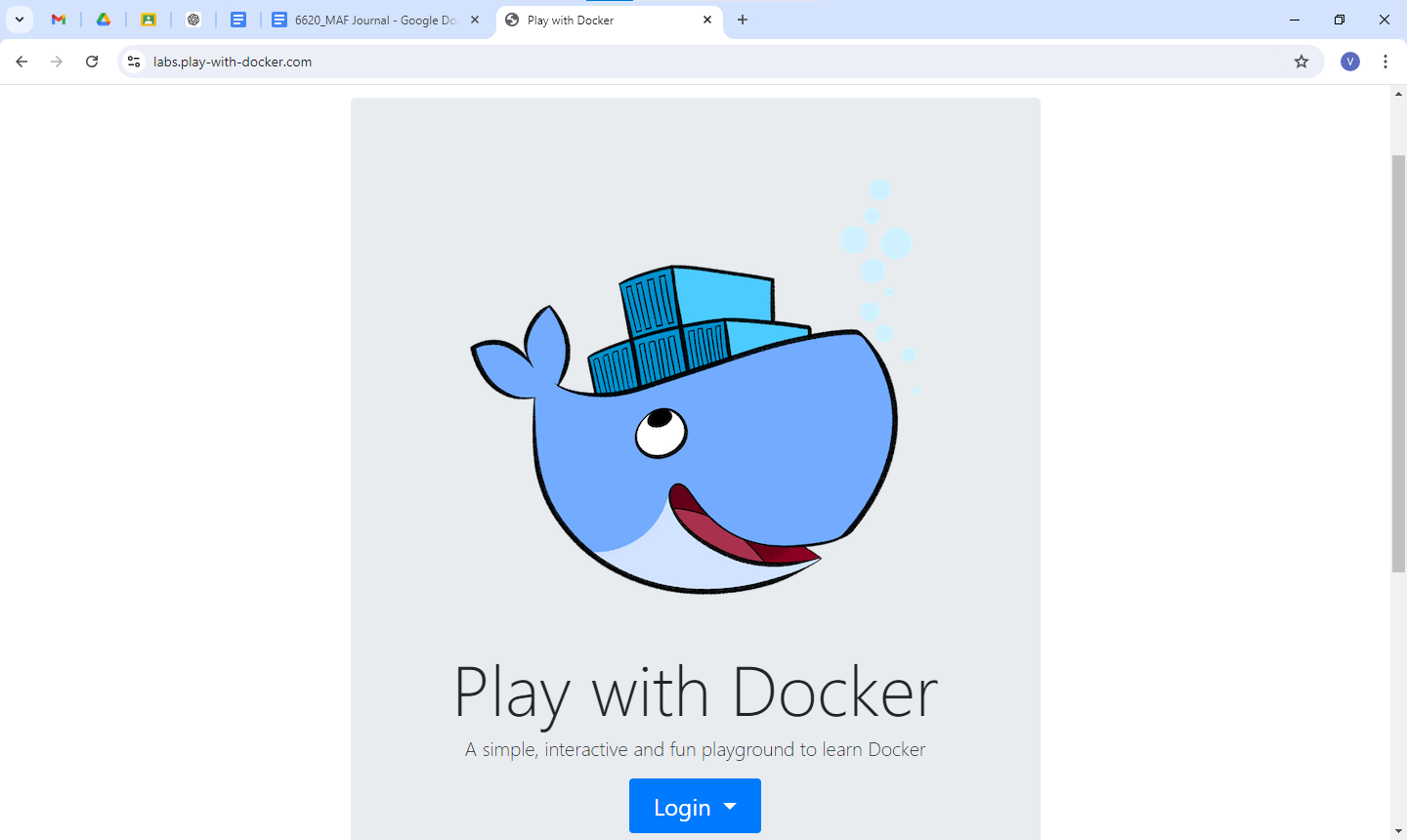


deploy



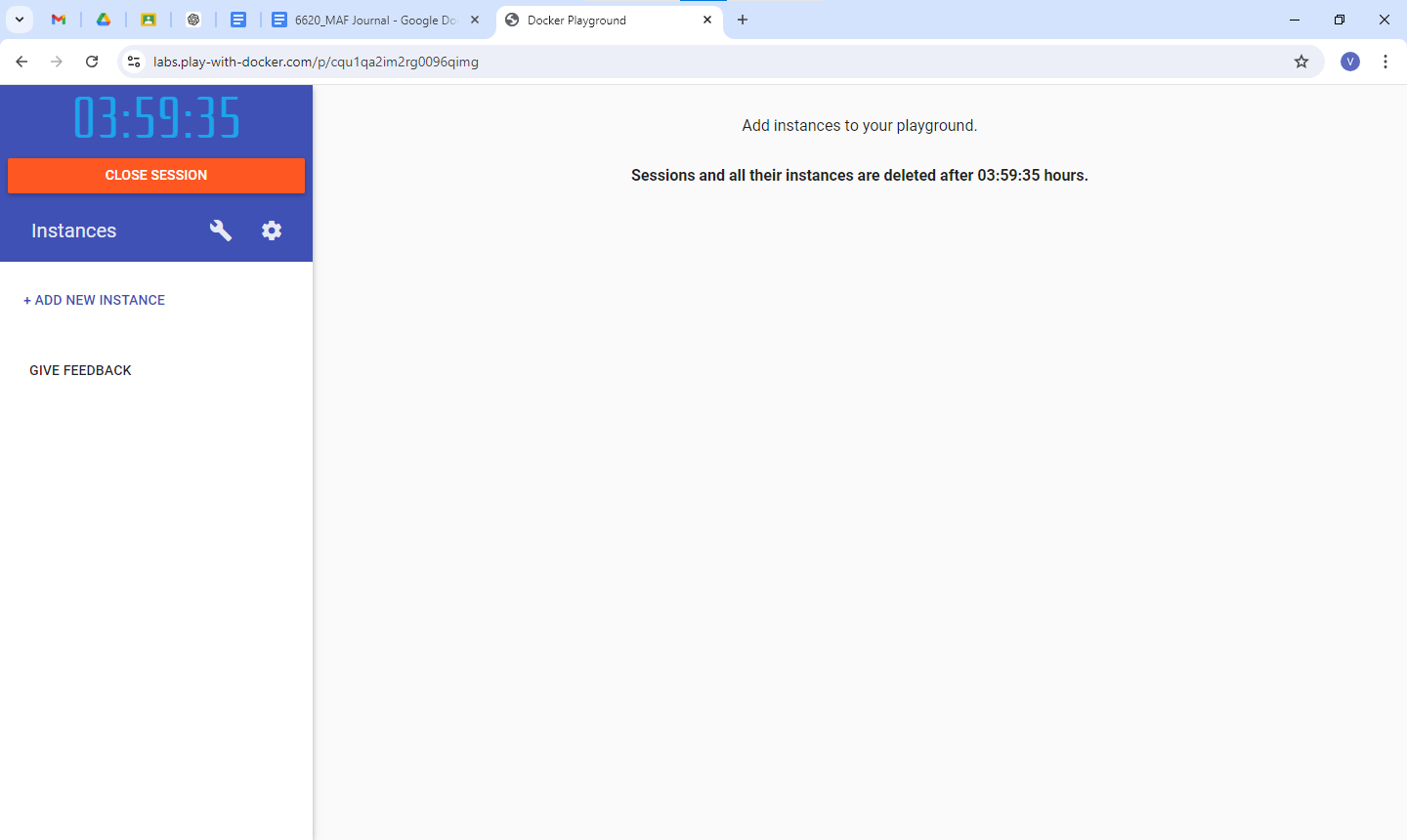
#### Aug 14, 2024 - Play with Docker

<https://labs.play-with-docker.com/>

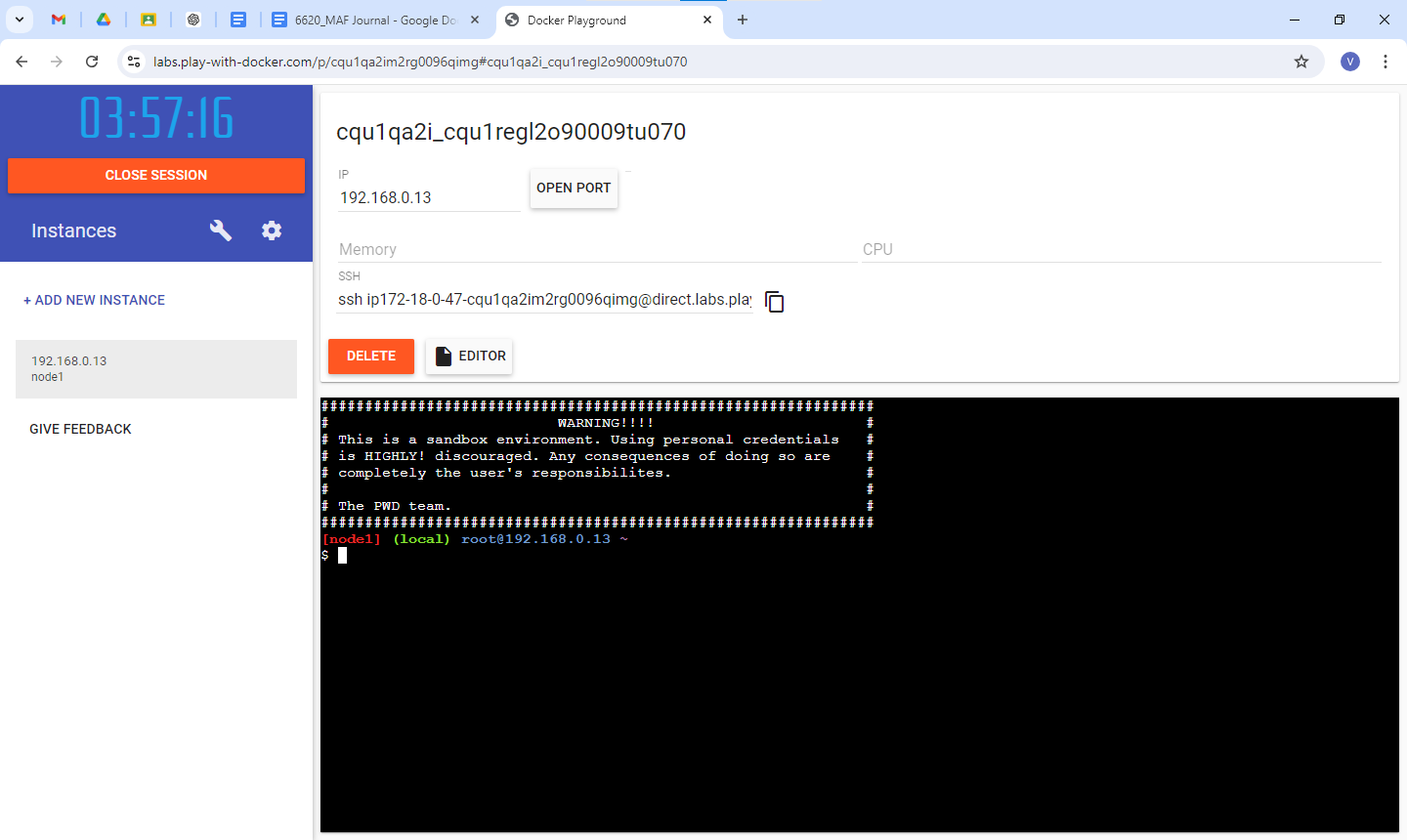


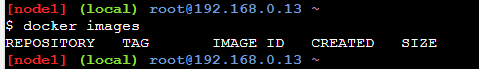
Login with your docker account

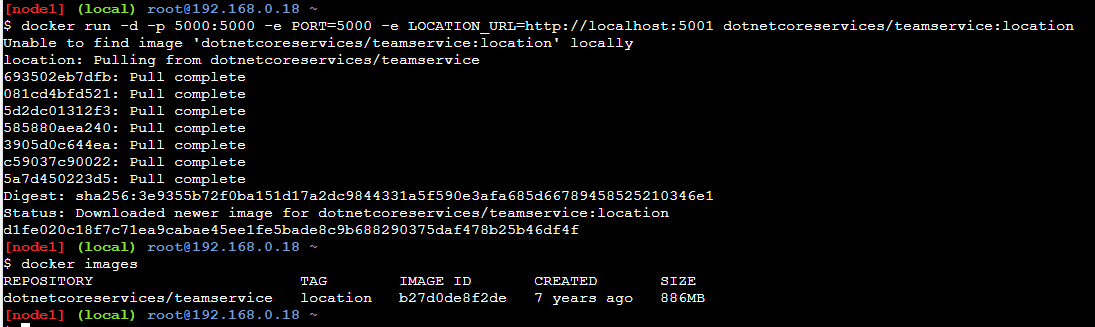
And start

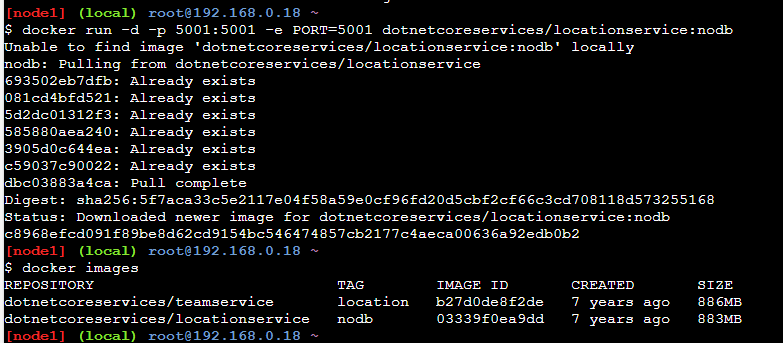


Click on Add new instance

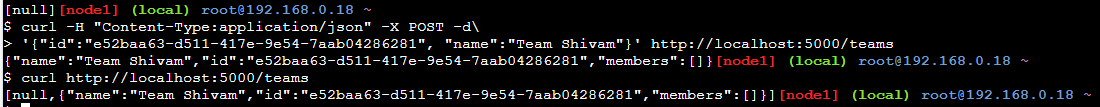












List of commands

1. docker run -d -p 5000:5000 -e PORT=5000 -e LOCATION\_URL=http://localhost:5001 dotnetcoreservices/teamservice:location
2. docker run -d -p 5001:5001 -e PORT=5001 dotnetcoreservices/locationservice:nodb
3. docker images
4. curl http://localhost:5000/teams

Team/Member ID format: 8-4-4-4-12 can be any character separated by dash or minus symbol

1. curl -H "Content-Type: application/json" -X POST -d '{"id":"e52baa63-d511-417e-9e54-7aab04286281", "name":"Team Shivam"}' http://localhost:5000/teams
2. curl -H "Content-Type: application/json" -X POST -d '{"id":"63e7acf8-8fae-42ce-9349-3c8593ac8292", "firstName":"6620", "lastName":"Shivam Vishwakarma"}' http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281/members

