Lab Manual for Cloud Computing

Lab No. 9

Creating Micro services Using ASP.NET Core

LAB 09: CREATING MICRO SERVICES USING ASP.NET CORE

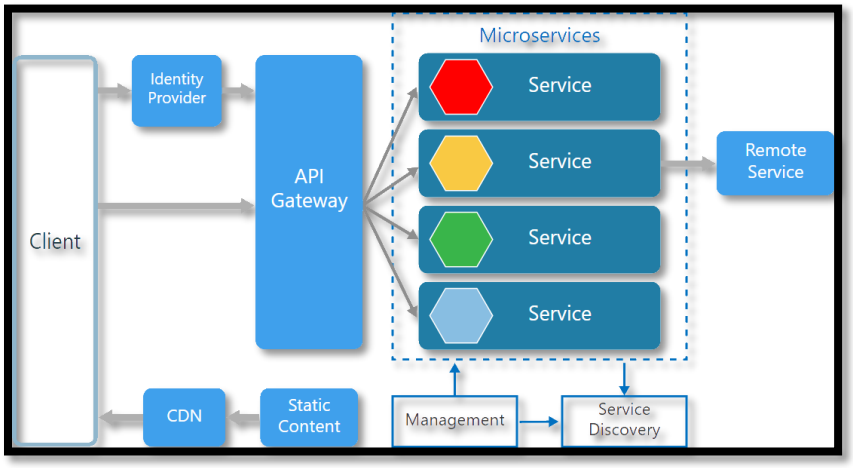
1. INTRODUCTION:

Microservices

The term microservices portrays a software development style that has grown from contemporary trends to set up practices those are meant to increase the speed and efficiency of developing and managing software solutions at scale. Microservices is more about applying a certain number of principles and architectural patterns an architecture. Each microservice lives independently, but on the other hand, also all rely on each other. All microservices in a project get deployed in production at their own pace, on- premise on the cloud, independently, living side by side.

Microservices Architecture

The following picture from Microsoft Docs shows the microservices architecture style.



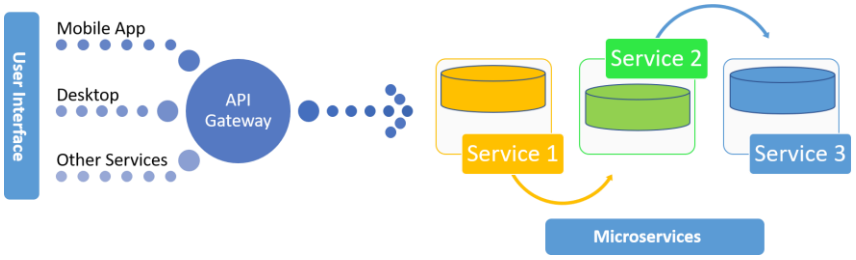
There are various components in a microservices architecture apart from microservices themselves.

1. **Management**: Maintains the nodes for the service.
2. **Identity Provider:** Manages the identity information and provides authentication services within a distributed network.
3. **Service Discovery:** Keeps track of services and service addresses and endpoints.
4. **API Gateway:** Serves as client's entry point. Single point of contact from the client which in turn returns responses from underlying microservices and sometimes an aggregated response from multiple underlying microservice.
5. **CDN:** A content delivery network to serve static resources for e.g. pages and web content in a distributed network.
6. **Static Content:** The static resources like pages and web content.

Microservices are deployed independently with their own database per service so the underlying microservices look as shown in the following picture.

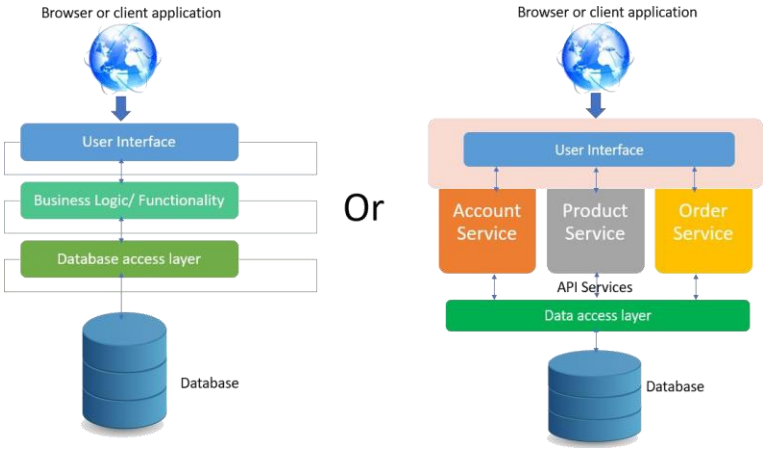
Monolithic vs Microservices Architecture

Monolithic applications are more of a single complete package having all the related needed components and services encapsulated in one package. Following is the diagrammatic representation of monolithic architecture being package completely or being service based.

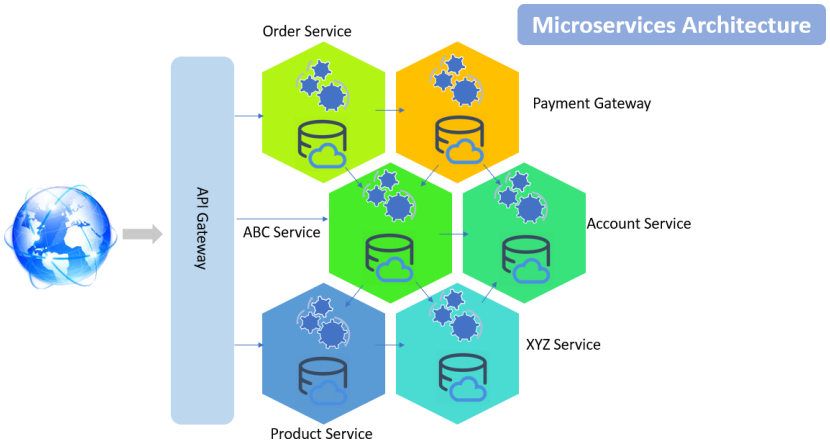


Monolithic vs Microservices Architecture

Monolithic applications are more of a single complete package having all the related needed components and services encapsulated in one package. Following is the diagrammatic representation of monolithic architecture being package completely or being service based.



Microservice is an approach to create small services each running in their own space and can communicate via messaging. These are independent services directly calling their own database. Following is the diagrammatic representation of microservices architecture.



Example of Creating MicroService in ASP .Net

Step 1: Create "Producs" table in MSSQL database

In this Lab, we will see how to create a simple micro service for Product data entry. So, we need to create a "Products" table first.

