

Современные платформы прикладной разработки

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ (UNIT TESTING)

Модульное тестирование

<https://xunit.github.io/>

<https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test?view=aspnetcore-5>

Примеры:

<https://forproger.ru/article/testirovanie-kontrollerov-mvc-v-aspnet-core>

Общая информация

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

Модульное тестирование

Модульное тестирование представляет собой проверку приложения на самом низком уровне.

Каждый тест проверяет только один низкоуровневый компонент, например, один класс или даже один метод класса.

«Правильный» тест должен быть атомарным, повторяемым, изолированным и быстрым (**FAIR – fast, atomic, independent, repeatable**).

Модульное тестирование

Атомарность

Тест должен проверять только одну небольшую часть функционала – класс или метод класса – зачастую даже специфичные условия для метода

Повторяемость

Тест должен обеспечивать один и тот же результат в любое время, в любых условиях

Модульное тестирование

Изолированность/независимость

Тест должен быть полностью независим от других систем (например, баз данных) и тестов. Тест не должен предполагать, что другие тесты прошли успешно. Тест, в свою очередь, не должен влиять на другие тесты

Скорость

Тест должен выполняться быстро, т.к. ввиду атомарности для тестирования приложения может понадобиться множество тестов.

Модульное тестирование

Структура теста

Для построения модульных тестов часто используется шаблон AAA :

Arrange – Act - Assert

Модульное тестирование

Arrange – подготовка исходных данных

Act – вызов метода

Assert – проверка того, что результат соответствует ожиданиям

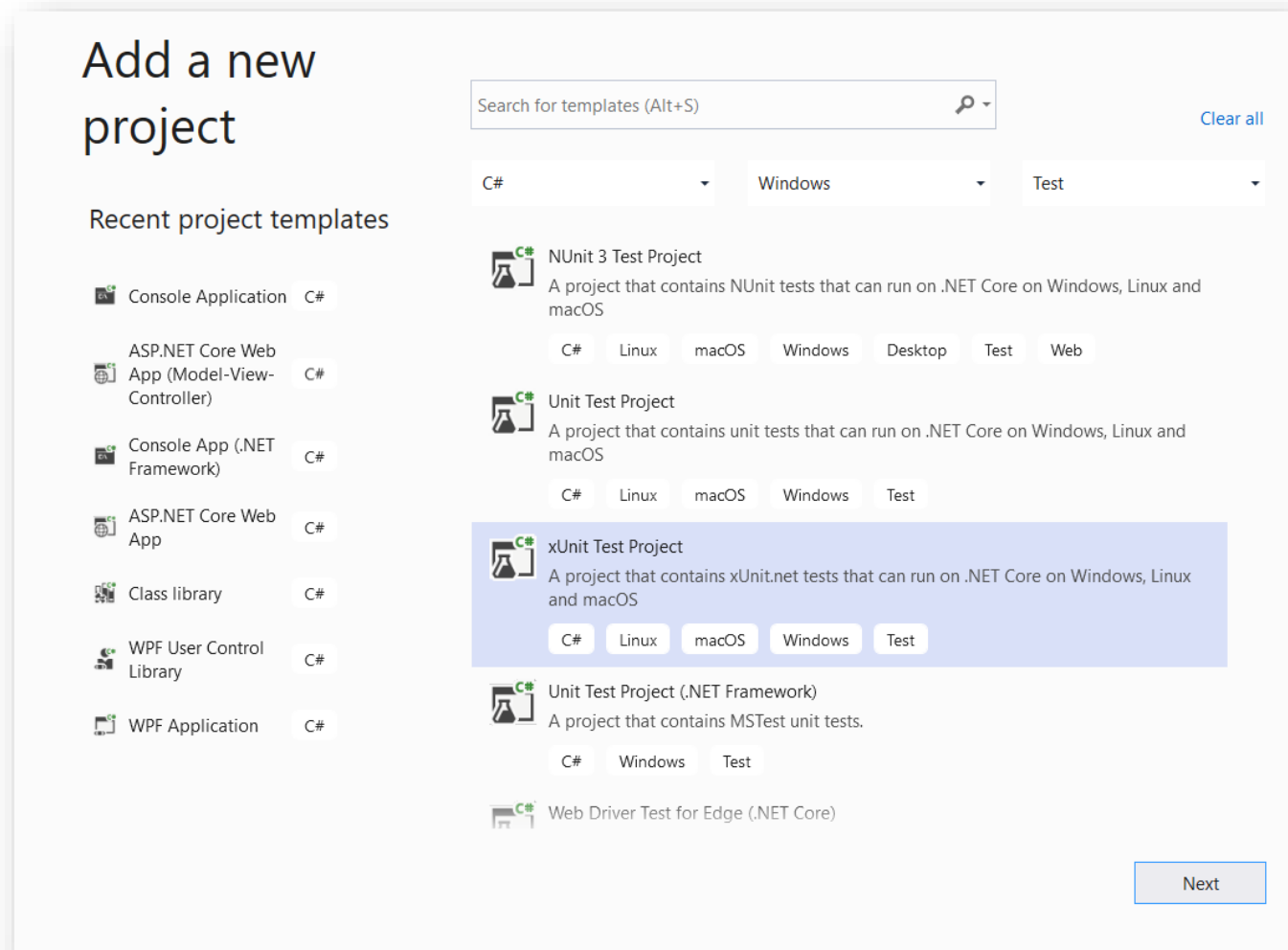
xUnit

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

xUnit

xUnit.net - бесплатный, opensource, ориентированный на сообщества инструмент модульного тестирования для .NET Framework.

