What are available tools for migration

Migrate .Net Web API2 to .Net Core Web API

WCF to gRPC Migration--

Migrating EF6 to EF

Demo using Dotnet Try Convert.

.Net Standard.

What is Unit Testing & Integration Testing

XUnit

Unit Testing Controller

Unit Testing Models

Migration

<https://docs.microsoft.com/en-us/dotnet/core/porting/>

* Porting from .NET to .NET is relatively straightforward
* Complexity of Project dictates how much work you will do after the initial migration of the project files.
* Projects where app model is available in .Net Core Such as libraries, Console apps and desktop apps usually require little change
* Project that require a new app model such as moving to ASP.Net Core from ASP.Net require more work.

Unavailble Tech

1.Remoting

2. CAS

3. WF and WCF

Consider following thing before migrating to Windows Forms

1. Project Files for .Net Core use a different format than .Net framework
2. Your project may use an API that isn’t available in .Net Core
3. Third party controls and libraries may not have been ported to .NET and remain only available to .NET framework.
4. Your project uses a technology that is no longer available in .NET

**Migration Tools Available for Porting application to .Net Core**

1. .Net Portability Analyzer

* ToolChain that can generate a report of how portable your code is between .Net and .Net Core

It can be used in 2 ways

1. Command line tool
2. Visual Studio Extension
3. Platform Compatibility analyzer

* he analyzer identifies all APIs that are not cross-platform
* Inform developers when they use platform specific APIS from call sites where the API might not be available.
* It identifies all APIS that are not cross platform.

For Example :

Console.WindowWidth works on Windows but not on Linux and macos.

<https://docs.microsoft.com/en-us/dotnet/standard/analyzers/api-analyzer>

1. .Net API analyzer

* Help detect platform compatibility issues in cross platform apps and libraries.
* .Net API analyses whether or not you are using an API that will throw PlatformNotSupportException at run time.
* It discovers Potential compatibilities risks for C# APIS on different platforms and detects calls to deprecated APIs
* Use ful for C# dev during dev
* This is the APIS that all are not supported in .Net Core

<https://docs.microsoft.com/en-us/dotnet/core/compatibility/unsupported-apis>

* Its available as Nuggets package MicrosoftDotNet.Analyzers.Compability
* When you add it in project it automatically monitors code and indicates problematic API usage.
* Its still in pre-release version.

1. Try-Convert :

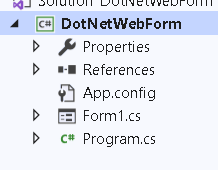
* Tool that can convert a project or entire solution the .Net SDK.
* IT can move desktop apps to .Net Core.
* Its may reject many project types that are incompatible with .NET core.
* This tool is not recommended if your project has custom tasks, targets or imports.
* This tool work only on windows.
* Class libraries or code with no platform specific code , this tool will help a lot.
* How it works : <https://github.com/dotnet/try-convert>

**Demo 1: How to Migrate Application From Windows Forms to .Net Core Windows App**

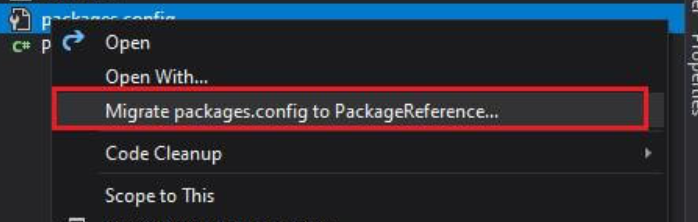
[**https://docs.microsoft.com/en-us/dotnet/desktop/winforms/migration/?view=netdesktop-5.0&preserve-view=true**](https://docs.microsoft.com/en-us/dotnet/desktop/winforms/migration/?view=netdesktop-5.0&preserve-view=true)

**Migrating .Net Windows App to .Net Core App**

1. **Go to DornetwindowsForm Application**

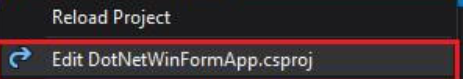
****

1. **Install Entity Framework**
2. **Right Click on Packages reference and Select Migrate Option**

****

1. **Packages information will be move to .csproject so this file will not be available**
2. **Right Click and unload the project and Select EditProject**

****

****

1. **Cut the content from this file and keep it in notepad**
2. **Paste this 4 settings in the file**

**<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">**

**<PropertyGroup>**

**<OutputType>WinExe</OutputType>**

**<TargetFramework>netcoreapp3.1</TargetFramework>**

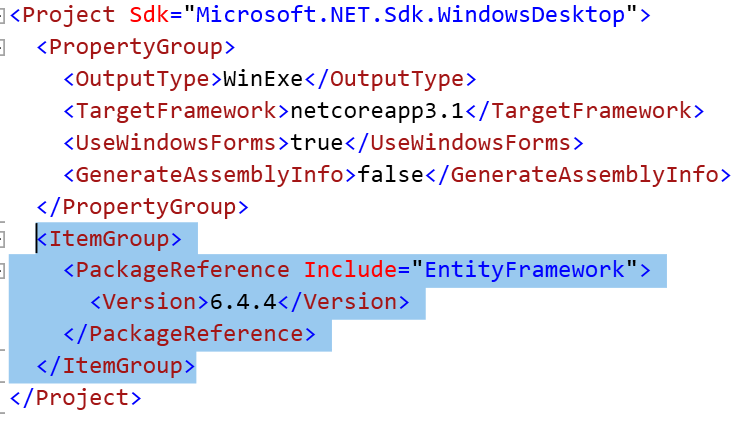
**<UseWindowsForms>true</UseWindowsForms>**

**<GenerateAssemblyInfo>false</GenerateAssemblyInfo>**

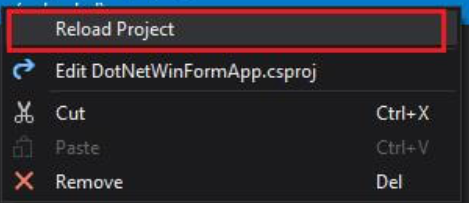
**</PropertyGroup>**

**</Project>**

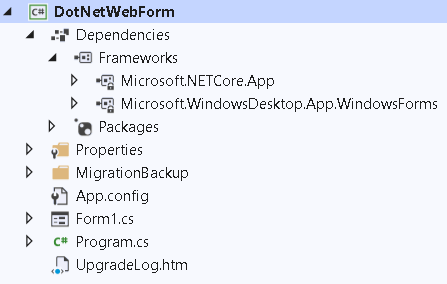
1. **Add a Packages information into it**

****

1. **Reload the Project**

****

1. **Check the References**

****

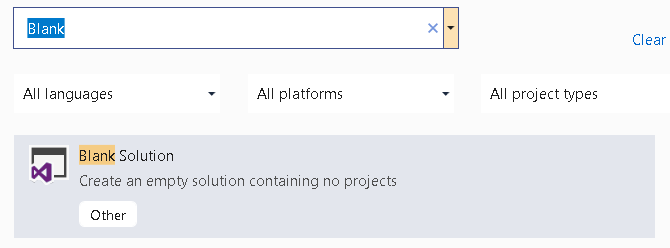
Demo 2: Migration From Web API 2 to .Net Core Web API