- Creating database named "CC_Lab_09".

```
create database CC_Lab_09;
```

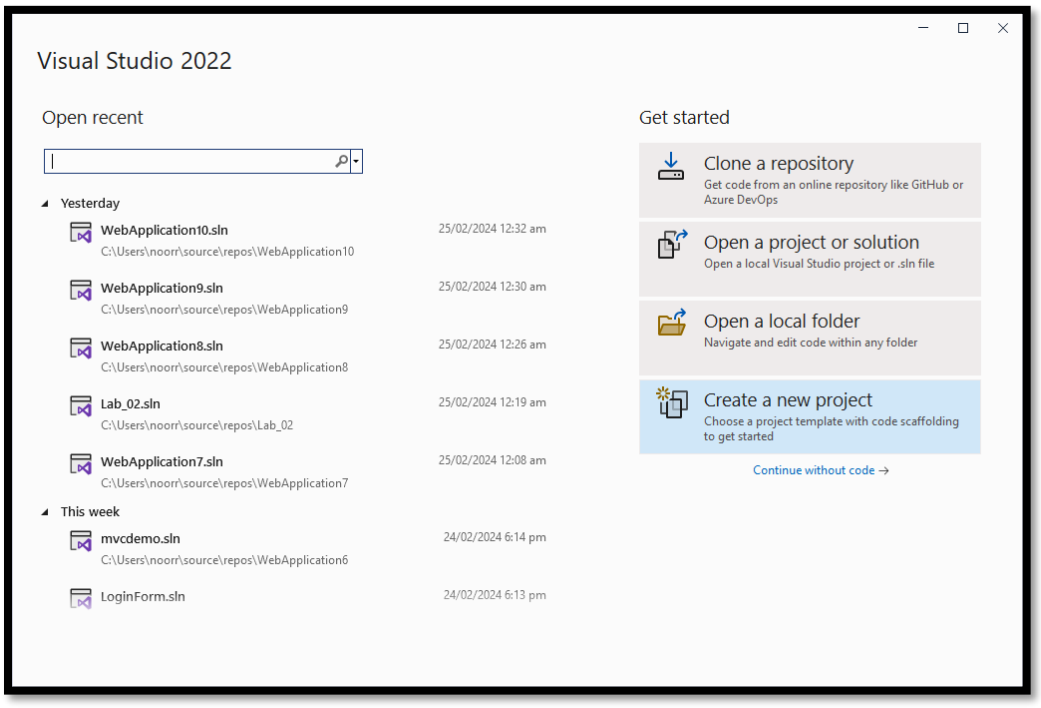
- Using the created database and then design tabel named as Student.

```
use CC_Lab_09;

create table Products(
   Id int not null primary key,
   Name varchar(255),
   Description varchar(255),
   Price decimal(10,2));
```

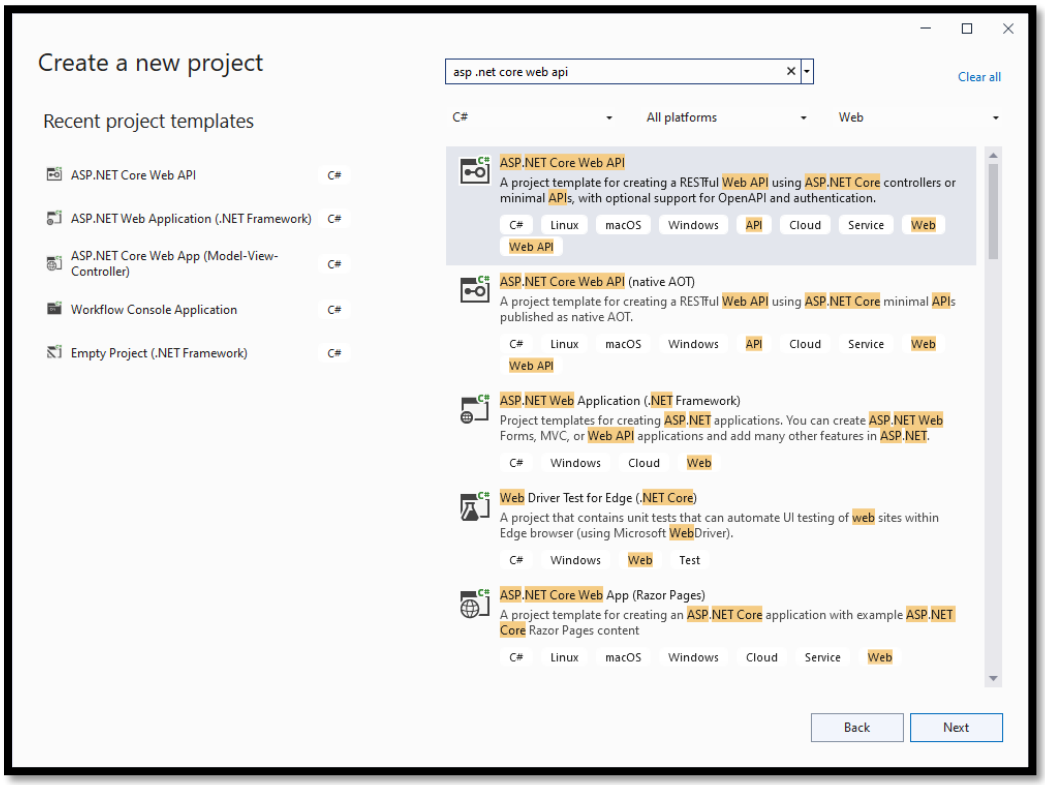
Step 2: Create a new project in Visual Studio 2022

- Open Visual Studio 2022.