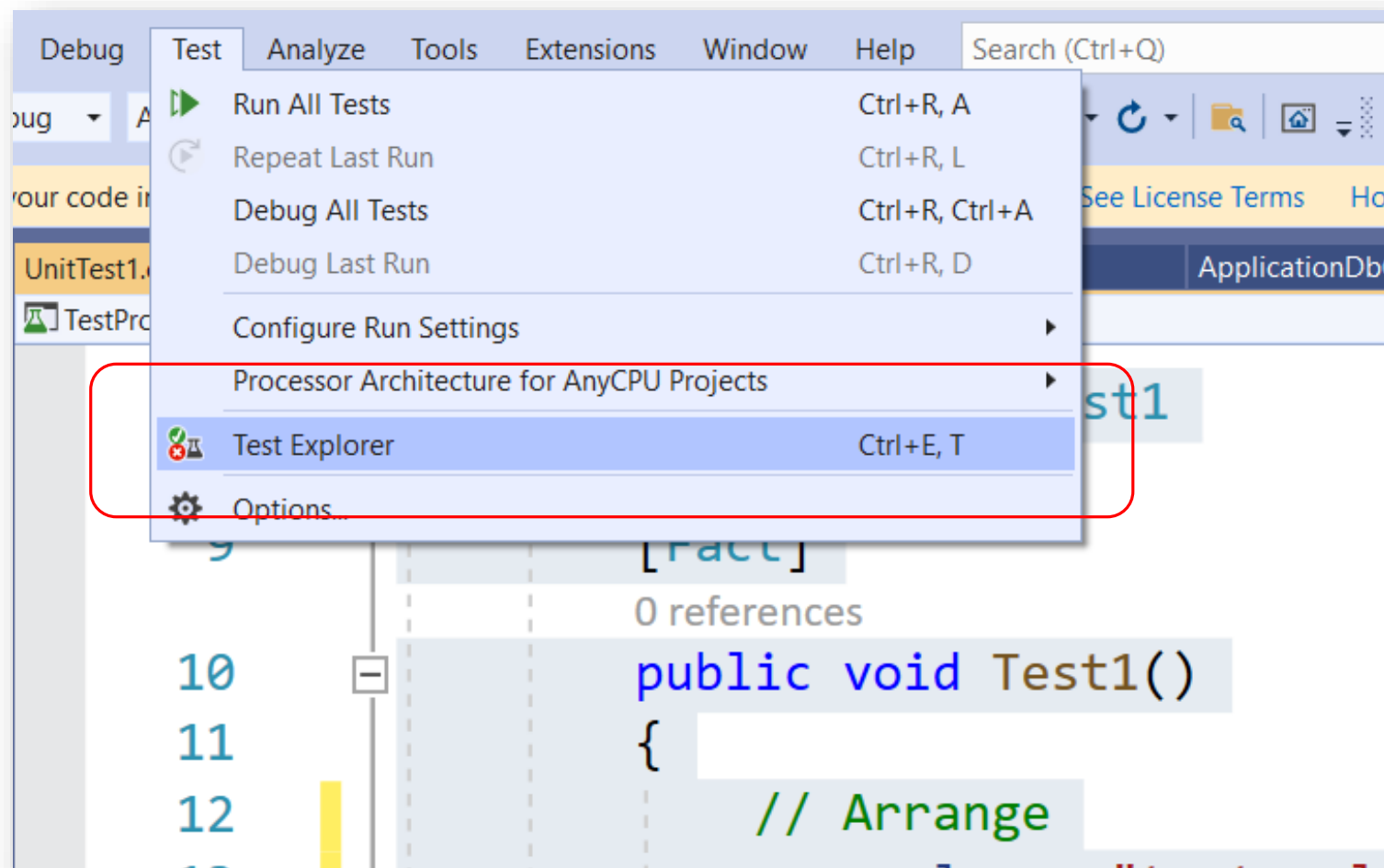
xUnit



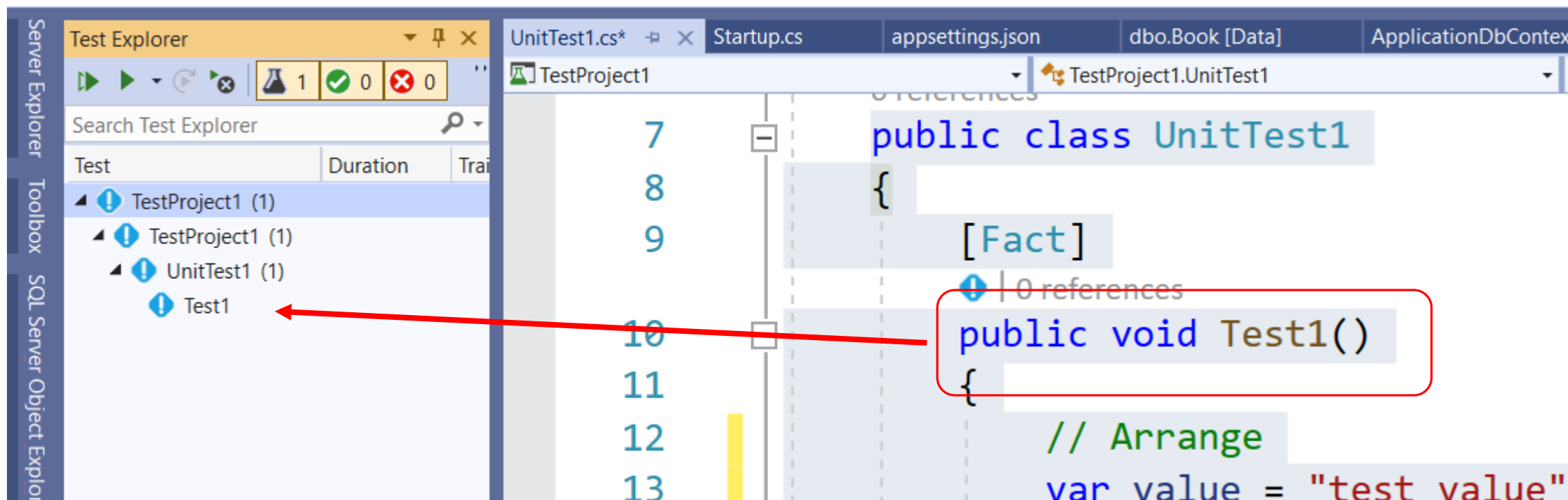
xUnit

```
public class UnitTest1
{
    [Fact]
    public void Test1()
    {
        // Arrange
        var value = "test value";
        var stack = new Stack<string>();
        stack.Push(value);
        // Act
        string result = stack.Pop();
        // Assert
        Assert.Equal(value, result);
    }
}
```

xUnit



xUnit



Класс Assert

XUNIT

Методы класса Assert

Contains – проверяет, что строка содержит указанную подстроку или коллекция содержит указанный элемент

DoesNotContain – проверяет, что строка НЕ содержит указанную подстроку или коллекция НЕ содержит указанный элемент

DoesNotThrow – проверяет, что код НЕ генерирует исключение

Методы класса Assert

Equal – проверяет, что два объекта эквивалентны.

False – проверяет условие на false

InRange – проверяет, что величина находится в указанных границах

IsAssignableFrom – проверяет, что объект принадлежит указанному типу или наследуется от него

Методы класса Assert

