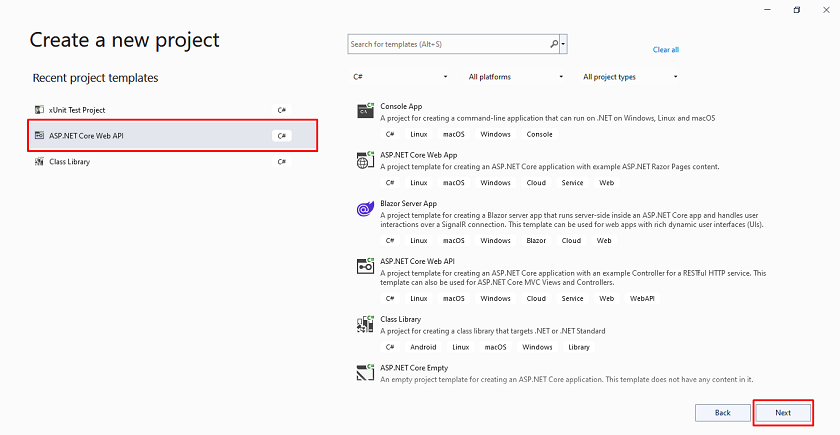
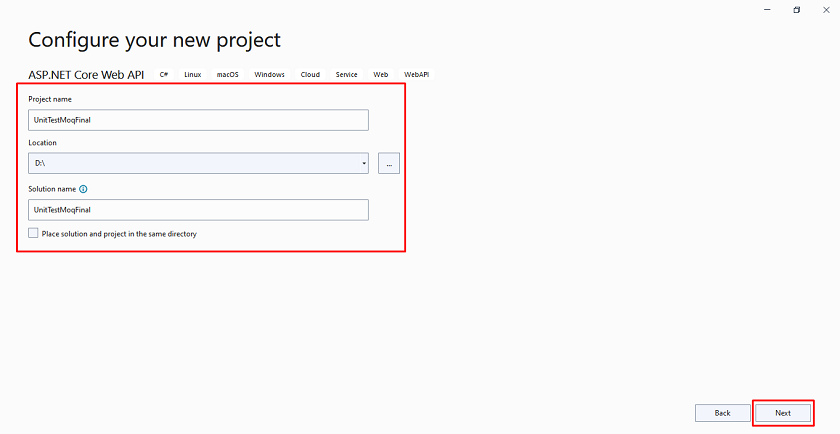
**Step 1**