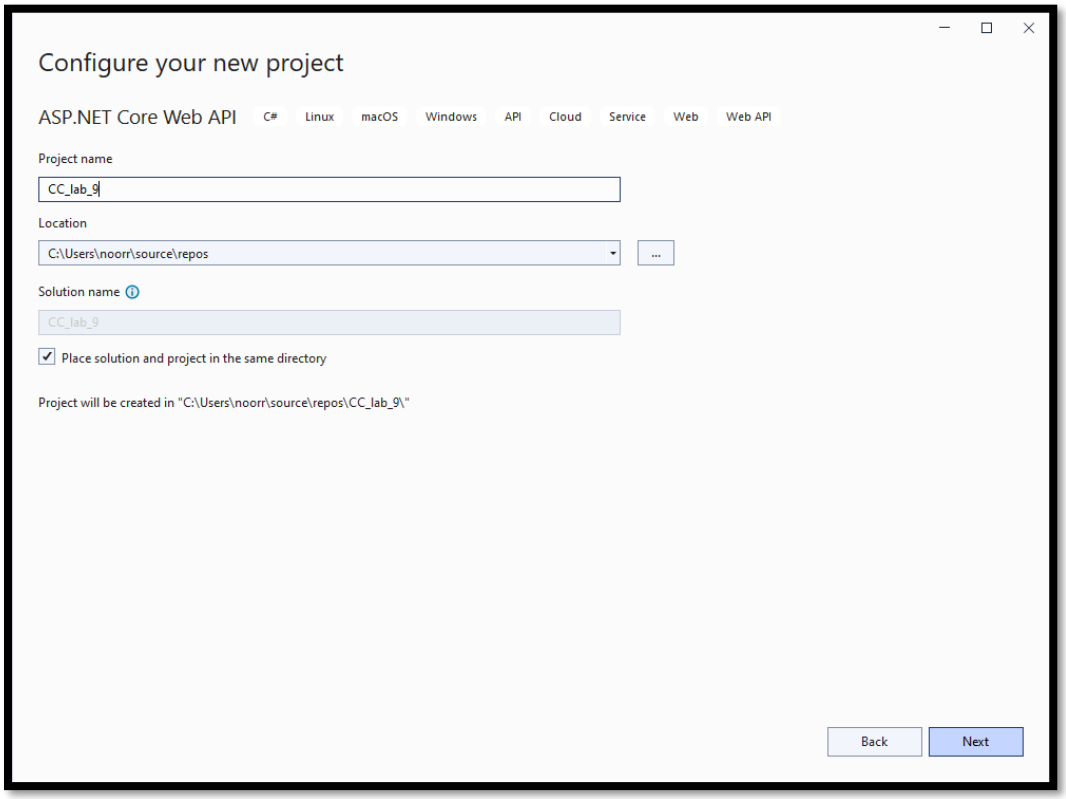
- Click on "Create a new project" in the start window.

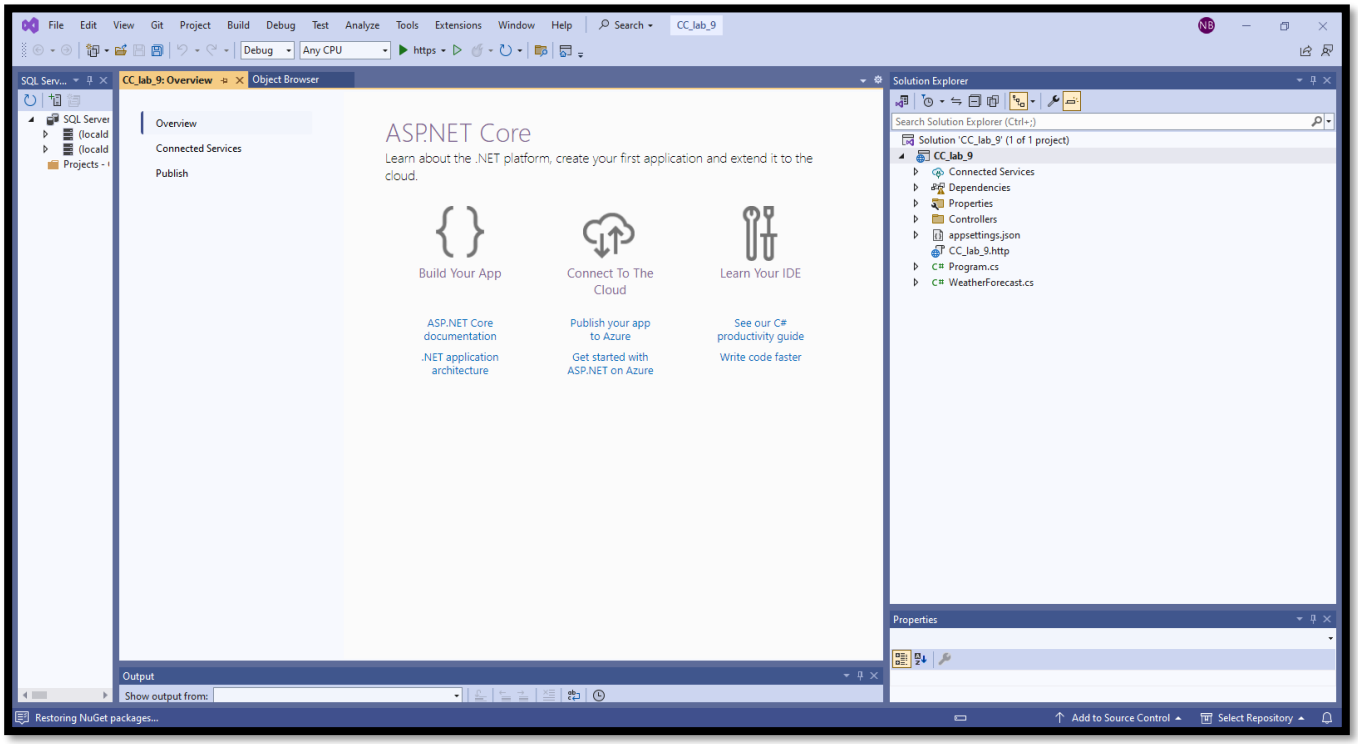


- In the "Create a new project" window, search for **ASP.NET Core Web API** in the search bar, and then click NEXT.