curl -H "Content-Type: application/json" -X POST -d '{"id":"63e7acf8-8fae-42ce-9349-3c8593ac8293", "firstName":"6621", "lastName":"random\_student"}' http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281/members

1. curl http://localhost:5000/teams/e52baa63-d511-417e-9e54-7aab04286281
2. curl -H "Content-Type: application/json" -X POST -d '{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f", "latitude":12.0, "longitude":12.0, "altitude":10.0, "timestamp":0, "memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
3. curl http://localhost:5001/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292

**Github links**

Team Service: <https://github.com/microservices-aspnetcore/teamservice>

Location Service: <https://github.com/microservices-aspnetcore/locationservice>

Sep 16, 2024 - new

docker run -d -p 5000:5000 -e PORT=5000 -e LOCATION\_URL=http://localhost:5001 dotnetcoreservices/teamservice:location

docker run -d -p 5001:5001 -e PORT=5001 dotnetcoreservices/locationservice:nodb

docker images

curl http://localhost:5000/teams

***Team/Member ID format: 8-4-4-4-12 can be any character separated by dash or minus symbol***

curl -H "Content-Type: application/json" -X POST -d '{"id":"team1111-team-1111-team-1111team0001", "name":"6620"}' <http://localhost:5000/teams>

curl -H "Content-Type: application/json" -X POST -d '{"id":"member0-0000-0000-0000-000000000001", "firstName":"Shivam", "lastName":"Vishwakarma"}' http://localhost:5000/teams/team0000-0000-0000-0000-000000000001/members

curl -H "Content-Type: application/json" -X POST -d '{"id":"member0-0000-0000-0000-000000000002", "firstName":"Ranvijay", "lastName":"Shukla"}' http://localhost:5000/teams/team0000-0000-0000-0000-000000000001/members

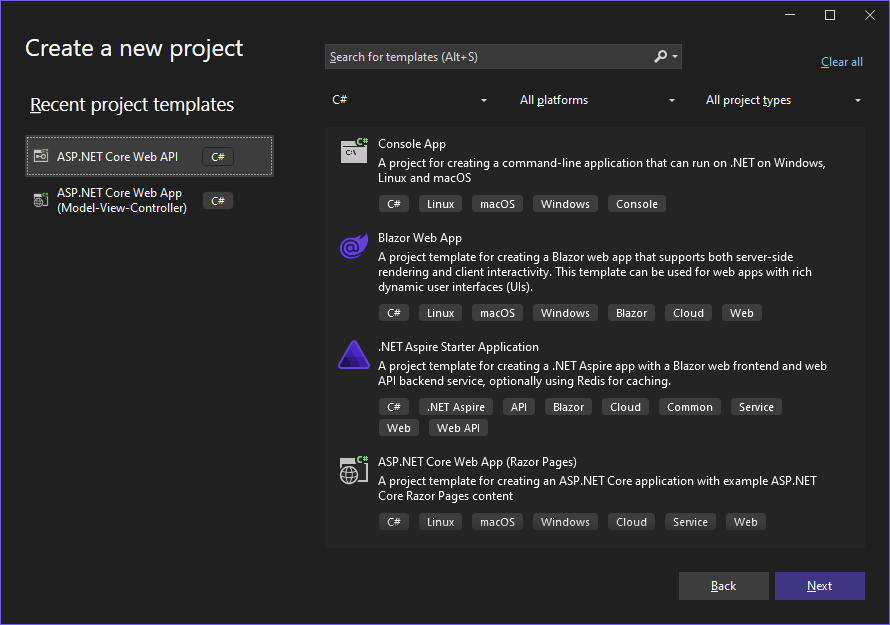
curl http://localhost:5000/teams/member0-0000-0000-0000-000000000001

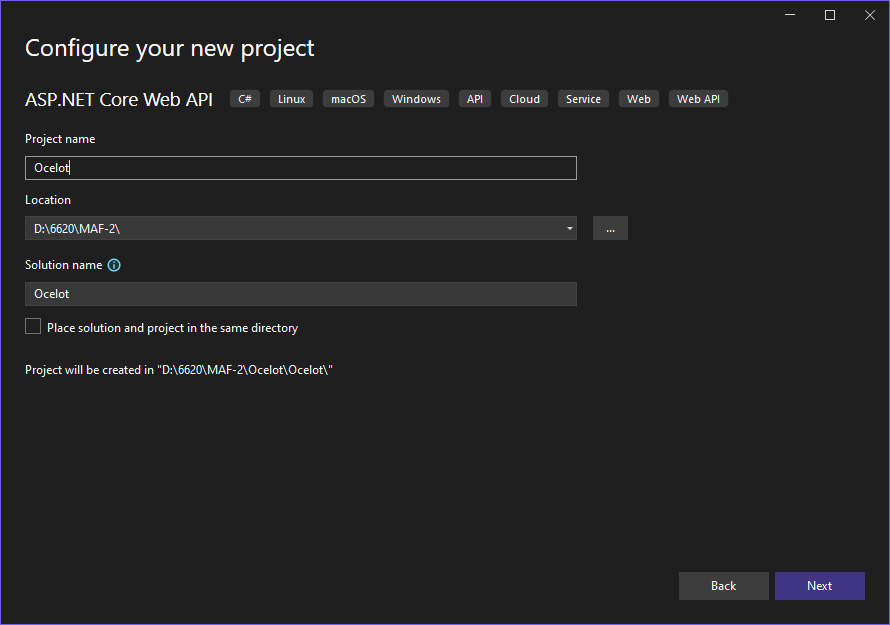
curl -H "Content-Type: application/json" -X POST -d '{"id":"location-0000-0000-0000-000000000000", "latitude":12.0, "longitude":12.0, "altitude":10.0, "timestamp":0, "memberId":"member0-0000-0000-0000-000000000001"}' http://localhost:5001/locations/member0-0000-0000-0000-000000000001