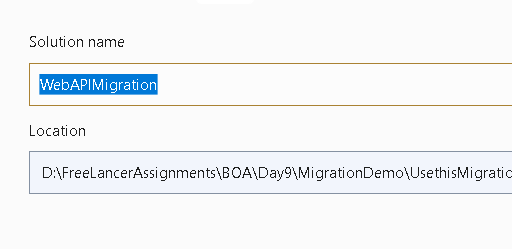
Migrating .Net Web API to .Net Core API : [Migrate from ASP.NET Web API to ASP.NET Core | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/migration/webapi?view=aspnetcore-5.0)

1. Open ProductWebAPI application

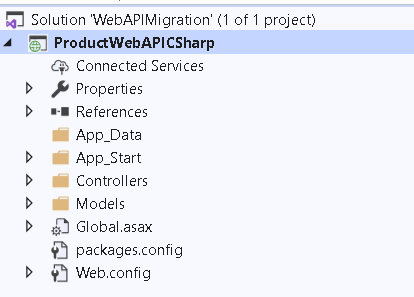
D:\FreeLancerAssignments\BOA\Day9\MigrationDemo\UsethisMigration\ProductWebAPICSharp

1. Run this and type <http://localhost/api/products>
2. Migrate this application to ASP.Net Core
3. Open Visual Studio and Create new Blank Solution Name it as WebAPI Migration

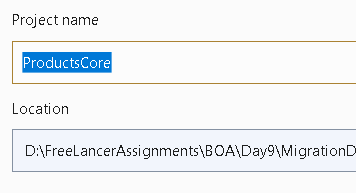


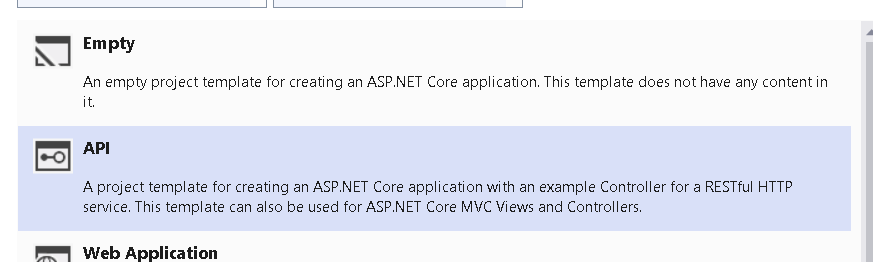


1. Add Existing Project

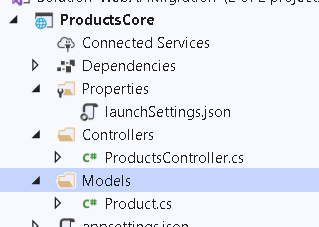


1. Add New ASP.Net Core Project and add API and Name it as ProductCore





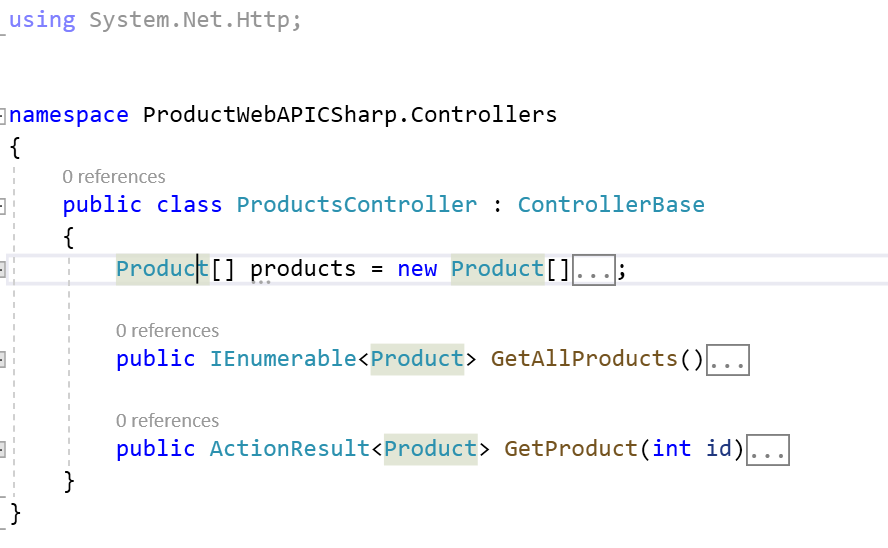
1. Remove Weather Related files
2. Copy Controller and product model from the API to core project



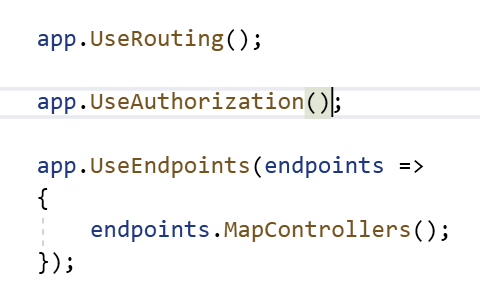
1. Go to ProductController

Following thing don’t exist in ASP.Net core so remove it and replace

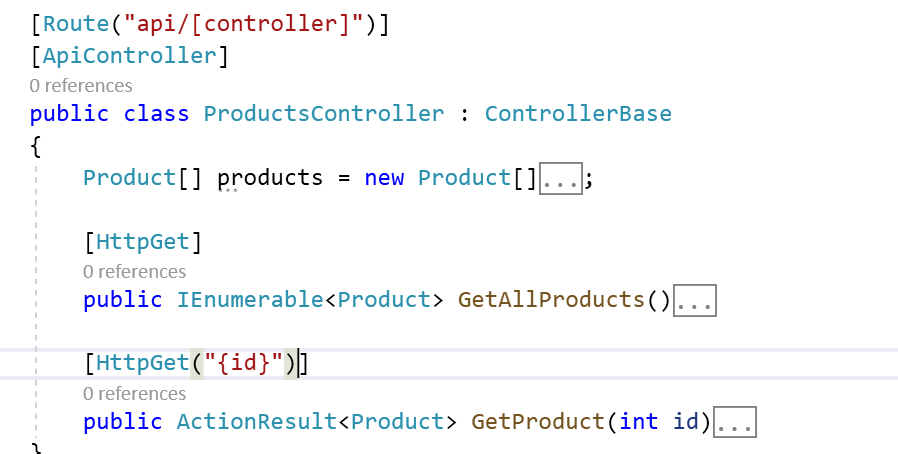
1. APIController = ControllerBase
2. System.Web.Http = using Microsoft.AspNetCore.Mvc
3. IHttpActionResult Interface :ActionResult<Product>
4. It look like this



1. Replace the Routing Configuration : ASP.Net Core doesn’t include webapi config so configure routes for the same in ASP.Net Core



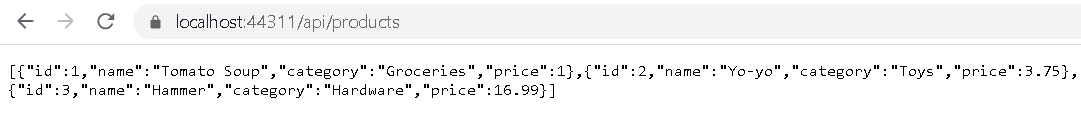
1. Mark the ProductsController with following attributes. Route , HttpGet and ApiController



1. Change the Launch Setting



1. Check the Output



.Net Migration From WCF to gRPC application.