Create a new .NET Core API Project



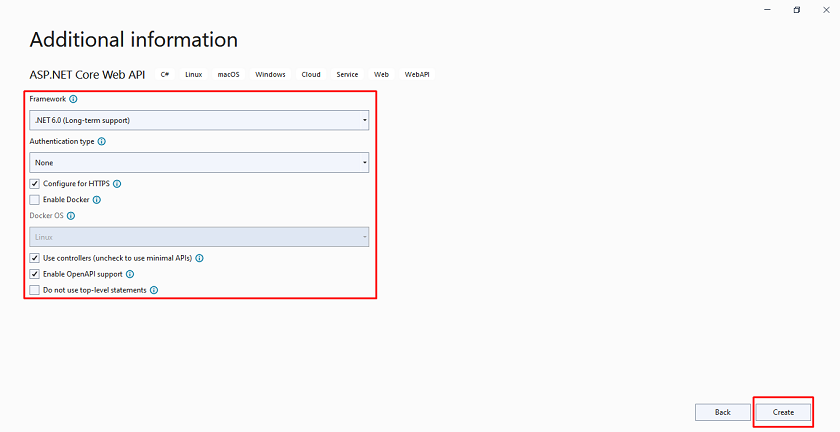
**Step 2**

Configure your project



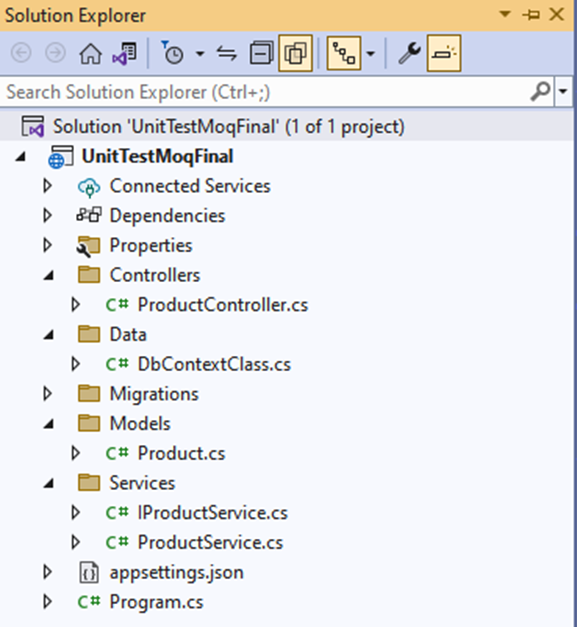
**Step 3**

Provide additional information about your project



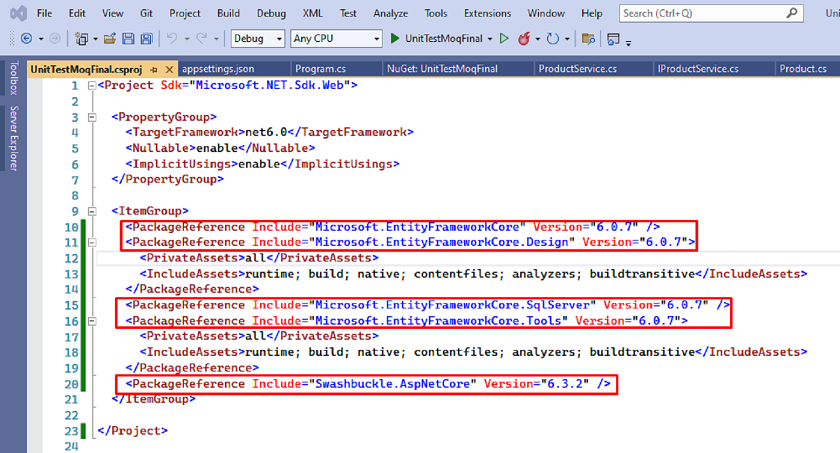
**Step 4**

Project Structure



**Step 5**

Install Following NuGet Packages



**Step 6**

Create the Models folder and create a new class Product

namespace UnitTestMoqFinal.Models

{

public class Product

{

public int ProductId { get; set; }

public string ProductName { get; set; }

public string ProductDescription { get; set; }

public int ProductPrice { get; set; }

public int ProductStock { get; set; }

}

}

**Step 7**

Next, Create DbContextClass inside the Data folder for data manipulation

using Microsoft.EntityFrameworkCore;

using UnitTestMoqFinal.Models;

namespace UnitTestMoqFinal.Data

{

public class DbContextClass : DbContext

{

protected readonly IConfiguration Configuration;

public DbContextClass(IConfiguration configuration)

{

Configuration = configuration;

}

protected override void OnConfiguring(DbContextOptionsBuilder options)

{

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

}

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

}

}

**Step 8**

Later on, Create IProductService and ProductService class for abstraction and dependency injection inside the Services folder.

using UnitTestMoqFinal.Models;

namespace UnitTestMoqFinal.Services

{

public interface IProductService

{

public IEnumerable<Product> GetProductList();

public Product GetProductById(int id);

public Product AddProduct(Product product);

public Product UpdateProduct(Product product);

public bool DeleteProduct(int Id);

}

}

Create ProductService class

using UnitTestMoqFinal.Data;

using UnitTestMoqFinal.Models;

namespace UnitTestMoqFinal.Services

{

public class ProductService : IProductService

{

private readonly DbContextClass \_dbContext;

public ProductService(DbContextClass dbContext)

{

\_dbContext = dbContext;

}

public IEnumerable<Product> GetProductList()

{

return \_dbContext.Products.ToList();

}

public Product GetProductById(int id)

{

return \_dbContext.Products.Where(x => x.ProductId == id).FirstOrDefault();

}

public Product AddProduct(Product product)

{

var result = \_dbContext.Products.Add(product);

\_dbContext.SaveChanges();

return result.Entity;

}

public Product UpdateProduct(Product product)

{

var result = \_dbContext.Products.Update(product);

\_dbContext.SaveChanges();

return result.Entity;

}

public bool DeleteProduct(int Id)