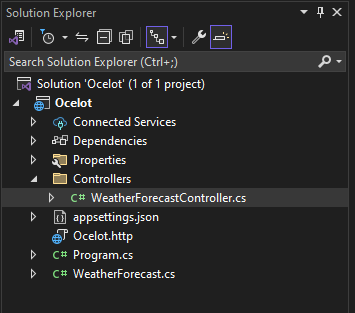
curl http://localhost:5001/locations/member0-0000-0000-0000-000000000001

#### Aug 21, 2024 -

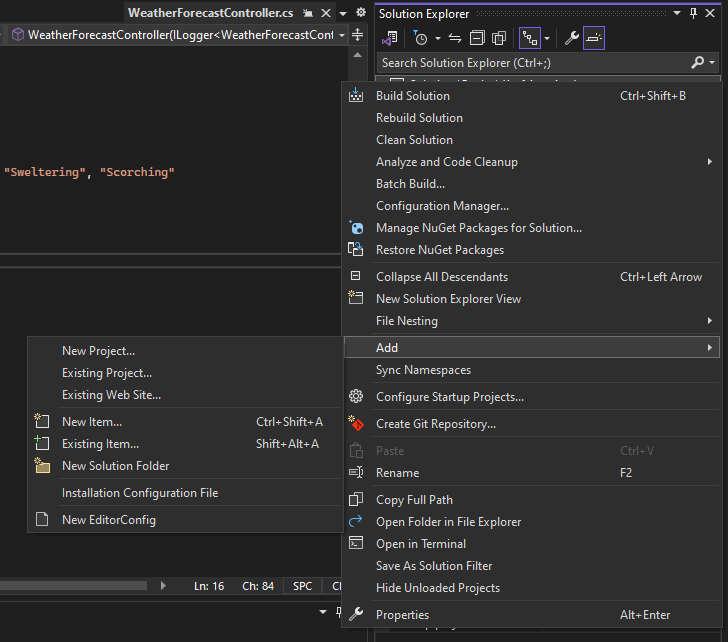
Create a new web api project