* Although Implementation and approach are different for gRPC , the experience of developing and consuming services with gRPC is same as WCF.

What Benefits you will get if you port to gRPC

1. Performance

* gRPC uses HTTP2
* Smaller , Faster binary protocol.
* Support multiplexing

1. Interoperability

* gRPC tools and libraries present for all major programming language.

1. Streaming : Full bi-directional support

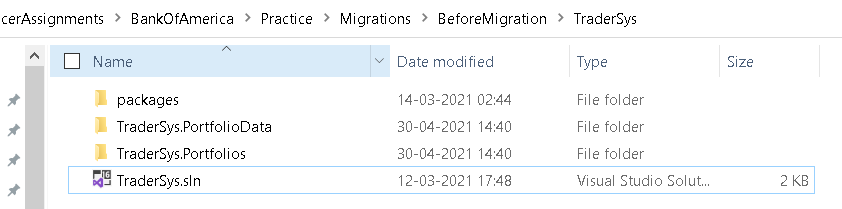
Download the Sample application from this link : [Migrate a WCF solution to gRPC - gRPC for WCF developers | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/architecture/grpc-for-wcf-developers/migrate-wcf-to-grpc)

How to Migrate Existing WCF to gRPC

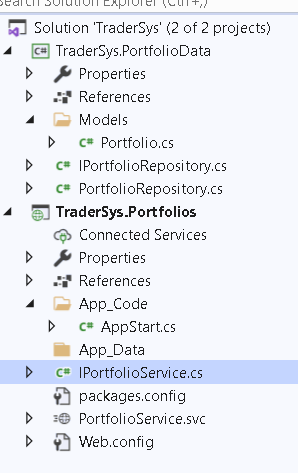
1. Create Proto file
2. Convert DataContract classes to gRPC Messages
3. Convert ServiceContract to gRPC service
4. Use ASP.Net Core DI
5. Implement gRPC Service

How to Perform Migrations

1. Check the WCF application stored in Before Migration folder



1. Check the Application Structure



1. So As Per Guideline Convert All the DataMembers into Proto,ServiceContract to Grpc Services
2. Create a new folder AfterMigration
3. Go to Command Prompt and Select that folder
4. Create new Solution

dotnet new sln -o TraderSys

cd TraderSys

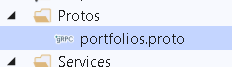
1. . Create the new project by using this template, putting it into an src subdirectory as is conventional for ASP.NET Core projects

dotnet new grpc -o src/TraderSys.Portfolios

1. Finally, add the project to the solution by using the dotnet sln command:

dotnet sln add src/TraderSys.Portfolios

1. Rename proto file to Portfolios.proto



1. Change the code to following

syntax = "proto3";

option csharp\_namespace = "TraderSys.Portfolios";

package PortfolioServer;

service Portfolios {

// RPCs will go here

}

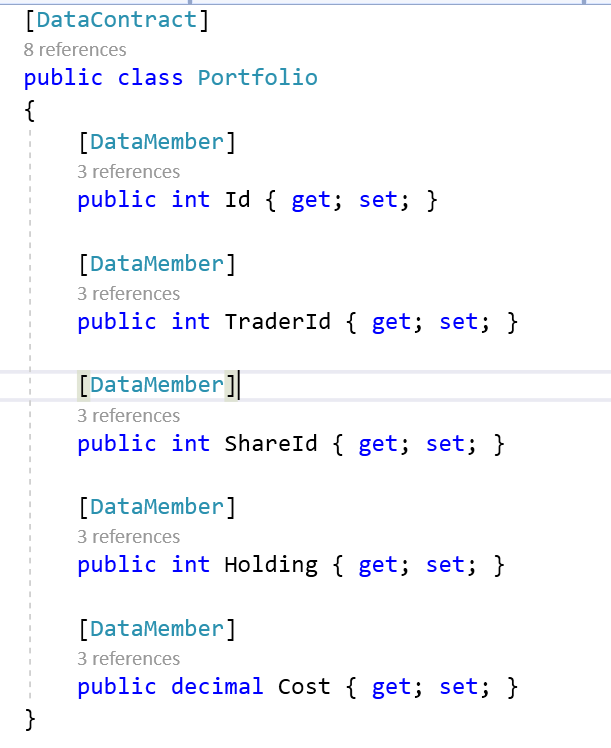
1. Rename Greeter Service to Portfolio Service



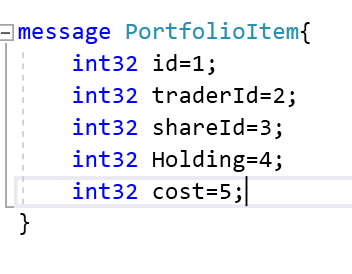
1. Go to Startup.cs and Change GreeterService to this new service



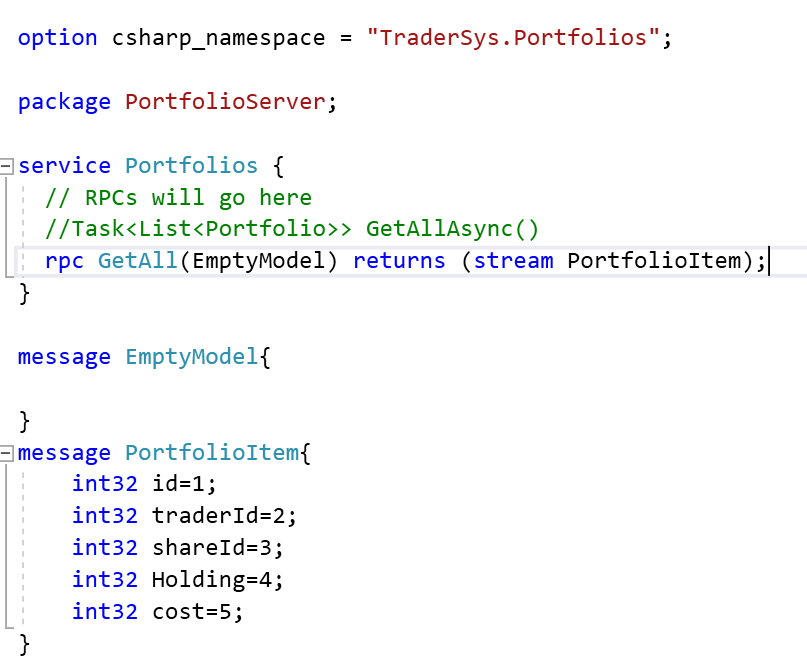
1. Convert following DataContract to gRPC Message



1. New Proto Messages are as below



1. Convert Service Contract to GRPC Service
2. Convert above into rpcServices

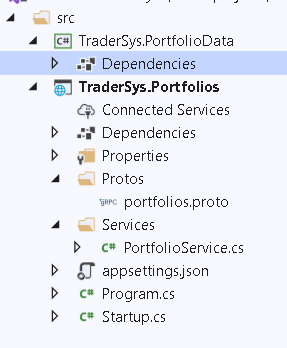


1. Project Needs the Library Now
2. Create New Project class library

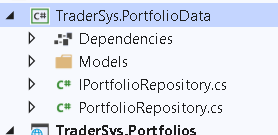
dotnet new classlib -o src/TraderSys.PortfolioData

dotnet sln add src/TraderSys.PortfolioData

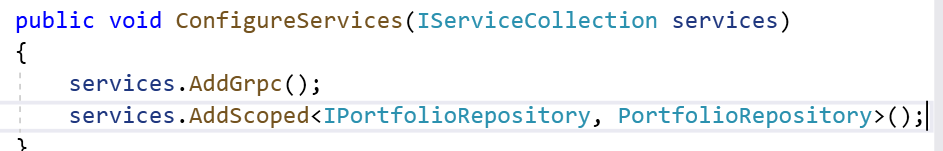
1. This will create following proj structure ,Delete automatically generated class libraries