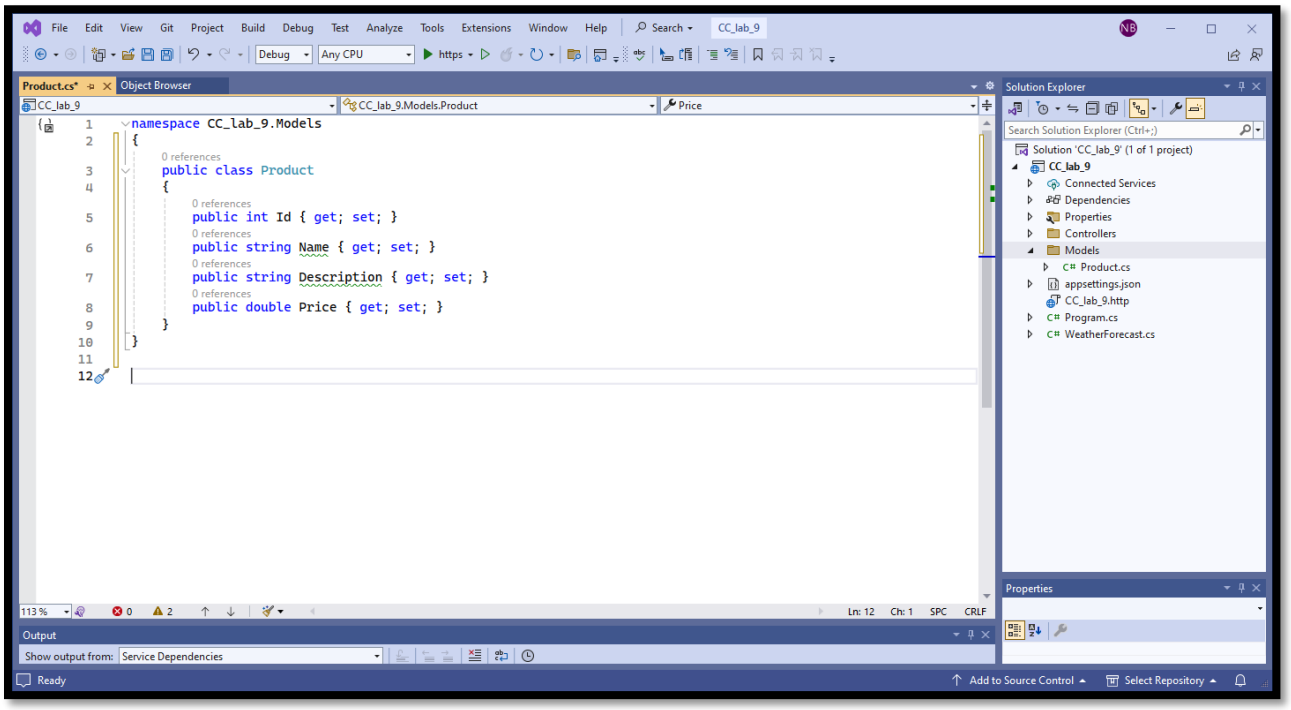
- Provide a name and location for your project and then Click NEXT.





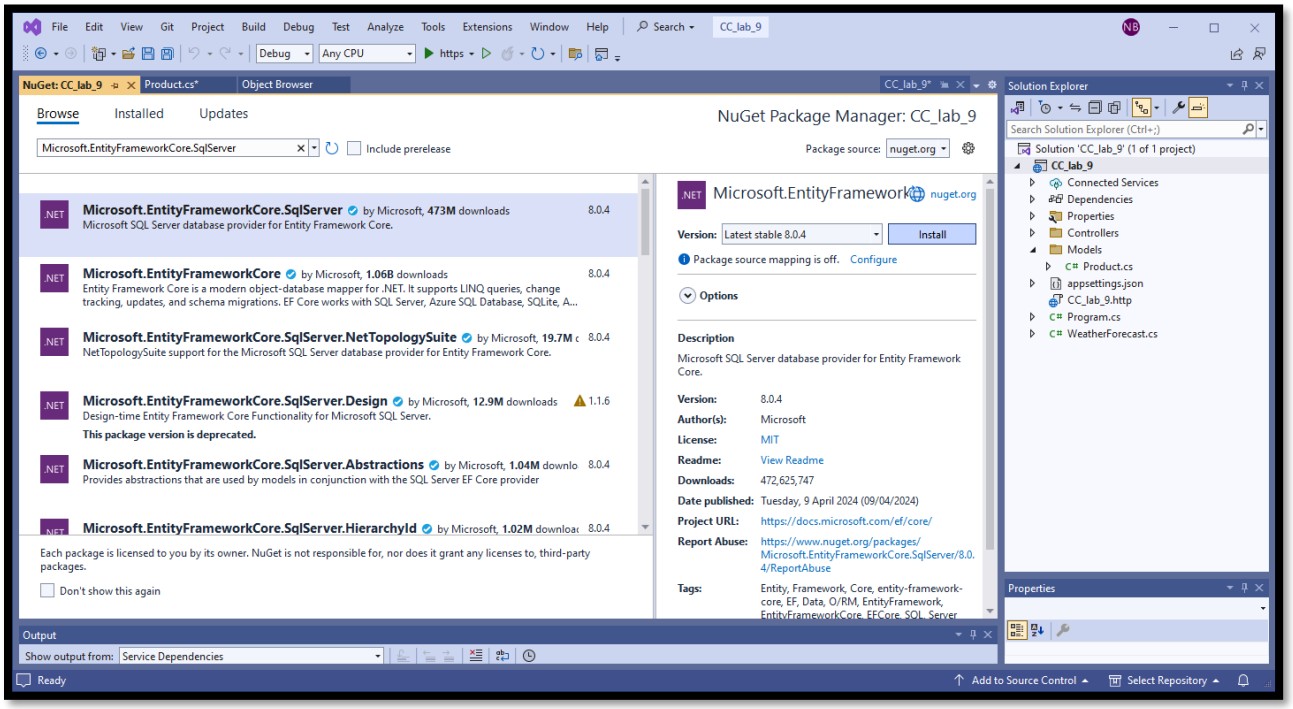
Step 3: Adding a Model

- Add a new folder named "Models" to the project, and in this folder add a class named "product". Then add the following code in it.