{

var filteredData = \_dbContext.Products.Where(x => x.ProductId == Id).FirstOrDefault();

var result = \_dbContext.Remove(filteredData);

\_dbContext.SaveChanges();

return result != null ? true : false;

}

}

}

**Step 9**

After that, Create a new ProductController

using Microsoft.AspNetCore.Mvc;

using UnitTestMoqFinal.Models;

using UnitTestMoqFinal.Services;

namespace UnitTestMoqFinal.Controllers

{

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

[ApiController]

public class ProductController : ControllerBase

{

private readonly IProductService productService;

public ProductController(IProductService \_productService)

{

productService = \_productService;

}

[HttpGet("productlist")]

public IEnumerable<Product> ProductList()

{

var productList = productService.GetProductList();

return productList;

}

[HttpGet("getproductbyid")]

public Product GetProductById(int Id)

{

return productService.GetProductById(Id);

}

[HttpPost("addproduct")]

public Product AddProduct(Product product)

{

return productService.AddProduct(product);

}

[HttpPut("updateproduct")]

public Product UpdateProduct(Product product)

{

return productService.UpdateProduct(product);

}

[HttpDelete("deleteproduct")]

public bool DeleteProduct(int Id)

{

return productService.DeleteProduct(Id);

}

}

}

**Step 10**

Add connection string inside app setting file

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Data Source=DESKTOP-Server;Initial Catalog=UnitTestMoqFinal;User Id=\*\*\*;Password=\*\*\*;"

}

}

**Step 11**

Next, register a few services inside Program Class

using UnitTestMoqFinal.Data;

using UnitTestMoqFinal.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddScoped<IProductService, ProductService>();

builder.Services.AddDbContext<DbContextClass>();

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at <https://aka.ms/aspnetcore/swashbuckle>

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Step 12**

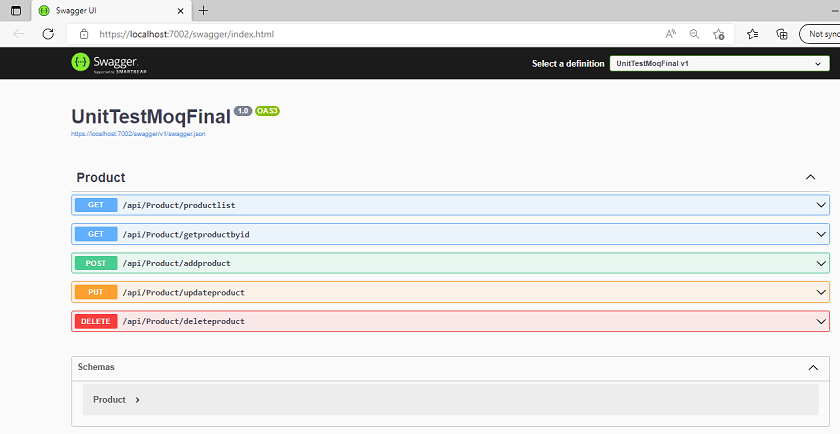
Add migrations and update the database using the following entity framework command after executing that into the package manager console under the main project

add-migration "First"

update-database

**Step 13**

Finally, run your application and you will see swagger UI and API endpoints

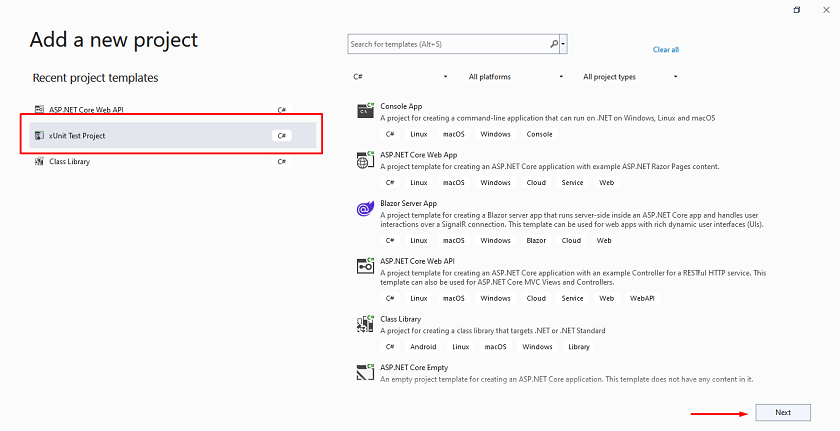


This is all about the .NET Core Web API, Let’s create a new Xunit project inside which we use Moq to write the test cases.