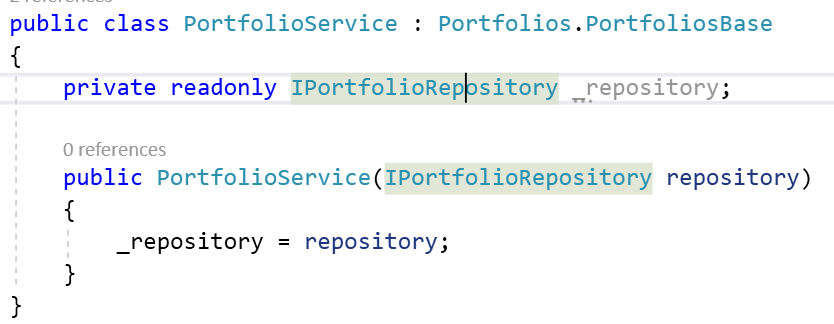
1. Copy and Paste the model and Repository files from the WCF solution to this new library project



1. Remove the unnecessary attribute from the model classes
2. Add a ref of this library into gRPC project
3. Register this Repository classes in Startup.cs



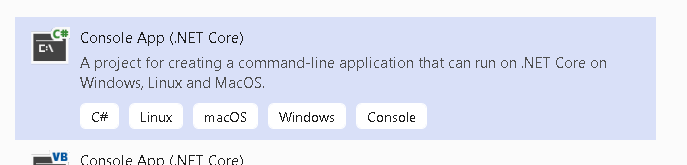
1. Inject Service to Portfolio Service



1. Implement Service



1. Create Client : TraderClient



1. Add a Ref of Client Lib

* Google.Protobuf
* Grpc.Net.Client
* Grpc.Tools

1. Make sure proto file is set to valid compiler and target to client only
2. Add following code in



1. **Check the Output**

**Using Try Convert**

**Try-Convert**

1. **Try-Convert :**

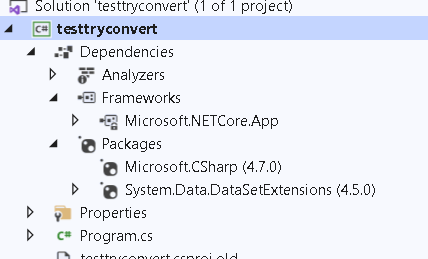
* **Tool that can convert a project or entire solution the .Net SDK.**
* **IT can move desktop apps to .Net Core.**
* **Its may reject many project types that are incompatible with .NET core.**
* **This tool is not recommended if your project has custom tasks, targets or imports.**
* **This tool work only on windows.**
* **Class libraries or code with no platform specific code , this tool will help a lot.**
* **How it works :** <https://github.com/dotnet/try-convert>

**Demo : how to use try Convert :** [GitHub - dotnet/try-convert: Helping .NET developers port their projects to .NET Core!](https://github.com/dotnet/try-convert)

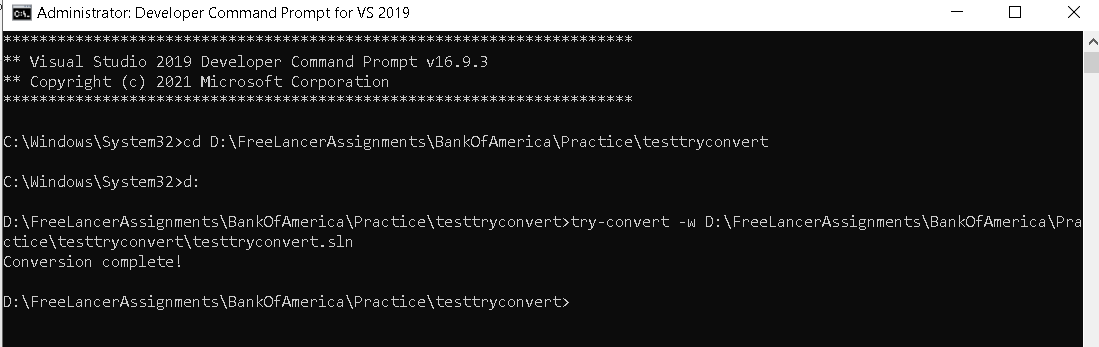
1. **Install try convert globally**

**dotnet tool update -g try-convert**

1. **Create .Net Framework Application**

****

1. **Open Developer Command Prompt for Visual Studio in administrator mode.**

****

1. **Check the Output**

**===================================== ===========================**

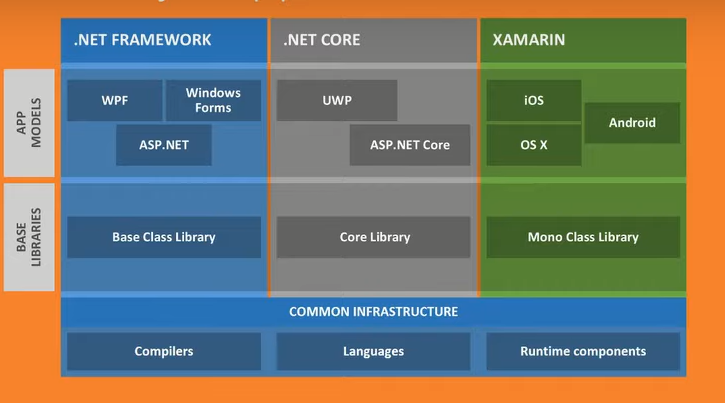
[.NET Standard | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/net-standard)

[NuGet Gallery | NETStandard.Library 2.0.3](https://www.nuget.org/packages/NETStandard.Library/)