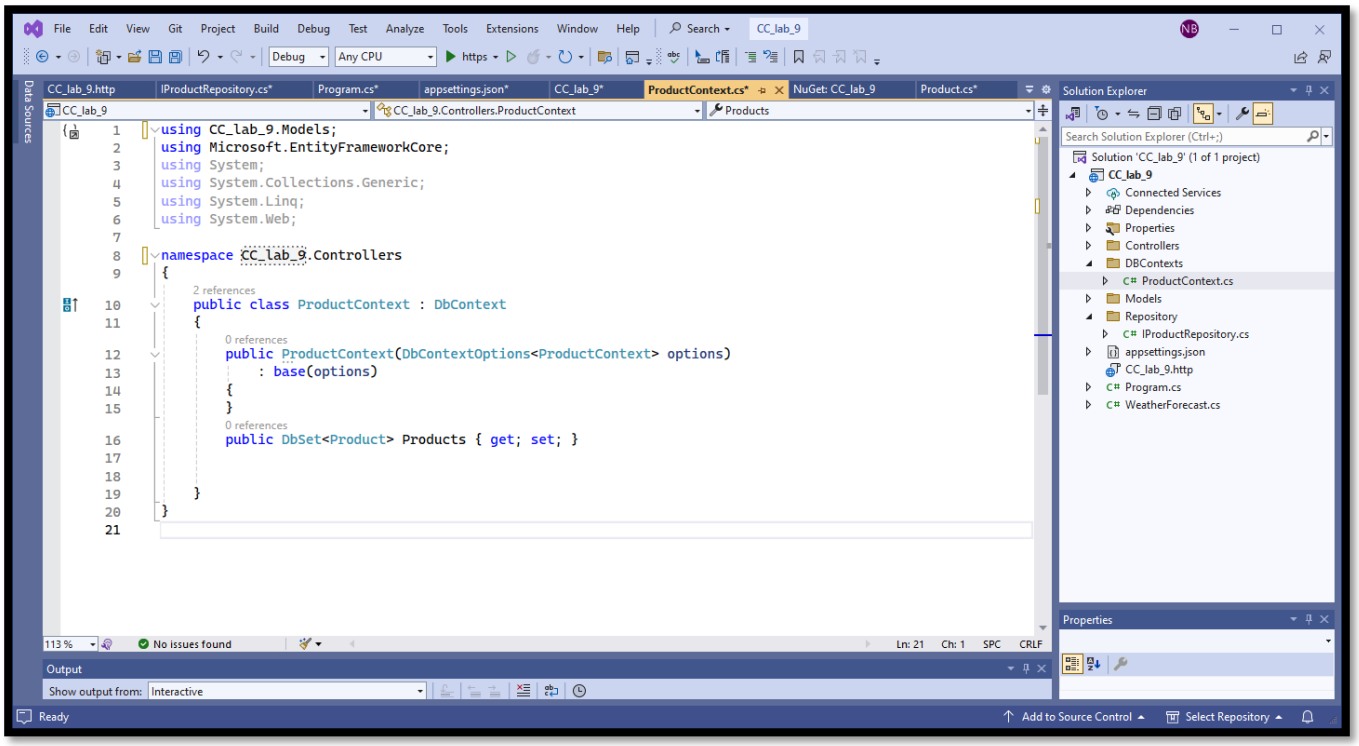
Step 4: Adding a NuGet package

- Right-click on your project in Solution Explorer, and then Choose "Manage Nugget Packages". On opening the new window, Select the **Browse** tab. Enter **Microsoft.EntityFrameworkCore.SqlServer** in the search box, and then select it, then click install.