Create a new Xunit project

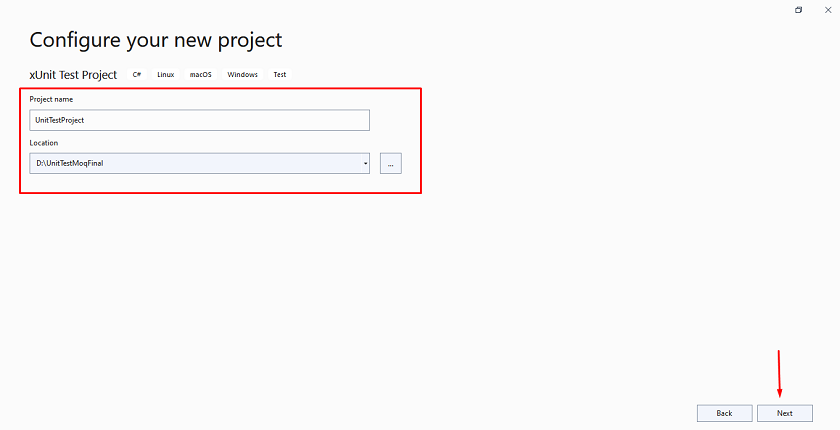
**Step 1**

Add a new Xunit project inside the existing solution



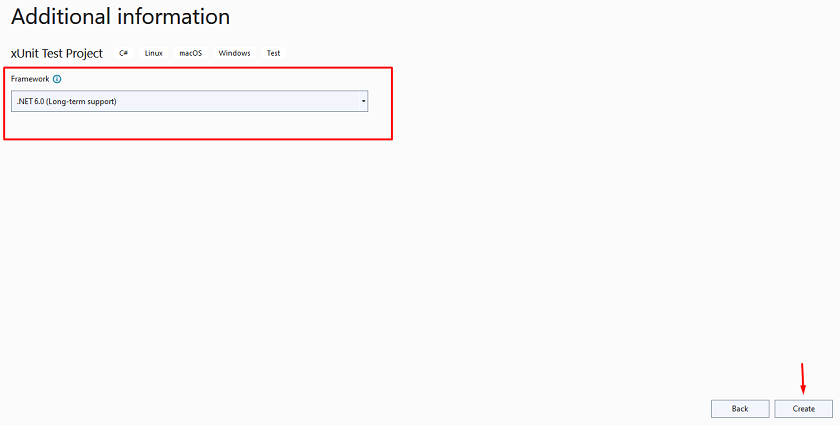
**Step 2**

Configure your new project



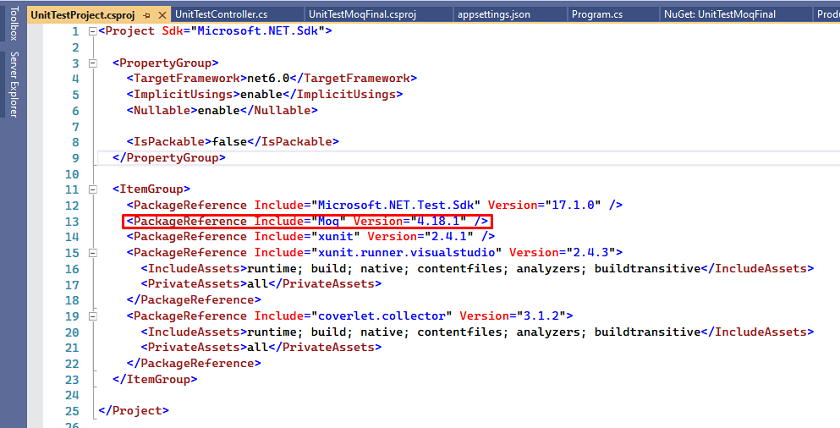
**Step 3**

Provide some additional information



**Step 4**

Install Moq NuGet Package for mocking purpose



**Step 5**

Create UnitTestController Class

using Moq;

using UnitTestMoqFinal.Controllers;

using UnitTestMoqFinal.Models;

using UnitTestMoqFinal.Services;

namespace UnitTestProject

{

public class UnitTestController

{

private readonly Mock<IProductService> productService;

public UnitTestController()