.Net Standard

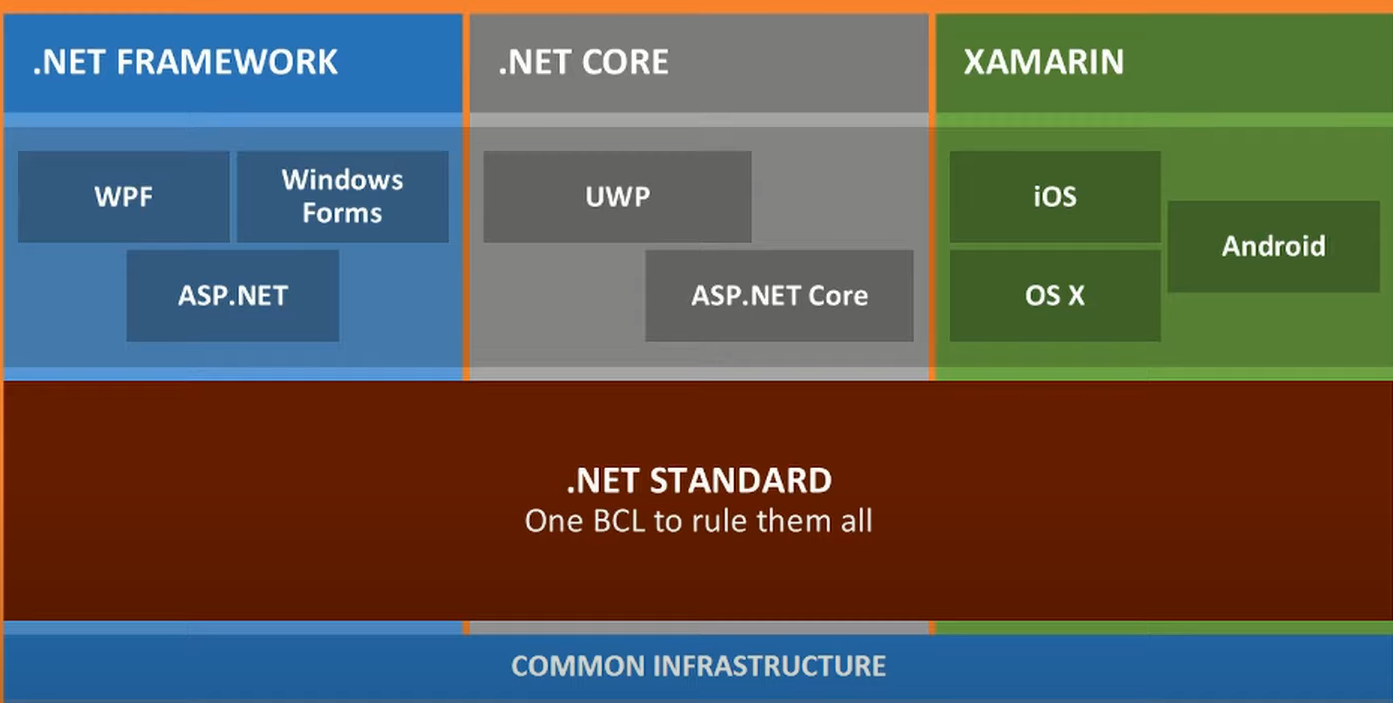
* .Net Core Cross Platform Implementation of .NET framework
* .Net Standard is the Set of API that is available on all .Net implementation. It will create some type of uniformness,
* Portability that supports .Net Core, Xamarin and .Net Framework
* Its Base class libraries that support wide range of technologies.
* Use .Net Standard library when you want to increase the number of applications. That will be compatible with your library..
* Solves the code sharing problem for .NET developers across all platforms by bringing all the APIS that you expect and love across the environment you need , desktop applications, mobile apps and games.

This is how it use to be



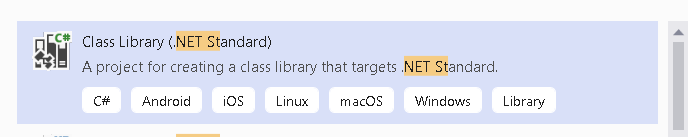
* Need to master 3+1 Base class libraries.

After .NetStandard.

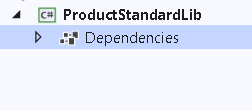
* 

Demo : .Net Standard

1. Create new .net standard library



1. Remove Class



1. Add a Code for the same

public List<Product> GetProducts()

{

return new List<Product> { new Product

{

Id=1,Name="P1"

} ,

new Product

{

Id=2,

Name="P2"

}

};

}

1. Create console application
2. Add a reference of the library
3. Consume and Run

ProductStandardLib.ProductRepository repository = new ProductStandardLib.ProductRepository();

var data = repository.GetProducts();

foreach (var item in data)

{

Console.WriteLine(item.Name);

}

Console.WriteLine(".Net Framework");

Console.ReadKey();

1. Check the Output

Unit Testing

* Form of Testing in which individual components are isolated from the rest of the application so their behavior can be thoroughly validated.
* ASP.Net has been designed to make it easy to create unit tests and there is support for wide range of unit test frameworks.
* Unit test helps to test every small unit of the source code.
* It used to mock the dependency of the code so that the individual unit of the code is tested separately without including the dependency in the test code.
* Unit test case do not detect issues in the interaction between components which is the purpose of integration testing.

Moq Object

Unit Test Of Code

How Many unit Test Cases , we should write

1. Ideally should be equal to the number of methods available in the application.
2. If we have conditional statement in these methods then we have to write on test case per conditional statement.
3. This is why unit tests are responsibility of the developer so that if TDD is understood by the developer then the code can be bug free.

Why Testing Controllers : [Testing Controller Logic — ASP.NET documentation (jakeydocs.readthedocs.io)](https://jakeydocs.readthedocs.io/en/latest/mvc/controllers/testing.html)

* Controllers are central part of any ASP.Net Core MVC application
* Automated tests can provide you confidence and can detect errors before they reach production,
* Controller logic should be minimal and should not focused on business logic like data access.
* In this we test how controller behaves based on invalid inputs . Test response based on the result of the business it performs.
* A controller unit test avoid things like filters, routing etc and model binding.

What needs to be tested

1. Verify ModelState.IsValid
2. Return Error Response if ModelState is invalid
3. Retrieve a business entity from persistence.
4. Return an appropriate IactionResult.

Creating unit test project

* For ASP.Net generally you create a separate visual studio project to hold unit tests
* Which is defined as a method in a C# class.
* Separate project means you can deploy your application without also deploying the tests.

Project Tools

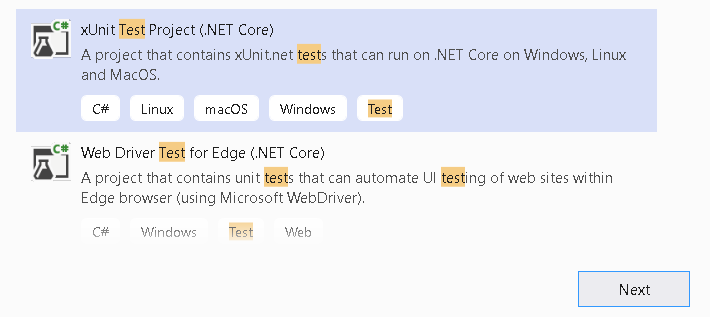
1. MS Test
2. NUnit
3. XUnit

XUnit

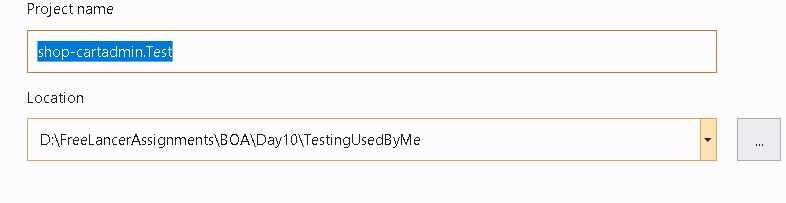
* Free , Open Source unit testing tool for .Net Framework.
* Written by Original inventor of NUnitv2
* XUnit .net is the latest technology for unit testing C#, F#..
* xUnit works better with Resharper, CodeRush, TestDriven.Net and Xamarin.
* xUnit is available as nugets also.