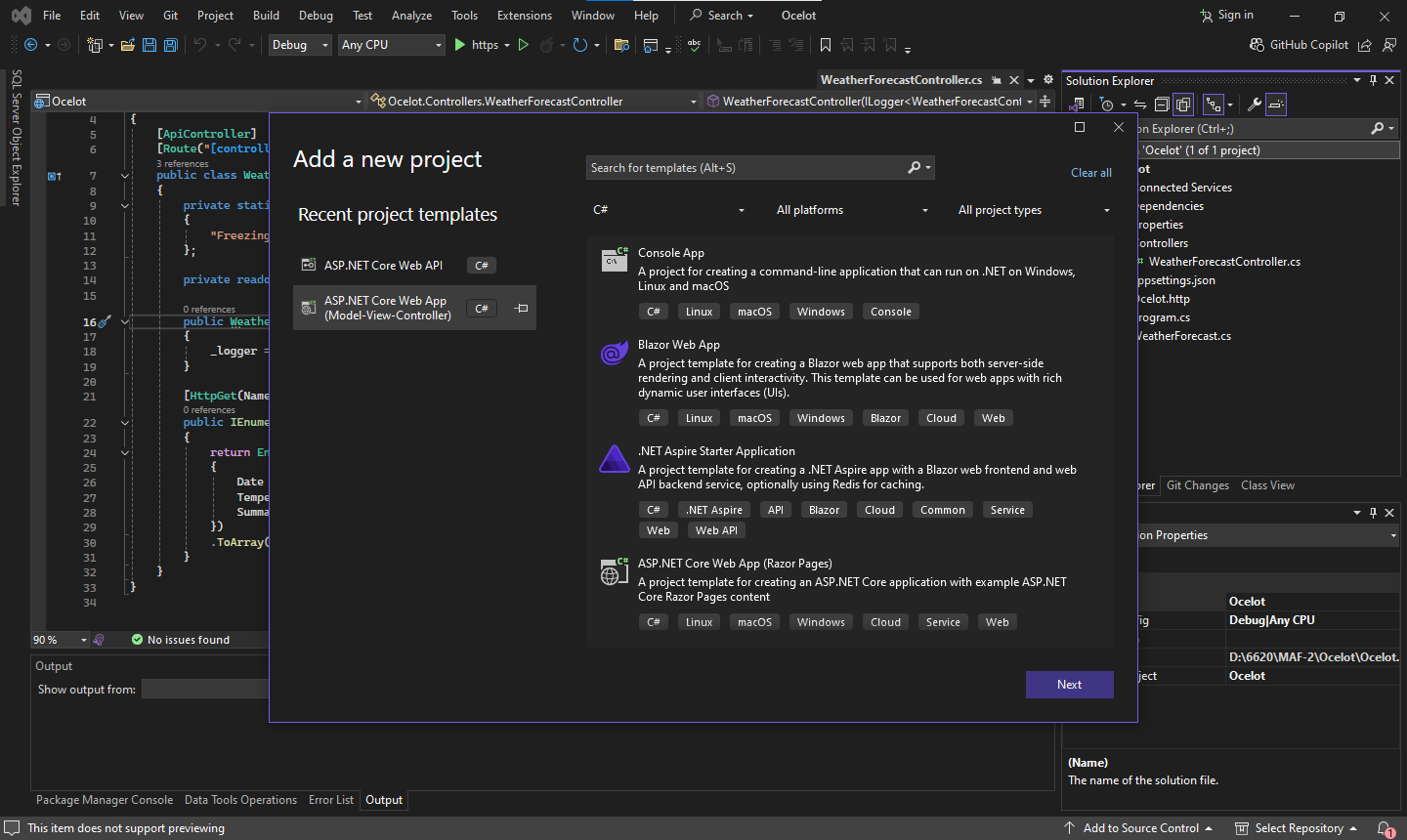


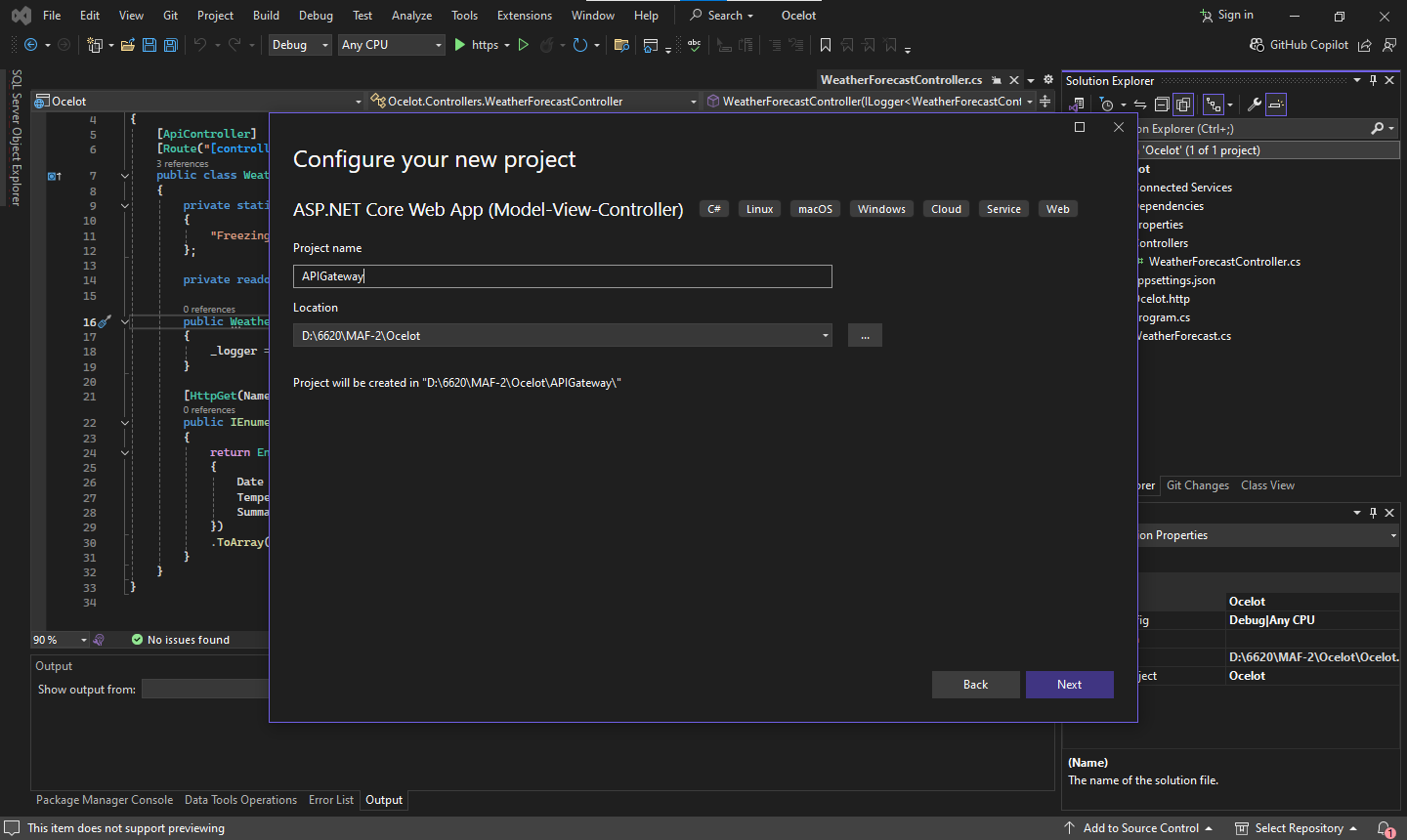




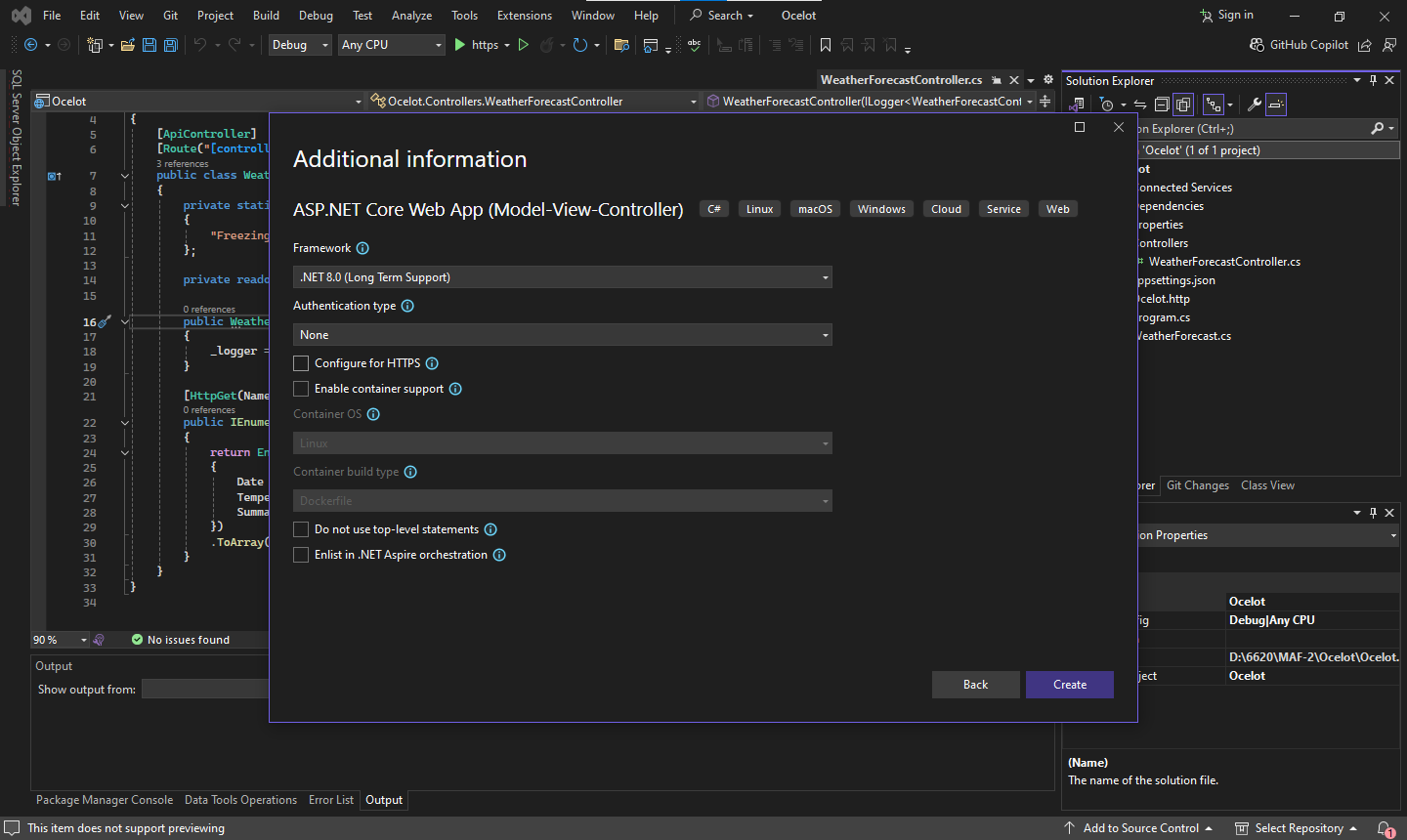
Right click on Solution Ocelot → Add → new project