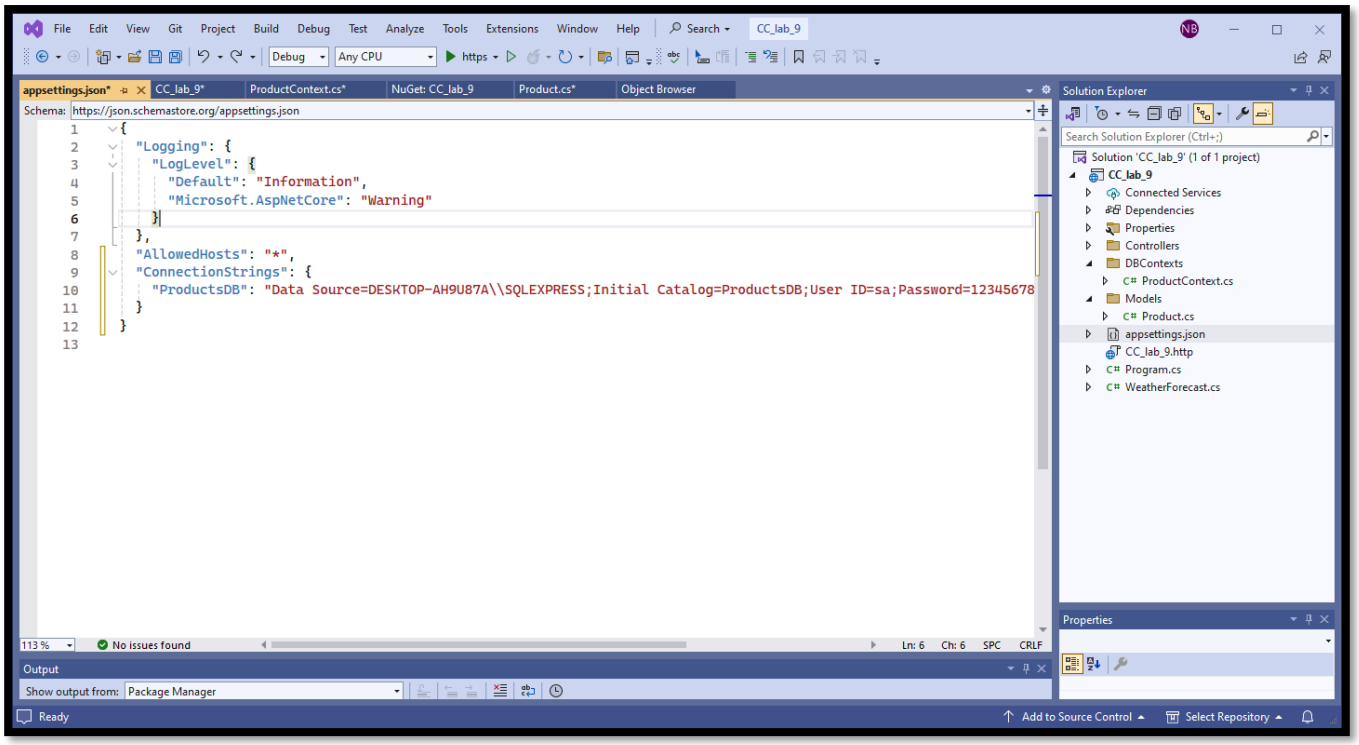


Step 5: Adding EF Core DbContext

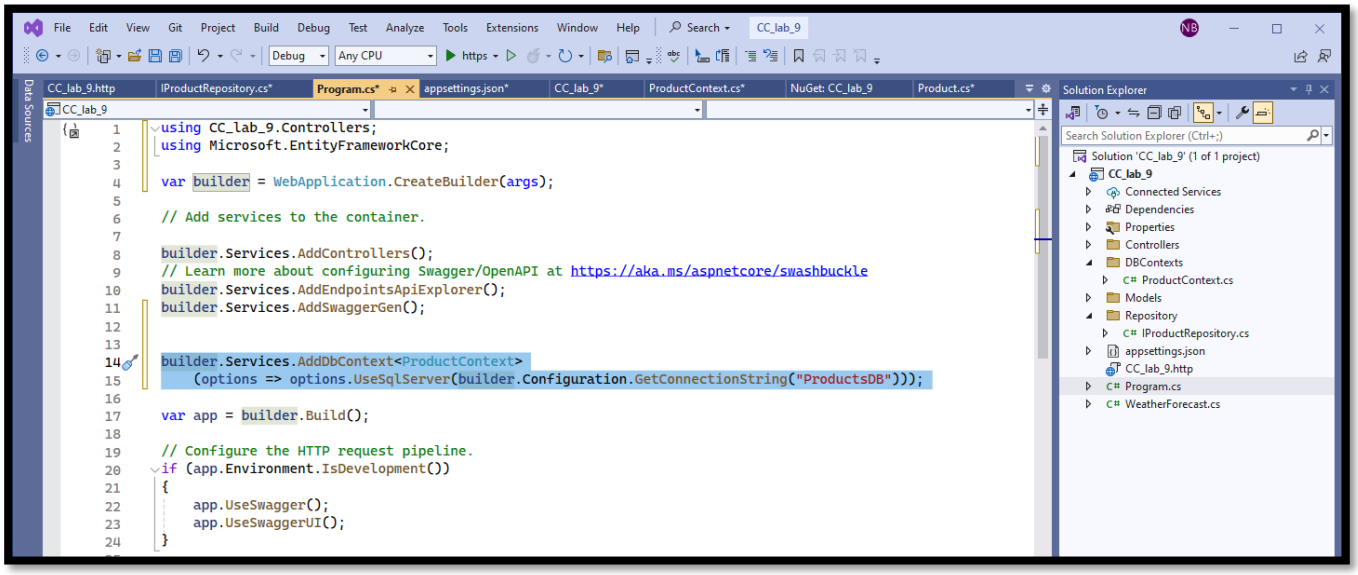
- Right click the project name and add a new folder named **"Db Context".** Then add a new class named **ProductContext** which includes the DbSet properties for Products. Add the following code in it.



- Add a connection string in the appsettings.json file.



- Open the **Program.cs** file to add the SQL server db provider for EF Core. Add the following code. Note that in the GetConnectionString method the name of the key of the connection string is passed that was added in appsettings file.

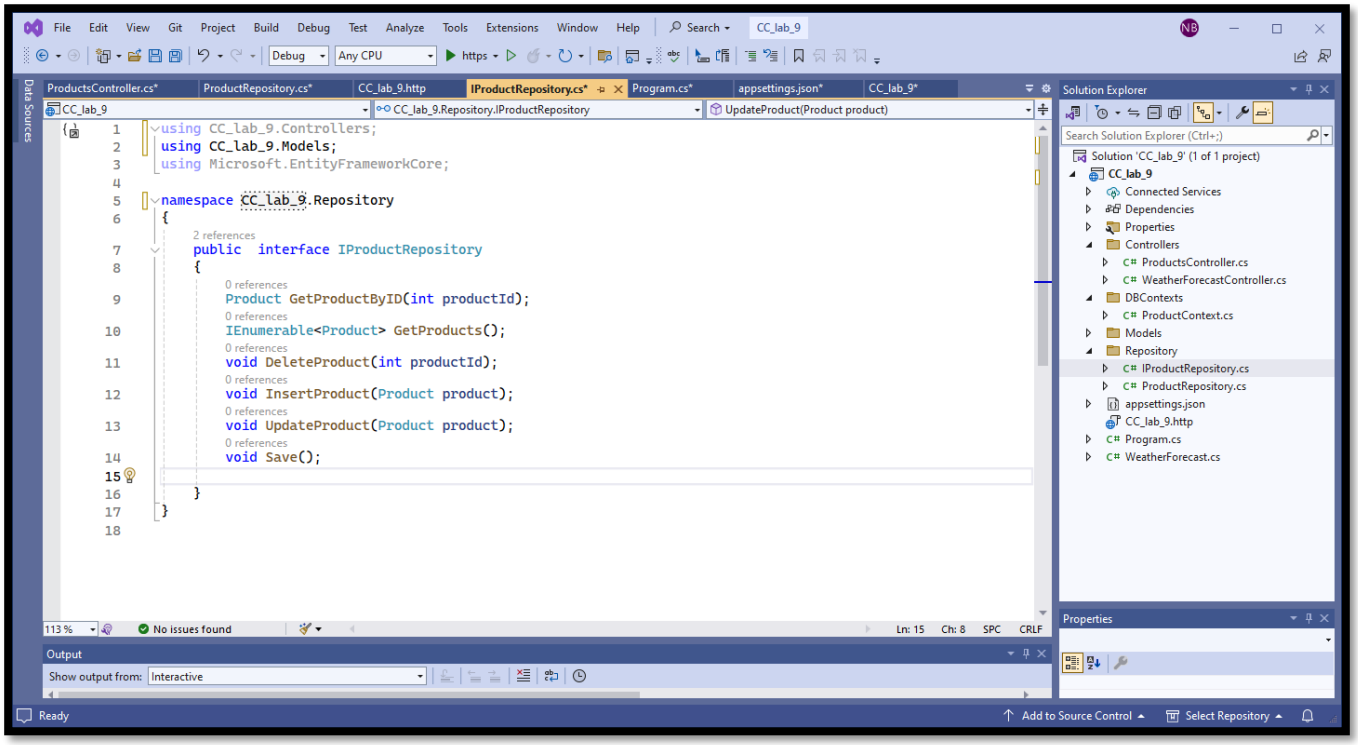


```
using CC_lab_9.Controllers;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();


builder.Services.AddDbContext<ProductContext>
    (options => options.UseSqlServer(builder.Configuration.GetConnectionString("ProductsDB")));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Step 6: Adding Repository

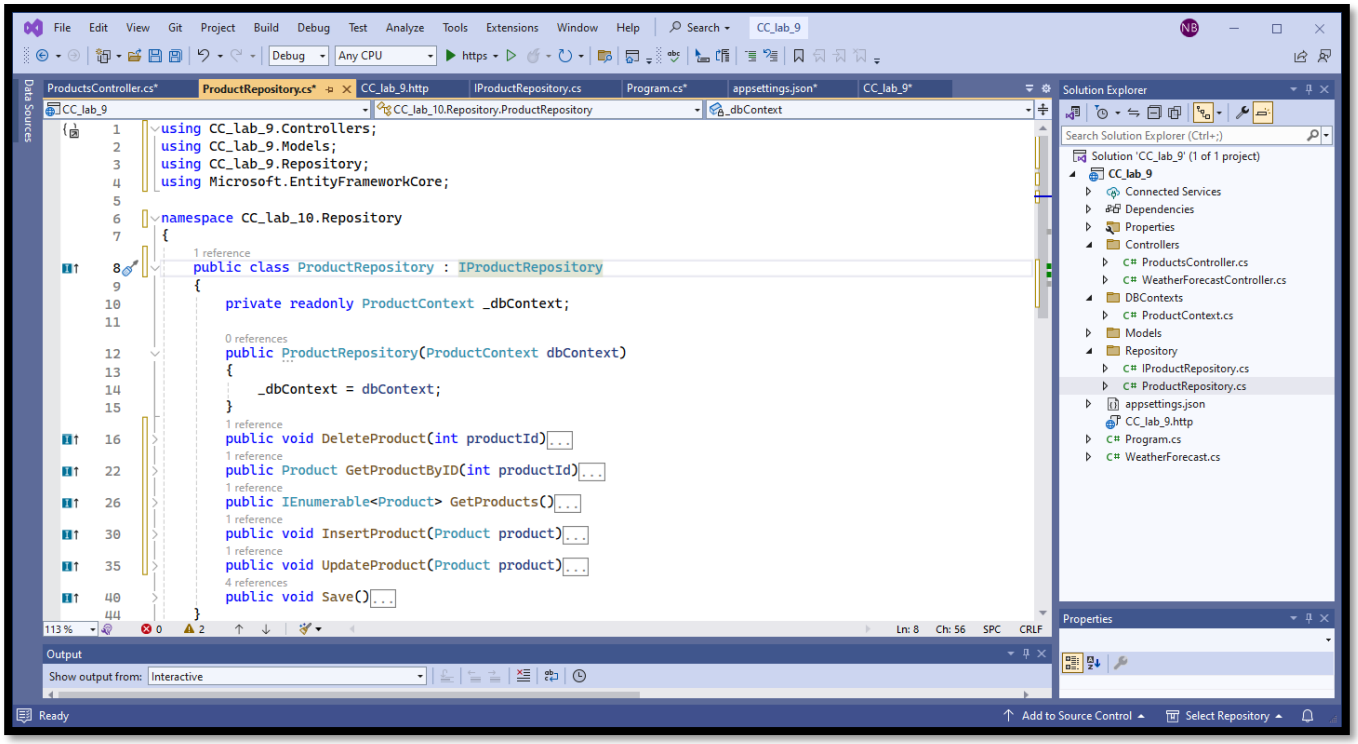
Repository works as a micro component of microservice that encapsulates the data access layer and helps in data persistence and testability as well.

- Add a new folder named **Repository** in the project and add an Interface name **IProductRepository in** that folder. Add the methods in the interface that performs CRUD operations for Product microservice.