Demo 1: How to Create Unit Test Project and test model

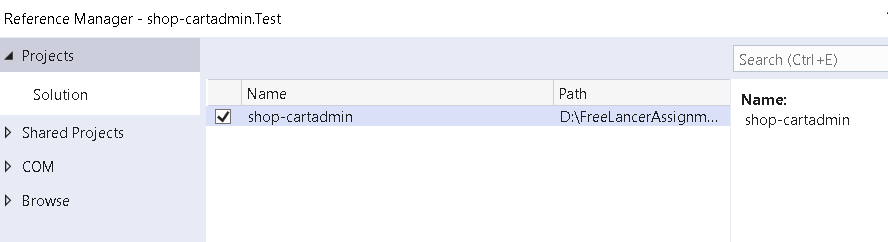
1. Right Click on Shop Cart Admin , Add Test Project



1. Add Following Project



1. Add a Ref of the project that needs to be tested



1. Lets write some simple test case

Name Of the Test Method describes what test does

Name of the class describes whats being test

**Fact** : Applied to each method to indicate it’s a test.

Test Body Follows Pattern

1. **Arrange** : Setting up the conditions for the rest
2. **Act** : Performing the test
3. **Assert** : Verifying the test result. Its handled by xUnit.Net. It’s a static class .
4. **Equal**
5. **NotEqual**
6. **True**
7. **False**
8. **Istype**
9. **ISNotType**
10. **IsNull**
11. **InRange**
12. **NotInRange**
13. **Throw(Exception)**

public class ProductTests

{

[Fact]

public void CanChangeProductName()

{

// Arrange

var p = new Product

{

Name = "Product1",

Price = 32.3M

};

// Act

p.Name = "IPhone";

// Assert

Assert.Equal("IPhone", p.Name);

}

[Fact]

public void CanChangeProductPrice()

{

// Arrange

var p = new Product

{

Name = "Product1",

Price = 32.3M

};

// Act

p.Price = 13.3M;

// Assert

Assert.Equal(13.3M, p.Price);

}

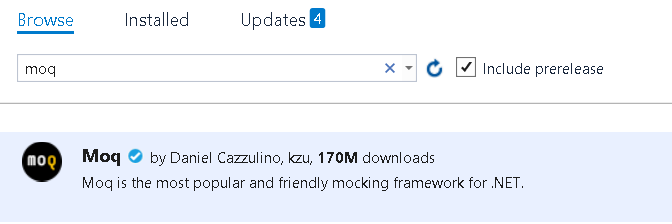
}

Mocking Package

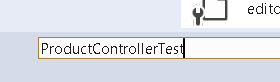
* Classes for which fake implementations are required are more complex and cannot be handled as easily. Better approach is to use a mocking package. Which makes it easy to create fake or mock objects for tests.

Demo : How to Test Controller using Mock Package.

1. Add moq package in Test project



1. Add Test Class for Controller



1. Add a Code in ProductController



2. Add a unit test code for the same

public class ProductControllerTest

{

[Fact]

public void Can\_Use\_Repository()

{

// Arrange

Mock<IStoreRepository> mock = new Mock<IStoreRepository>();

mock.Setup(m => m.Products)

.Returns((new Product[]

{ new Product{ProductID = 1,Name="P1"},

new Product{ProductID =2, Name="P2"}

}).AsQueryable<Product>());

ProductController productController = new ProductController(mock.Object);

// Act

IEnumerable<Product> result =

(productController.Index()

as ViewResult).ViewData.Model

as IEnumerable<Product>;

// Assert

Product[] prodArray = result.ToArray();

Assert.True(prodArray.Length == 2);

Assert.Equal("P1", prodArray[0].Name);

Assert.Equal("P2", prodArray[1].Name);

}

}

1. Run the Test Explorer.

Demo: Testing Pagination

1. Add Action Method to Controller

public ViewResult IndexWithPagination(int productPage)

{

return View(\_repository.Products

.OrderBy(p => p.ProductID)

.Skip((productPage - 1) \* PageSize)

.Take(PageSize));

}

1. Right a Test case for Pagination

[Fact]

public void Can\_Paginate()

{

// Arrange

Mock<IStoreRepository> mock = new Mock<IStoreRepository>();

mock.Setup(m => m.Products)

.Returns((new Product[]

{

new Product{ProductID = 1,Name="P1"},

new Product{ProductID =2, Name="P2"},

new Product{ProductID = 3,Name="P3"},

new Product{ProductID =4, Name="P4"},

new Product{ProductID =5,Name="P5"},

new Product{ProductID =6, Name="P6"}

}).AsQueryable<Product>());

ProductController productController = new ProductController(mock.Object);

productController.PageSize = 2;

// Act

var result = (productController.IndexWithPagination(2)

as ViewResult)

.ViewData.Model as IEnumerable<Product>;

// Assert

Product[] prodArray = result.ToArray();

Assert.True(prodArray.Length == 2);

Assert.Equal("P3", prodArray[0].Name);

Assert.Equal("P4", prodArray[1].Name);

}

1. Run the Test Explorer.

Testing Components

We want to check whether our component is capable of reading data from route and set in the ViewData or not.

1. Add Unit Test case for the NavigationMenu

[Fact]

public void Check\_Navigation()

{

// Arrange

string categoryToSelect = "Cricket";

Mock<IStoreRepository> mock = new Mock<IStoreRepository>();

mock.Setup(m => m.Products)

.Returns((new Product[]

{

new Product{ProductID = 1,Name="P1",Category="Cricket"},

new Product{ProductID =2, Name="P2",Category="Cricket"},

new Product{ProductID = 3,Name="P3",Category="Cricket"},

new Product{ProductID =4, Name="P4",Category="Cricket"},

new Product{ProductID =5,Name="P5",Category="Chess"},

new Product{ProductID =6, Name="P6",Category="Chess" }

}).AsQueryable<Product>());

NavigationMenuViewComponent target =

new NavigationMenuViewComponent(mock.Object);

target.ViewComponentContext = new ViewComponentContext

{

ViewContext = new ViewContext

{

RouteData = new Microsoft.AspNetCore.Routing.RouteData()

}

};

target.RouteData.Values["category"] = categoryToSelect;

// Action

string result = (string)(target.Invoke() as ViewViewComponentResult).ViewData["SelectedCategory"];

// Assert

Assert.Equal(categoryToSelect, result);

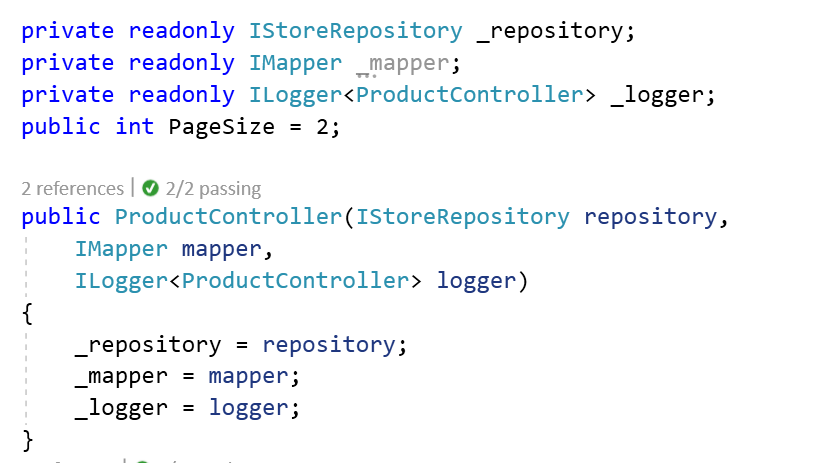
}

Testing Filters

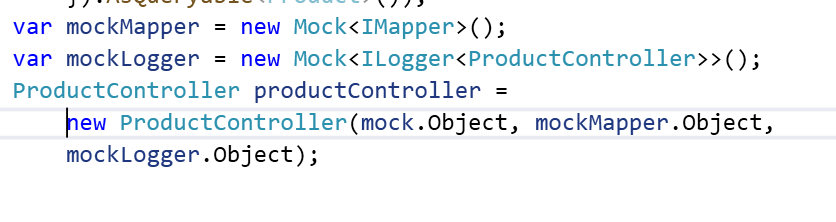
1. Check the Test Case.