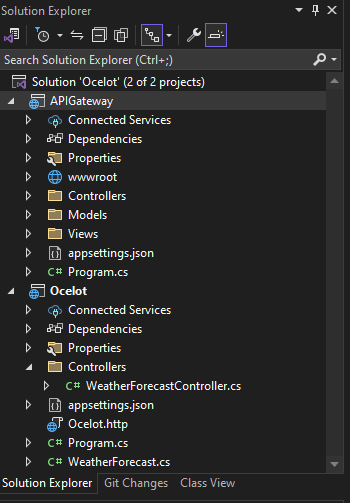




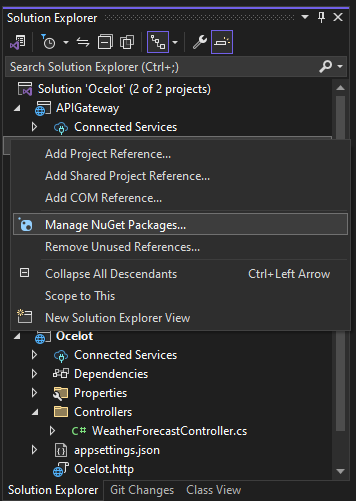


Uncheck “Configure for HTTPS” Checkbox

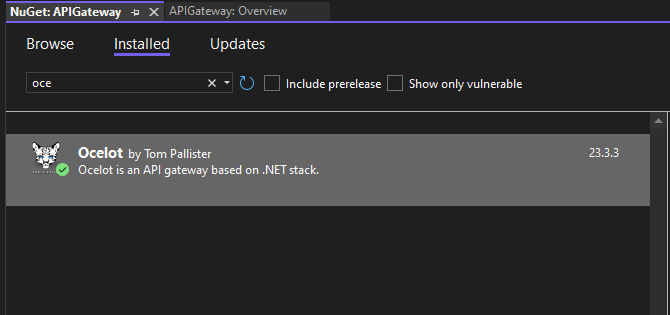


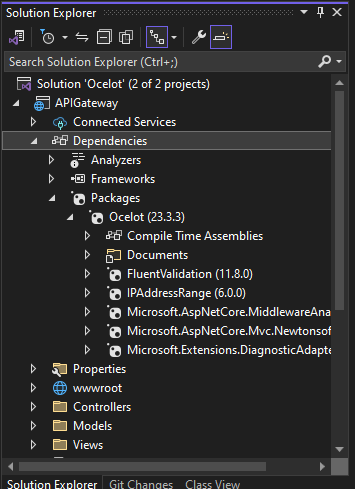


Right click on Dependencies → manage NuGet packages

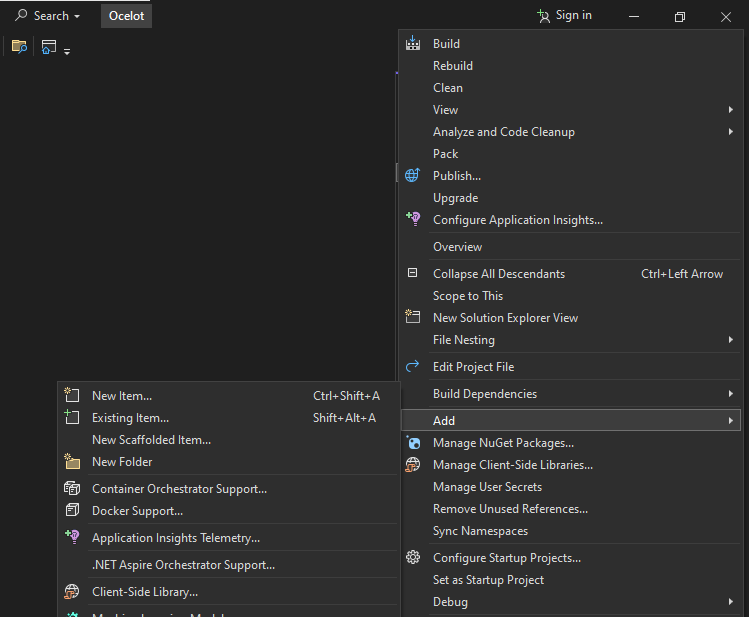


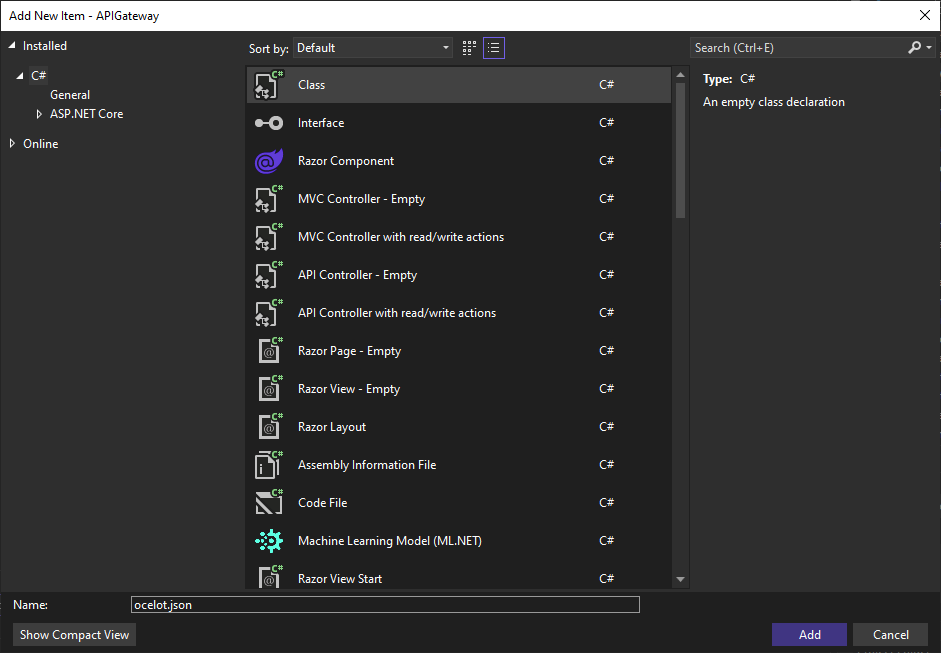
Go to Browse → search Ocelot and install the packages