IsNotType – проверяет, что объект НЕ принадлежит указанному

IsType – проверяет, что объект принадлежит указанному

NotEmpty – Проверяет, что коллекция содержит элементы

NotEqual – проверяет, что два объекта НЕ эквивалентны

Методы класса Assert

NotInRange – проверяет, что величина находится ВНЕ указанных границ

NotNull – проверяет, что объект не Null

NotSame – проверяет, что два объекта НЕ представляют собой одну и ту же сущность

Методы класса Assert

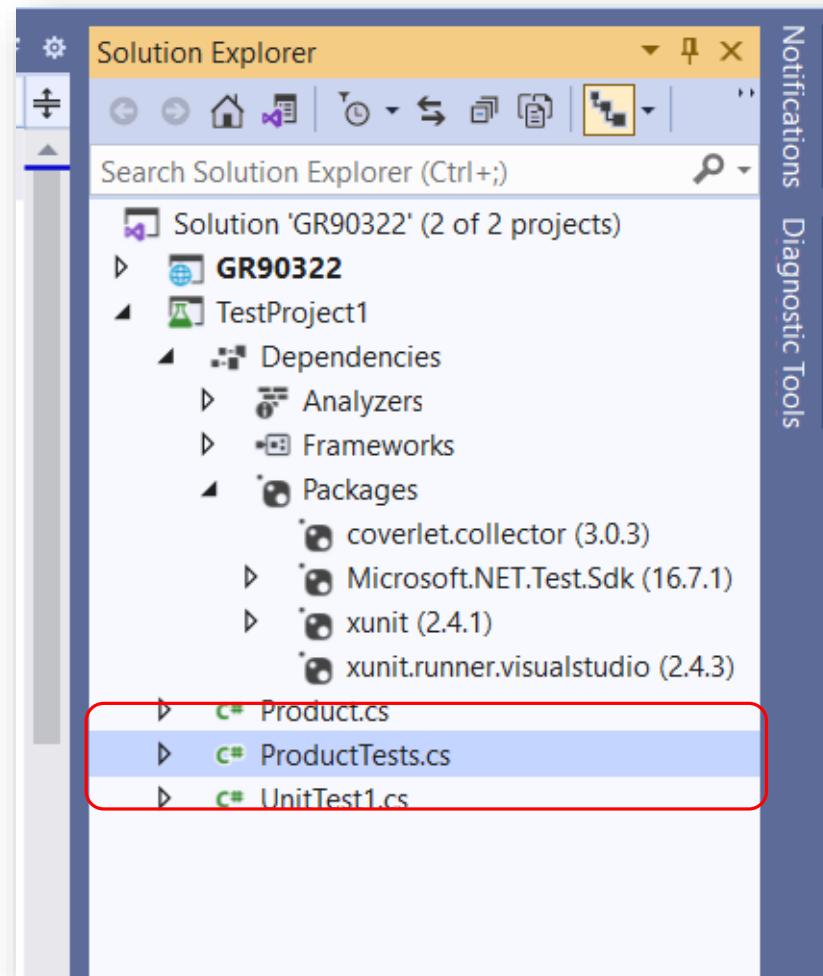
Null – проверяет, что объект равен Null

Same – проверяет, что два объекта представляют собой одну и ту же сущность

Throws – проверяет, что код генерирует исключение

True – проверяет условие на true

```
public class Product
{
    public int ID { set; get; }
    public string Name { set; get; }
    public decimal Price { set; get; }
    public void ApplyDiscount(decimal d)
    {
        if (Price <= d)
            throw new
                ArgumentOutOfRangeException();
        else Price -= d;
    }
}
```