```
using CC_lab_9.Controllers;
using CC_lab_9.Models;
using Microsoft.EntityFrameworkCore;

namespace CC_lab_9.Repository
{
    2 references
    public  interface IProductRepository
    {
        0 references
        Product GetProductByID(int productId);
        0 references
        IEnumerable<Product> GetProducts();
        0 references
        void DeleteProduct(int productId);
        0 references
        void InsertProduct(Product product);
        0 references
        void UpdateProduct(Product product);
        0 references
        void Save();

    }
}
```

- Add a new concrete class named **ProductRepository** in the same Repository folder that implements IProductRepository. All these methods need implementation.

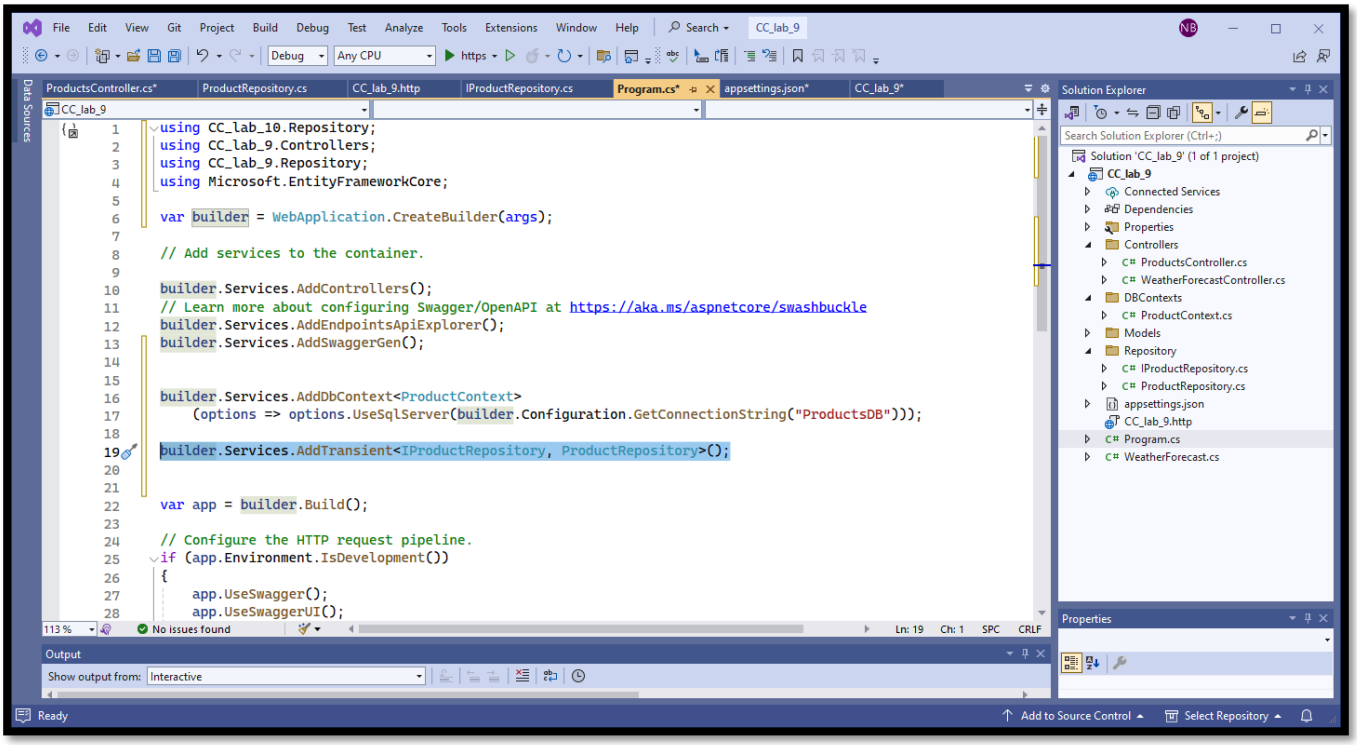


- Add the implementation for the methods via accessing context methods as:

```
public class ProductRepository : IProductRepository
{

    private readonly ProductContext _dbContext;


    public ProductRepository(ProductContext dbContext)
    {
        _dbContext = dbContext;
    }

    public void DeleteProduct(int productId)
    {
        var product = _dbContext.Products.Find(productId);
        _dbContext.Products.Remove(product);
        Save();
    }

    public Product GetProductByID(int productId)
    {
        return _dbContext.Products.Find(productId);
    }
```

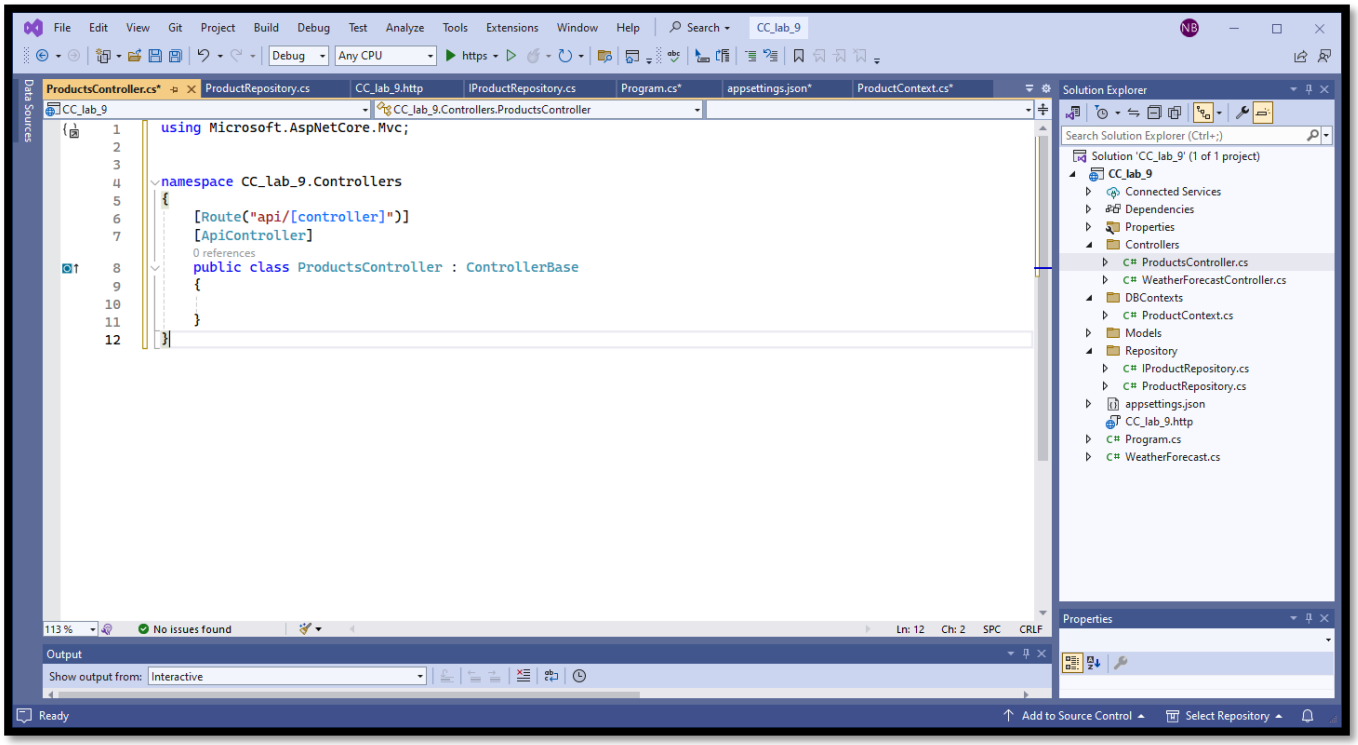
```
    public IEnumerable<Product> GetProducts()
    {
        return _dbContext.Products.ToList();
    }

    public void InsertProduct(Product product)
    {
        _dbContext.Add(product);
        Save();
    }

    public void UpdateProduct(Product product)
    {
        _dbContext.Entry(product).State = EntityState.Modified;
        Save();
    }

    public void Save()
    {
        _dbContext.SaveChanges();
    }

}
```

- Open the Startup class in the project and add the code the following so that the repository's dependency is resolved at a run time when needed:



Step 7: Creating API Controller

- Right-click the *Controllers* folder. Select **Add** > **Controller.** Select **API > Web API 2 Controller – Empty,** add it and name it as **ProductsController**.



Now add the following code in it:

```
public class ProductsController : ControllerBase
{
    private readonly IProductRepository _productRepository;

    public ProductsController(IProductRepository productRepository)
    {
        _productRepository = productRepository;
    }

    [HttpGet]
    public IActionResult Get()
    {
        var products = _productRepository.GetProducts();
        return new OkObjectResult(products);
    }

    [HttpGet("{id}", Name = "Get")]
    public IActionResult Get(int id)
    {
        var product = _productRepository.GetProductByID(id);
        return new OkObjectResult(product);
    }
```

```
        [HttpPost]
        public IActionResult Post([FromBody] Product product)
        {
            using (var scope = new TransactionScope())
            {
                _productRepository.InsertProduct(product); scope.Complete();
                return CreatedAtAction(nameof(Get), new { id = product.Id },
                product);
            }
        }

        [HttpPut]
        public IActionResult Put([FromBody] Product product)
        {
            if (product != null)
            {
                using (var scope = new TransactionScope())
                {
                    _productRepository.UpdateProduct(product); scope.Complete();
                    return new OkResult();
                }
            }
            return new NoContentResult();
        }

        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            _productRepository.DeleteProduct(id); return new OkResult();
        }
    }
```

After following the steps, the microservice is now ready, set to improve system flexibility and growth. Following the same approach, you can also develop microservices using ASP.NET Core.

These microservices can be tested directly by running the application and using Swagger, or they can also be tested using Postman.

2. Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

3. Objectives

After completing this lab the student should be able to:

a. *Grasp the essence of microservices and their crucial role in system architecture.*

b. *Create a microservice using ASP.NET Core Web API.*

4. Lab Tasks/Practical Work

1. Create a microservice for user profile management, including registration, login, and profile updates, using ASP.NET Core Identity.
2. Build a microservice for managing inventory and stock levels of products using ASP.NET.