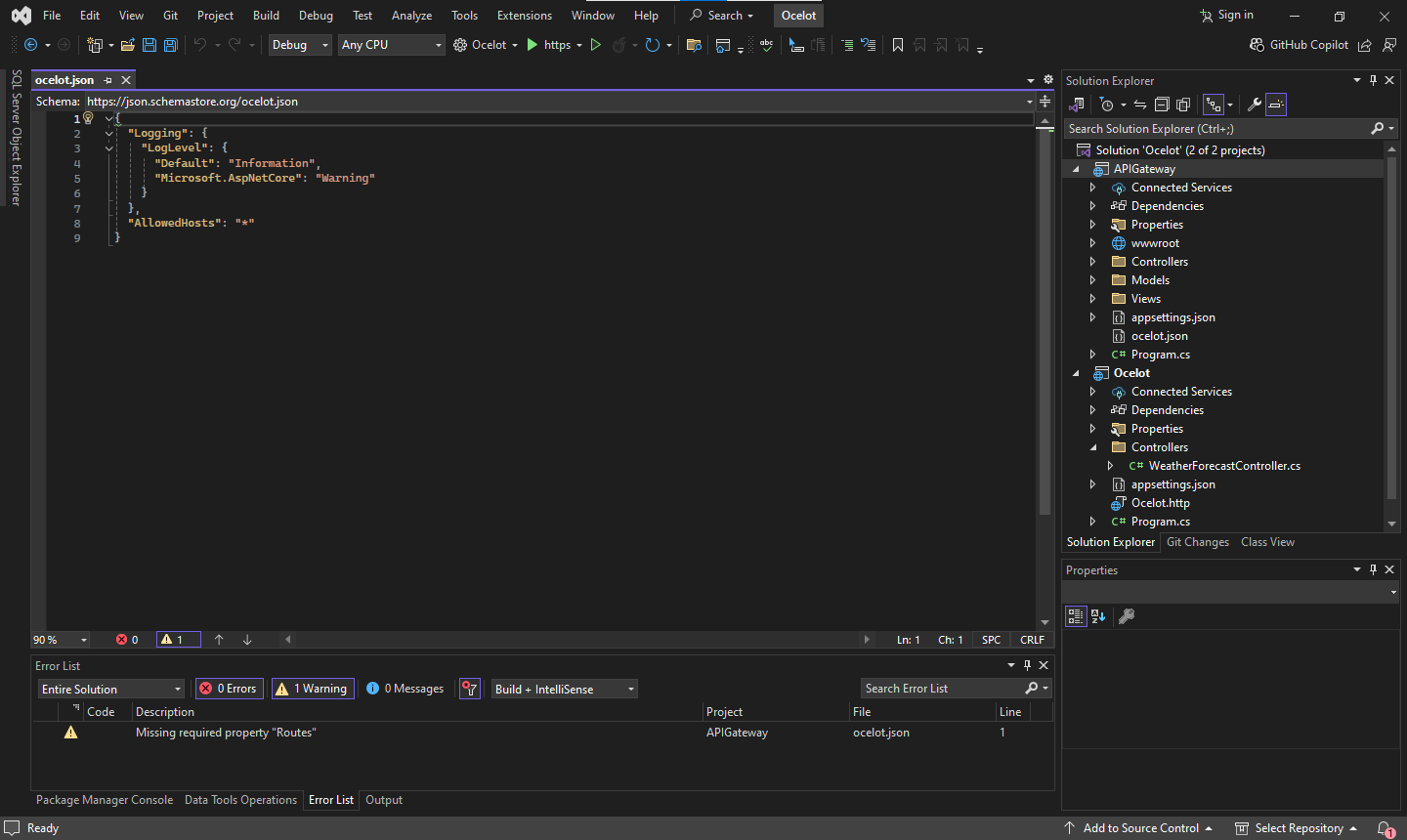




Right click on API Gateway root dir and add → new item and add ocelot.json







Write below following code in json file

**"Routes": [**

**{**

**"DownstreamPathTemplate": "/api/weatherforecast",**

**"DownstreamScheme": "http",**

**"DownstreamHostAndPorts": [**

**{**

**"Host": "localhost",**

**"Port": 44352**

**}**

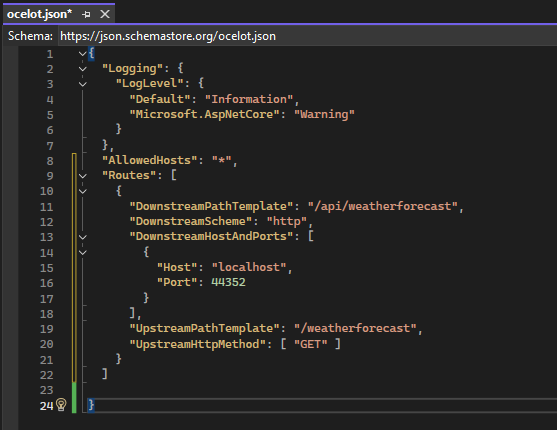
**],**

**"UpstreamPathTemplate": "/weatherforecast",**

**"UpstreamHttpMethod": [ "GET" ]**

**}**

**]**

****

And then go to APIGateway Program.cs file leave the default code as it is and write below following code

**using Ocelot.DependencyInjection;**

**using Ocelot.Middleware;**

var builder = WebApplication.CreateBuilder(args);

//Add services to the container.

builder.Services.AddControllersWithViews();

var app = builder.Build();

//Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

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

app.Run();