xUnit

```
[Fact]
public void DiscountIsApplied()
{
    //Arrange
    Product prod = new Product() {Price = 100 };

    //Act
    prod.ApplyDiscount(10);

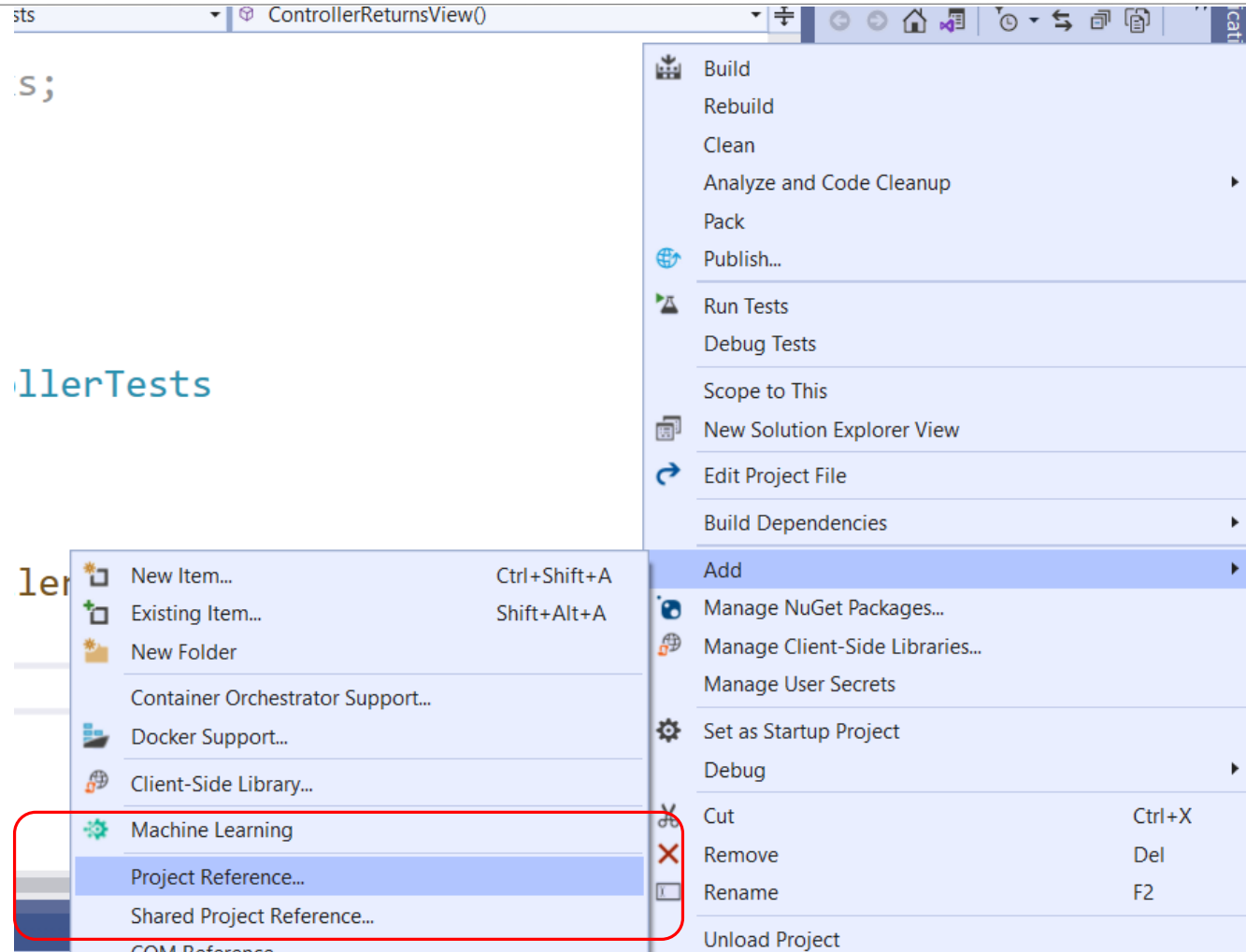
    //Assert
    Assert.Equal<decimal>(90, prod.Price);
}
```

xUnit

```
[Fact]
public void ExceptionGenerated()
{
    var prod = new Product() { Price = 10 };

    Assert.Throws<ArgumentOutOfRangeException>(
        ()=> prod.ApplyDiscount(10));
}
```


Тестирование контроллера



xUnit

[Fact]

```
public void IndexReturnsView()
{
    //Arrange
    var controller = new HomeController();
    //Act
    var result = controller.Index().Result;
    //Assert
    Assert.NotNull(result);
    var viewResult = Assert
        .IsType<ViewResult>(result);
    Assert.IsType<Book>(viewResult.Model);
}
```

```
string someData = viewResult  
                    .ViewData["SomeData"]  
                    .ToString();
```

Fact & Theory

XUNIT

Fact & Theory

xUnit поддерживает два разных типа модульных тестов, **Fact** и **Theory**.