Demo : How to Add Other mocks like ILogger and IMapper

1. Add dI in the controller



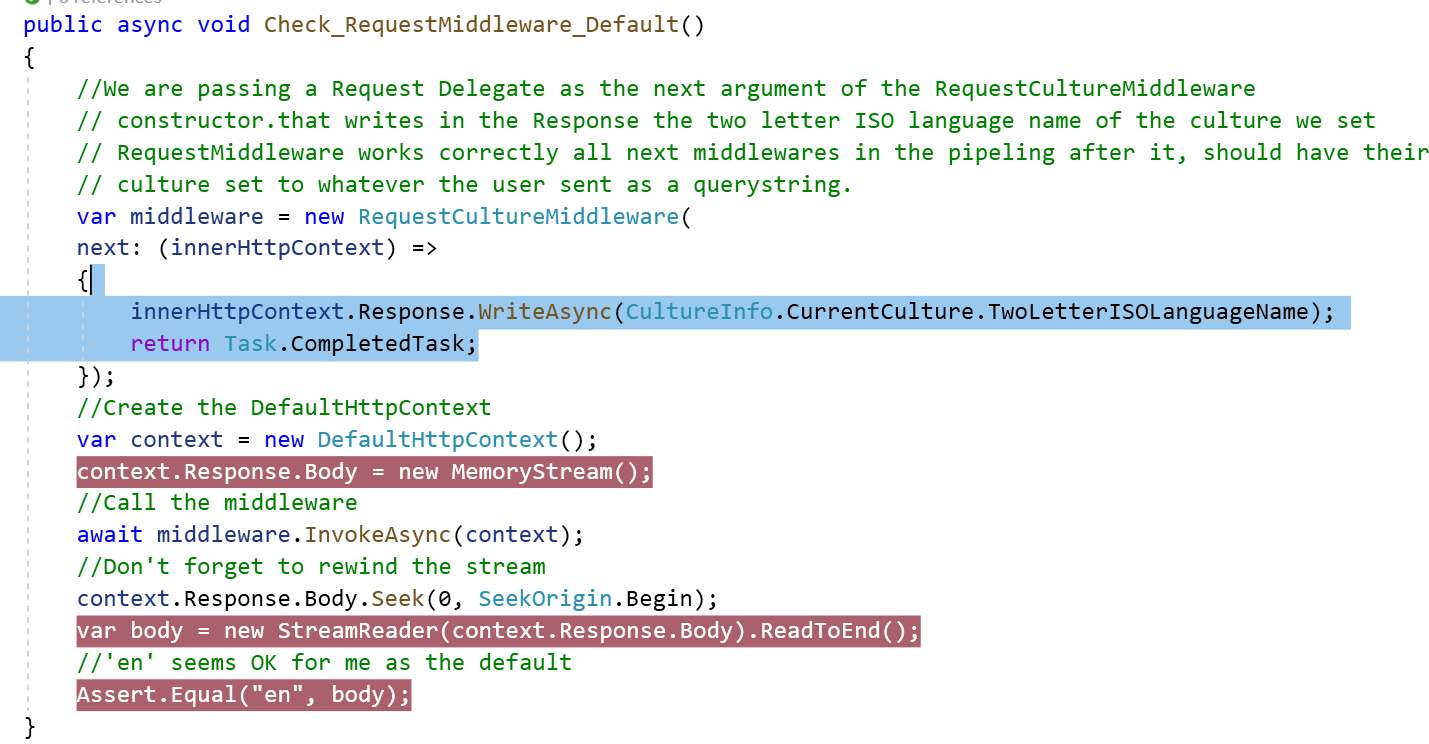
1. Go to ProductControllerTest and add DI for the same



1. Check the Output

**Demo : Testing Middleware**

1. **Add Middleware Testing for RequestCulture Middleware**



1. **Add a QueryString in the Culture information.**



1. **Go to test explorer and run it**

**=========================================================================================================**

**Integration Testing**

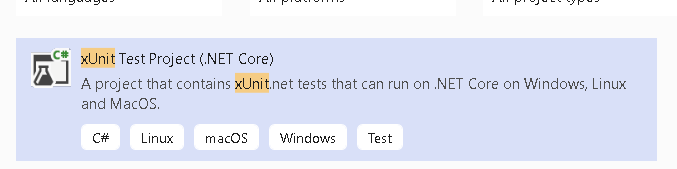
* Verifies different parts of an application work correctly together .
* It involves application infrastructure concerns such as DB, file system ,network resources or web requests and response.
* Testing the logic within your own methods is usually the domain of unit tests, how your application works within its framework or with database is where integration come into play.

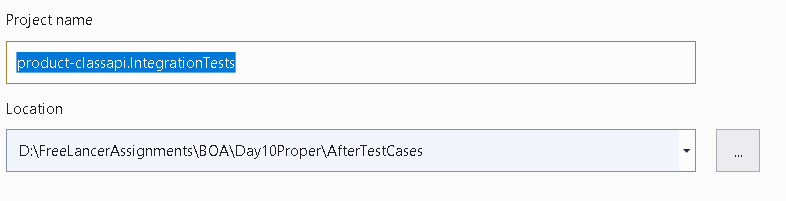
Problem

* Limit how many integration tests you write as it contain larger segments of code due to which its slower than unit tests.

Demo : ProductClass API

1. Use Product API Created
2. Add New xUnit Project





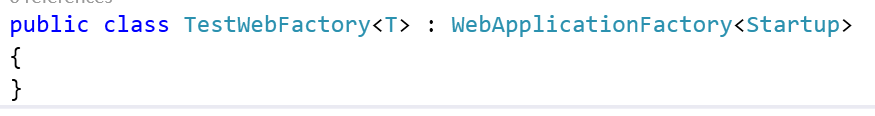
1. Add Following References

* AspNetCore.Mvc.Testing – this package provides the TestServer and an important class WebApplicationFactory to help us bootstrap our app in-memory
* Microsoft.EntityFrameworkCore.InMemory – In-memory database provider