**builder.Configuration.SetBasePath(builder.Environment.ContentRootPath)**

**.AddJsonFile("ocelot.json", optional: false, reloadOnChange: true)**

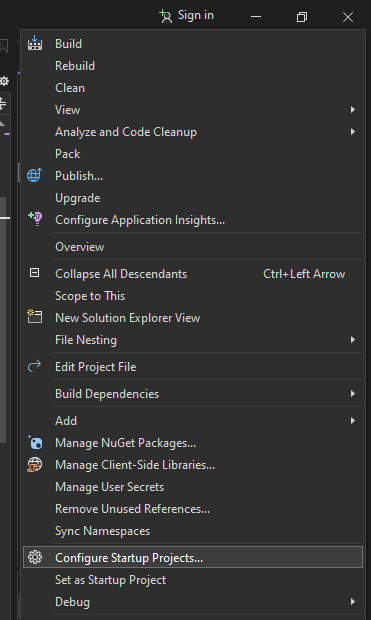
**.AddEnvironmentVariables();**

**builder.Services.AddOcelot(builder.Configuration);**

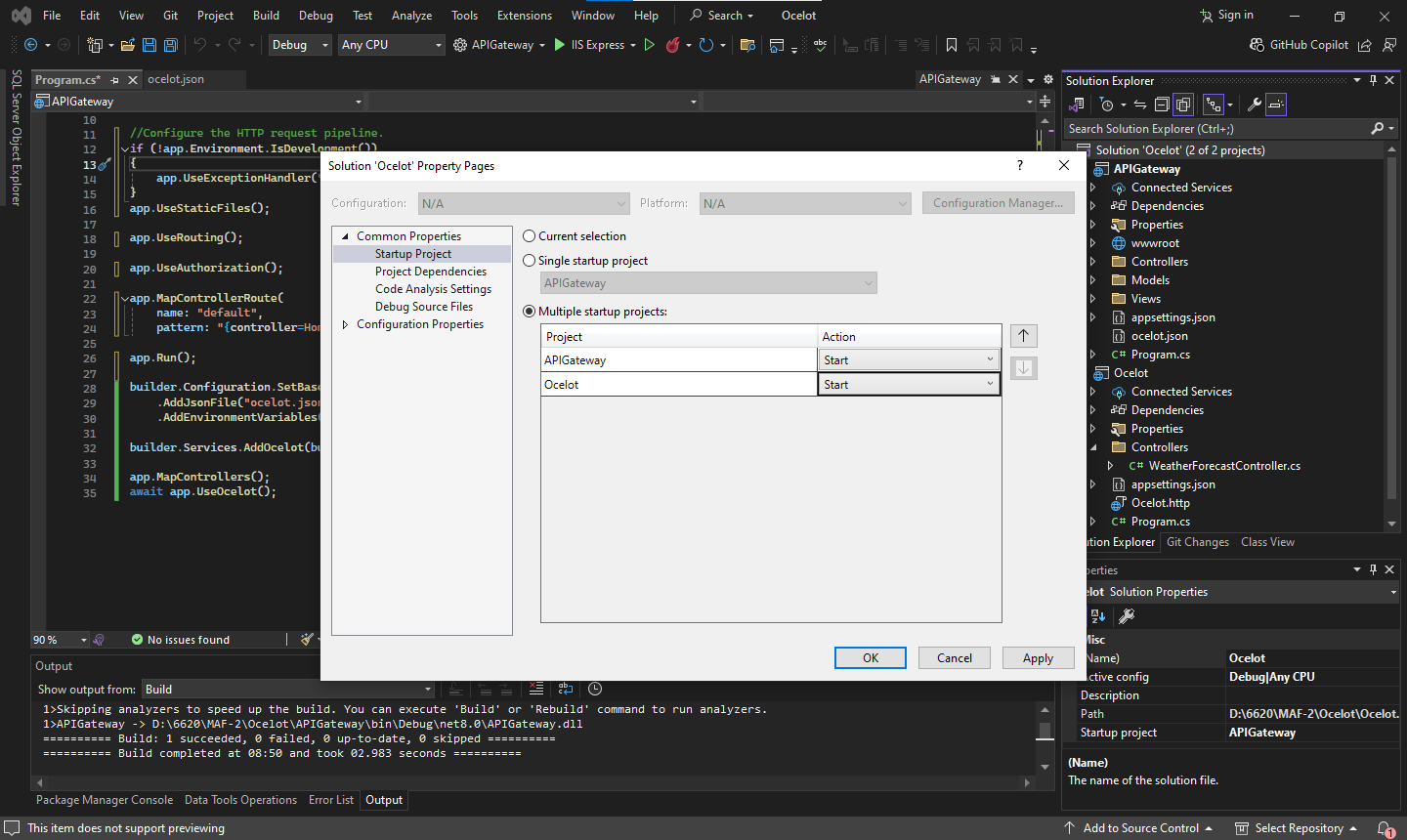
**app.MapControllers();**

**await app.UseOcelot();**

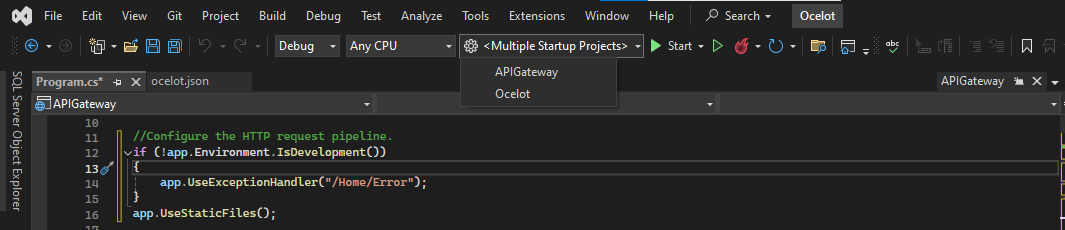
Right click on APIGateway → configure startup Project

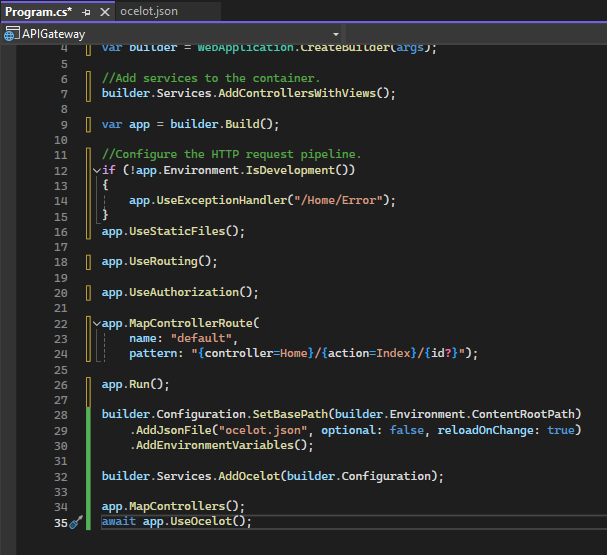


Set both project as start and then → apply →ok



Multiple project option will be visible



****