{

productService = new Mock<IProductService>();

}

[Fact]

public void GetProductList\_ProductList()

{

//arrange

var productList = GetProductsData();

productService.Setup(x => x.GetProductList())

.Returns(productList);

var productController = new ProductController(productService.Object);

//act

var productResult = productController.ProductList();

//assert

Assert.NotNull(productResult);

Assert.Equal(GetProductsData().Count(), productResult.Count());

Assert.Equal(GetProductsData().ToString(), productResult.ToString());

Assert.True(productList.Equals(productResult));

}

[Fact]

public void GetProductByID\_Product()

{

//arrange

var productList = GetProductsData();

productService.Setup(x => x.GetProductById(2))

.Returns(productList[1]);

var productController = new ProductController(productService.Object);

//act

var productResult = productController.GetProductById(2);

//assert

Assert.NotNull(productResult);

Assert.Equal(productList[1].ProductId, productResult.ProductId);

Assert.True(productList[1].ProductId == productResult.ProductId);

}

[Theory]

[InlineData("IPhone")]

public void CheckProductExistOrNotByProductName\_Product(string productName)

{

//arrange

var productList = GetProductsData();

productService.Setup(x => x.GetProductList())

.Returns(productList);

var productController = new ProductController(productService.Object);

//act

var productResult = productController.ProductList();

var expectedProductName = productResult.ToList()[0].ProductName;

//assert

Assert.Equal(productName, expectedProductName);

}

[Fact]

public void AddProduct\_Product()

{

//arrange

var productList = GetProductsData();

productService.Setup(x => x.AddProduct(productList[2]))

.Returns(productList[2]);

var productController = new ProductController(productService.Object);

//act

var productResult = productController.AddProduct(productList[2]);

//assert

Assert.NotNull(productResult);

Assert.Equal(productList[2].ProductId, productResult.ProductId);

Assert.True(productList[2].ProductId == productResult.ProductId);

}

private List<Product> GetProductsData()

{

List<Product> productsData = new List<Product>

{

new Product

{

ProductId = 1,

ProductName = "IPhone",

ProductDescription = "IPhone 12",

ProductPrice = 55000,

ProductStock = 10

},

new Product

{

ProductId = 2,

ProductName = "Laptop",

ProductDescription = "HP Pavilion",

ProductPrice = 100000,

ProductStock = 20

},

new Product

{

ProductId = 3,

ProductName = "TV",

ProductDescription = "Samsung Smart TV",

ProductPrice = 35000,

ProductStock = 30

},

};

return productsData;

}

}

}

* Here, you can see first we add reference of our main project into the current unit test project
* We mock IProductService and create an instance inside the constructor
* Next, we write one test case which takes a list of a product
* Later on, we take a list of products from our custom method which is present in the same class at the bottom
* Next, set up the list of products for the product service with some mock data
* Also, our product controller is dependent on product service and because of that, we pass the product service object inside the product controller constructor to resolve some dependencies.
* In the act section, we call the ProductList method of the controller.
* Finally, in the assert section, we check actual and expected results using a few conditions

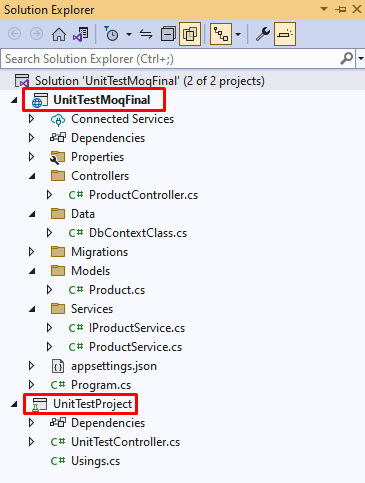
Similarly, all the test cases are worked step by step

**Step 6**

Next, go to the Test Section at the top of Visual Studio and open the Test Explorer inside that you can see all the test cases which we write inside the UnitTestControllerClass

**Step 7**

Final Project Structure



**Step 8**

Finally, run your test cases and check if they will be worked properly or not, also if you want to debug a test case simply right-click on the test case and click on debug after attaching the debugger point inside the test case