Fact используется, когда у нас есть некоторые критерии, которые всегда должны выполняться независимо от данных.

Theory зависит от множества параметров и ее данных. Тест пройдет для определенного набора данных, а не других.

```
public class Calculator
{
    public double Div(int a, int b)
    {
        return (double)a / b;
    }
}
```

InlineData

[Theory]

[InlineData(2, 4, .5)]

[InlineData(2, 4, 1)]

[InlineData(3, 4, (double)3/4)]

```
public void CanDivide(int a, int b, double c)
{
    //arrange
    var calc = new Calculator();

    //assert
    Assert.Equal(c, calc.Div(a, b));
}
```


InlineData

Test Explorer SQL Server Object Explorer

ModuleTestDemo (5 tests) 1 failed

- ModuleTestDemo.Test (5) 49 ms
 - ModuleTestDemo.Test (5) 49 ms
 - CalcTests (3) 30 ms
 - CanDivide (3) 30 ms
 - CanDivide(a: 2, b: 4, c: 0.5) 19 ms
 - CanDivide(a: 2, b: 4, c: 1) 10 ms**
 - CanDivide(a: 3, b: 4, c: 0.75) 1 ms
 - StackTests (2) 19 ms

ModuleTestDemo.Test.CalcTests.CanDivide(a: 2, b: 4, c: 1) [Copy All](#)

Source: [CalcTests.cs](#) line 16

ModuleTestDemo.Test.CalcTests.CanDivide(a: 2, b: 4, c: 1)

Message: Assert.Equal() Failure
Expected: 1
Actual: 0.5

Elapsed time: 0:00:00.01

Stack Trace:
[CalcTests.CanDivide\(Int32 a, Int32 b, Double c\)](#)

InlineData

Достоинства: простота

Недостатки: нельзя передавать объекты классов

ClassData

ClassData – атрибут, позволяющий в качестве источника набора данных использовать класс.

ClassData

Класс, используемый в качестве источника данных, должен наследоваться от

`IEnumerable<object[]>`

ClassData

```
public class DataSource : IEnumerable<object[]>
{
    private List<object[]> datas = new List<object[]>
    {
        new object[] {2,4,.5},
        new object[] {2,4,1},
        new object[] {3,4,(double)3/4}
    };

    public IEnumerator<object[]> GetEnumerator()
        => datas.GetEnumerator();

    IEnumerator IEnumerable.GetEnumerator() =>
        GetEnumerator();
}
```

ClassData

```
public class CalcTests
{
    [Theory]
    [ClassData(typeof(DataSource))]
    public void CanDivide(int a, int b, double c)
    {
        //arrange
        var calc = new Calculator();

        //assert
        Assert.Equal(c, calc.Div(a, b));
    }
}
```

MemberData

Атрибут **MemberData** позволяет в качестве источника тестовых данных использовать метод.

Метод должен возвращать

`IEnumerable<object[]>`

MemberData

```
public class DataSource : IEnumerable<object[]>
{
    . . .

    public static IEnumerable<object[]> GetTestData()
    {
        yield return new object[] { 2, 4, .5 };
        yield return new object[] { 2, 4, 1 };
        yield return new object[] { 3, 4, (double)3 / 4 };
    }
}
```


MemberData

```
[Theory]
[MemberData(nameof(DataSource.GetTestData),
              MemberType = typeof(DataSource))]
public void CanDivide(int a, int b, double c)
{
    //arrange
    var calc = new Calculator();

    //assert
    Assert.Equal(c, calc.Div(a, b));
}
```

MemberData – передача объекта

```
public class Data
{
    public int a { get; set; }
    public int b { get; set; }
}
```

```
public bool AIsGreater(Data data)
    => data.a > data.b;
```

```
public static
    IEnumerable<object[]> GetTestData()
{
    yield return
        new object[] {new Data{a=4, b=3}};
    yield return
        new object[]{new Data{a=8, b=11}};
}
```

```
[Theory]
[MemberData(nameof(GetTestData))]
public void CompareAandB(Data a)
{
    Assert.True(AIsGerater(a));
}
```

Moq

XUNIT

Moq

<https://github.com/moq/moq/wiki/Quickstart#customizing-mock-behavior>

<https://github.com/moq/moq>

<https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/unit-testing-moq-framework>

Mock

Для проверки работы класса желательно проверять его независимо от других классов.

Если класс зависит от работы объектов других классов (в классе есть ссылки на другие классы, например, на класс работы с базой данных), то в этом случае удобно использовать макеты, фиктивные объекты (`mock` или `fake objects`), которые симулируют работу реальных объектов.