1. Add a Ref Of the Project that needs to be tested.



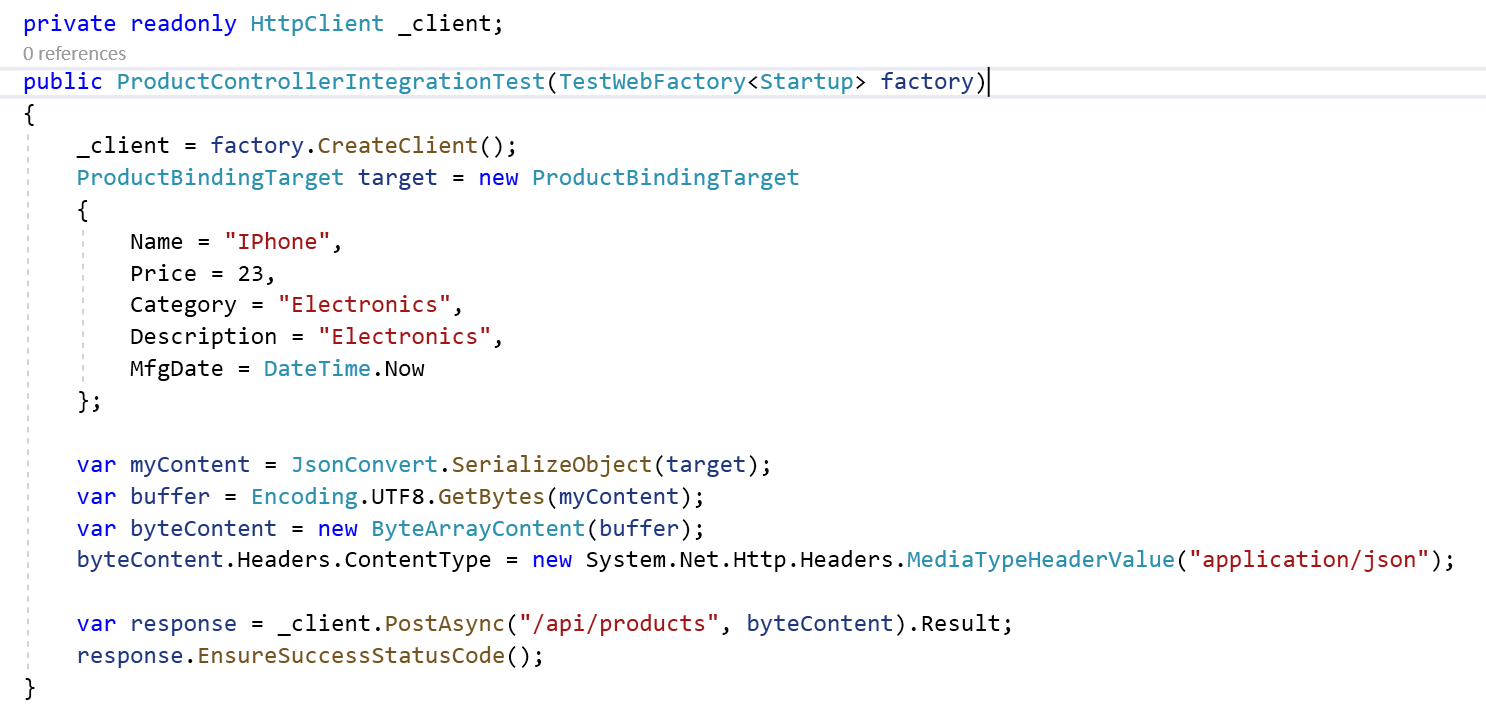
1. We need to create a dependencies of app in memory with respect to all the necessary infra required to Test the application . So Create Class that will create necessary infra for us.
   1. Add a class



* 1. Add Code to create infra required for testing



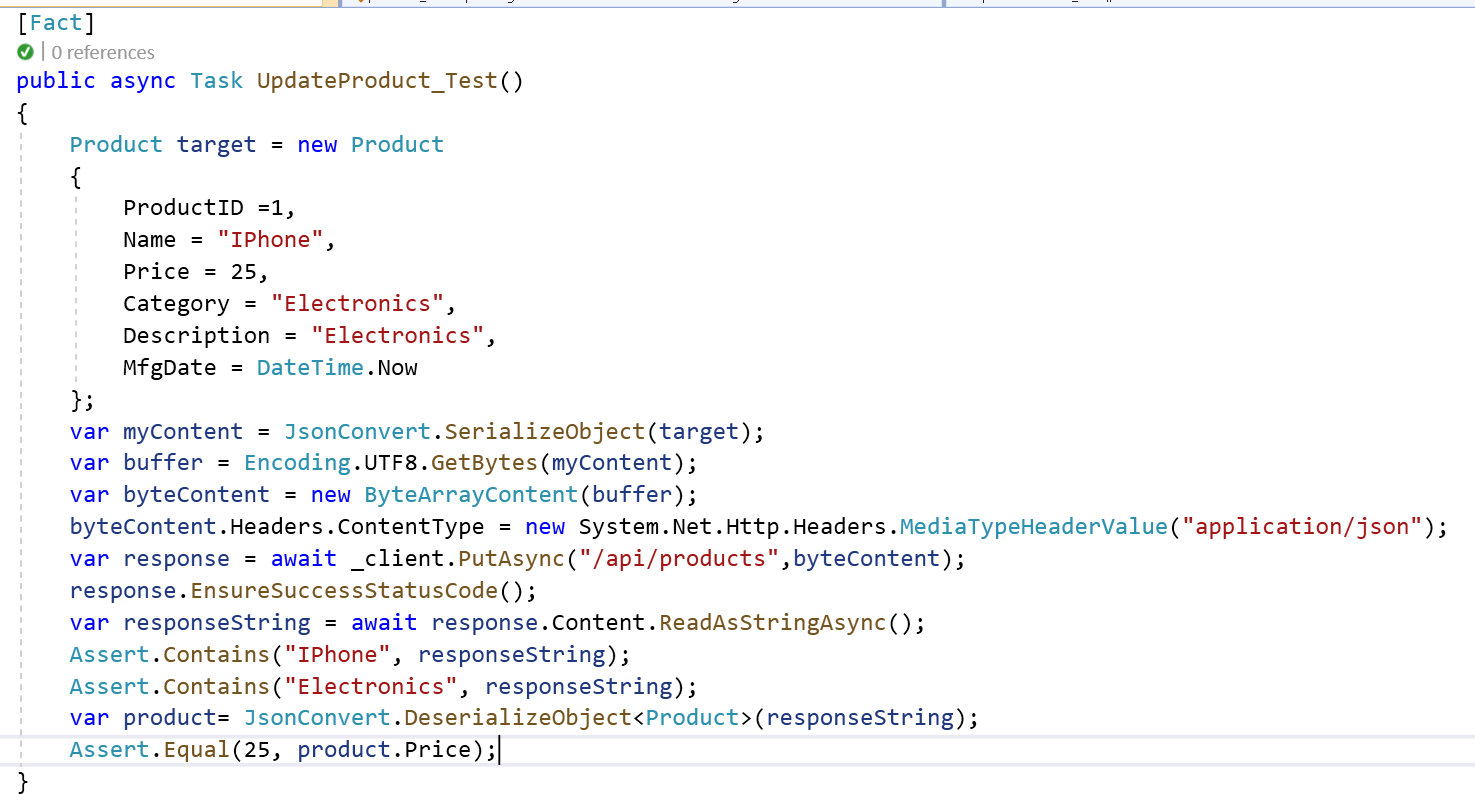
1. Add a code For Insertion



1. Write Test case for Get Products



1. Write Test case for UpdateProduct



1. Check the Output