Moq

Moq – это простой и легковесный изоляционный фреймворк (Isolation Framework), который позволяет имитировать интерфейсы, виртуальные методы (и даже защищенные методы)

Подключение библиотеки Moq

В Package Manager Console введите команду:

`Install-Package Moq -version 4.1.1309.1617 -projectname xxx`

(номера версий могут отличаться)

Подключение библиотеки Moq

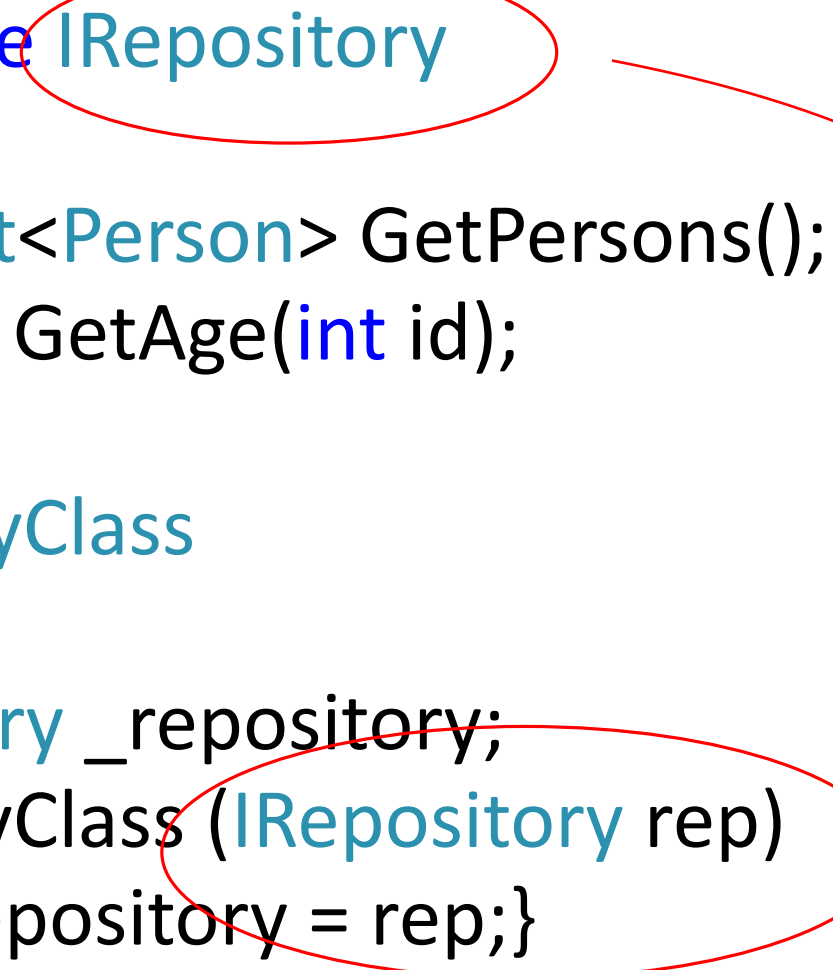
В меню Manager NuGet Packages for Solution

найдите и установите библиотеку Moq

Пример

```
public interface IRepository
{
    public List<Person> GetPersons();
    public int GetAge(int id);
}

public class MyClass
{
    IRepository _repository;
    public MyClass (IRepository rep)
    {
        _repository = rep;
    }
}
```



Создание объекта Mock

```
Mock<IRepository> moq = new Mock<IRepository>();
```

Моделирование метода GetPersons

```
mock.Setup(m => m.GetPersons())  
    .Returns(new List<Person> {  
        new Person { Name = "John", Age = 25 },  
        new Person { Name = "Mary", Age = 17 },  
        new Person { Name = "Ken", Age = 55 }  
    });
```

Моделирование метода GetAge

```
mock.Setup(m => m.GetAge(It.IsAny<int>()))  
    .Returns<int>(m => 30);
```

Методы класса `It`

`Is<T>(predicate)` - определяет значение типа `T`, для которого предикат возвращает `true`.

`IsAny<T>()` - любое значение типа `T`.

`IsInRange<T>(min, max, kind)` – параметр типа `T` должен находиться в пределах между `min` и `max`. Последний параметр принимает значение `Inclusive` или `Exclusive`.

`IsRegex(expr)` – текст должен удовлетворять регулярному выражению.

Использование Moq

Moq.Object()

Комплексный Moq object

```
Mock<IDiscount> mock = new Mock<IDiscount>();

mock.Setup(m => m.ApplyDiscount(It.IsAny<decimal>()))
    .Returns<decimal>(total => total);
mock.Setup(m => m.ApplyDiscount(It.Is<decimal>(v => v == 0)))
    .Throws<System.ArgumentOutOfRangeException>();
mock.Setup(m => m.ApplyDiscount(It.Is<decimal>(v => v > 100)))
    .Returns<decimal>(total => (total * 0.9M));
mock.Setup(m => m.ApplyDiscount(It.IsInRange<decimal>(10, 100,
    Range.Inclusive))).Returns<decimal>(total => total - 5);
```

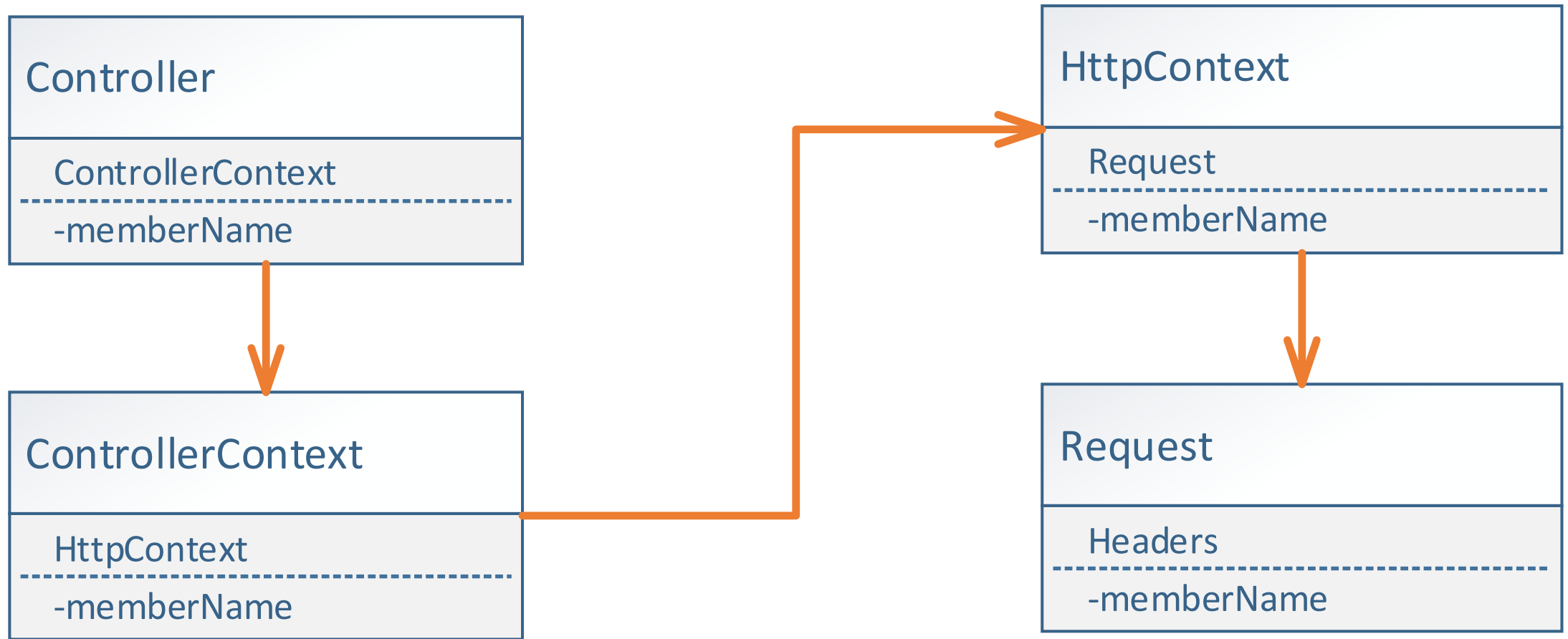
```
public IActionResult Index(int? group, int pageNo=1 )  
{
```

...

HttpContext.Request

```
    if (Request  
        .Headers["x-requested-with"]  
        .Equals("XMLHttpRequest");)  
        return PartialView("_listpartial", model);  
    else  
        return View(model);  
}
```

ControllerContext.HttpContext.Request.Headers



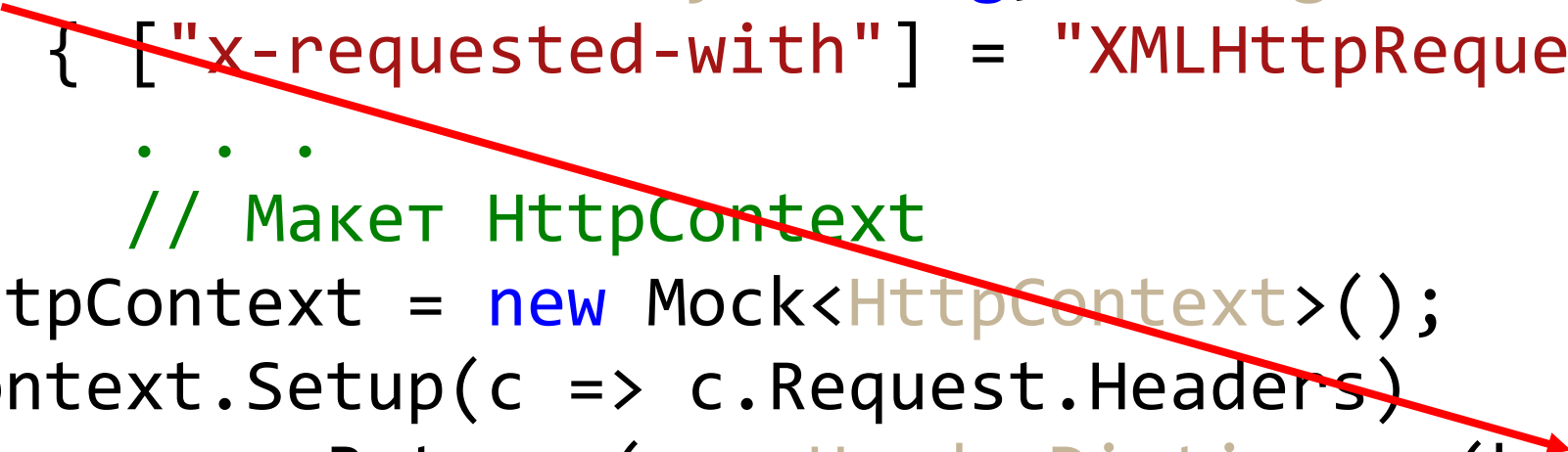
```
// Контекст контроллера
var controllerContext = new ControllerContext();
// Макет HttpContext
var mockHttpContext = new Mock<HttpContext>();

mockHttpContext.Setup(c => c.Request.Headers)
    .Returns(new HeaderDictionary());

controllerContext.HttpContext = mockHttpContext.Object;

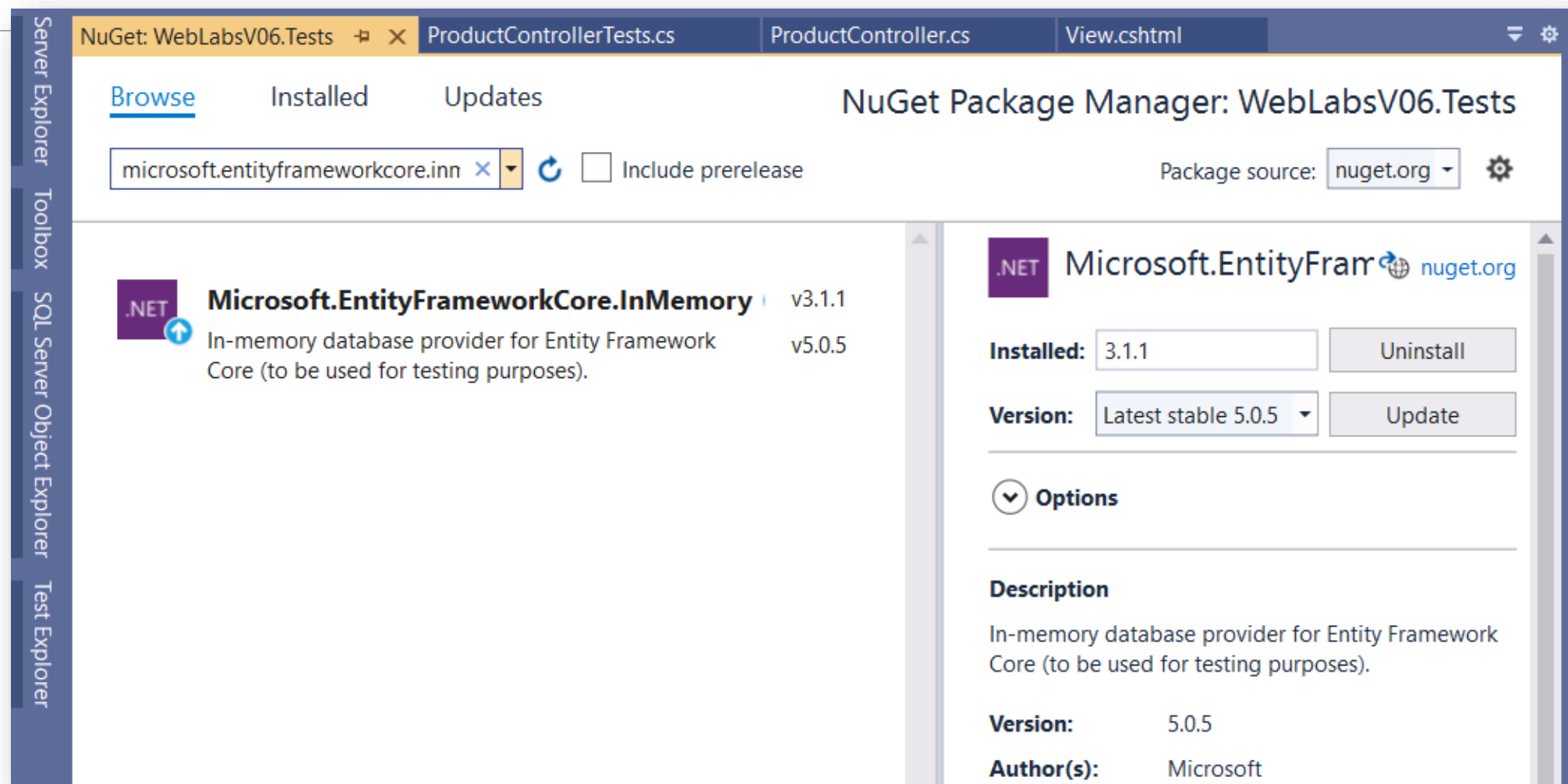
var controller = new ProductController()
    { ControllerContext = controllerContext };
```

```
var header = new Dictionary<string, StringValues>()  
    { ["x-requested-with"] = "XMLHttpRequest" };  
    . . .  
    // MakeT HttpContext  
var mockHttpContext = new Mock<HttpContext>();  
mockHttpContext.Setup(c => c.Request.Headers)  
    .Returns(new HeaderDictionary(header));
```



ВНЕДРЕНИЕ КОНТЕКСТА БД

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        :base(options)
    {
    }
}
```

```
_options = new DbContextOptionsBuilder<ApplicationDbContext>()  
            .UseInMemoryDatabase(databaseName: "testDb")  
            .Options;
```

```
using (var context = new ApplicationDbContext(_options))
{
    // создать объект класса контроллера
    var controller = new ProductController(context)
        { _context = context };
    . . .
}

// удалить базу данных из памяти
using (var context = new ApplicationDbContext(_options))
{
    context.Database.EnsureDeleted();
}
```