Code Academy

Dėstytojas

Vilmantas Neviera

pati įdomiausia dalis

API Unit testavimas

Data

Šiandien išmoksite

01 Moq

02 Autofixture

03 xUnit testavimas

04 SpecimenBuilder

NUnit vs xUnit

- Testus rašysime xUnit projekte, nes pagrindinis skirtumas tarp NUnit ir xUnit projektų yra tas jog xUnit kiekvienam testui duomenis sukuria iš naujo, o NUnit juos pernaudoja, ko pasekoje testai nėra taip gerai izoliuoti vienas nuo kito.

Moq ir Autofixture

Norint ištestuoti funkcionalumą, pradžioje reikia duomenų.

Paprastų duomenų mock'inimui yra labai geras framework'as pavadinimu Autofixture, jis suveda reikšmes į objektus kaip tai padarytų žmogus ir tam tereikia sukurti objektą testo parametruose.

```
public class UnitTest1
{
    [Theory, AutoData]
    public void Test1(Car car)
    {
        // Arrange
        var repositoryMock = new Mock<ICarRepository>();
        var mapperMock = new Mock<IMapper>();
        var loggerMock = new Mock<ILogger<CarController>>();
        var sut = new CarController(repositoryMock.Object, mapperMock.Object, loggerMock.Object);
        repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns(car);
        // Act
        var testResponse = sut.GetCarById(It.IsAny<Guid>());
        //Assert
        Assert.Equal(testResponse, car);
    }
}
```

Moq ir Autofixture

Toliau seka servisų mock'inimas su Moq framework'u.

Kadangi servisais operuojame per intraface'us su DI(dependency injection) mes galime susikurti testavimui skirtą **serviso implementacija**

Šiuo atveju testuojame Controllerį, todėl jį kurdami pavadiname jį **sut (subject under test)** ir per parametrus perduodame **.Object** reikšmę, kuri simbolizuoja testinę interface'o implementaciją

```
public class UnitTest1
{
    [Theory, AutoData]
    public void Test1(Car car)
    {
        // Arrange
        var repositoryMock = new Mock<ICarRepository>();
        var mapperMock = new Mock<IMapper>();
        var loggerMock = new Mock<ILogger<CarController>>();
        var sut = new CarController(repositoryMock.Object, mapperMock.Object, loggerMock.Object);
        repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns(car);
        // Act
        var testResponse = sut.GetCarById(It.IsAny<Guid>());
        //Assert
        Assert.Equal(testResponse, car);
    }
}
```

Moq ir Autofixture

Kita eilutė atsakingą už mūsų norimo testo scenarijaus sukūrimą.

Kadangi unit testai atsakingi už mažą kodo dalį, mes turime izoliuoti scenarijus, kuriuos būtent norime patestuoti.

Testuojamas controller’is turi du kelius - kai car objektas repozitorijoje rastas ir kai nerastas.

Mes bandome ištestuoti scenarijų, kai objektas rastas ir yra grąžintas.

Todėl mums reikia, kad repozitorija grąžintų objektą.

Per lambdą funkciją parodome, kad norimte setup’inti **GetCar** metodą ir taip pat nurodome,

kad testo metu gali būti paduotas bet koks parametras į **GetCar** metodą. Pabaigoje pasakome, kad tokiame repozitorijos kvietimo scenarijuje bus grąžintas mūsų testo metu su Autofixture sukurtas objektas

```
repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns(car);
```

Moq ir Autofixture

Pabaigoje Assert'inam ar tai ką grąžino controller'is yra tas pats, ką mes pasakėme, kad grąžintų mūsų scenarijuje

```
public class UnitTest1
{
    [Theory, AutoData]
    public void Test1(Car car)
    {
        // Arrange
        var repositoryMock = new Mock<ICarRepository>();
        var mapperMock = new Mock<IMapper>();
        var loggerMock = new Mock<ILogger<CarController>>();
        var sut = new CarController(repositoryMock.Object, mapperMock.Object, loggerMock.Object);
        repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns(car);
        // Act
        var testResponse = sut.GetCarById(It.IsAny<Guid>());
        //Assert
        Assert.Equal(testResponse, car);
    }
}
```

Moq ir Autofixture

Atkreipkite dėmesį į testų pavadinimus, jie gali būti ir labai ilgi, bet turi nusakyti testuojamą scenarijų.

```
[Theory, AutoData]
public void Repository_Returns_Car_Object_Correctly(Car car)
{
```

Taip pat, jeigu neperdavinėsite nieko parametruose, galite naudoti atributą

[Fact] vietoj **[Theory, AutoData]**

Moq ir Autofixture

Kito scenarijaus testavimas, kai repozitorija neranda objekto duomenų bazėje.

Atkreipkite dėmesį, kad nuorodau jog repozitorija grąžins null.

Kadangi controlleris gavęs null reikšme grąžina NotFound(), mums reikia patikrinti ar controller’io response.GetType() yra NotFoundObjectResult

```
[Fact]
public void Repository_Returns_Null()
{
    // Arrange
    var repositoryMock = new Mock<ICarRepository>();
    var mapperMock = new Mock<IMapper>();
    var loggerMock = new Mock<ILogger<CarController>>();
    var sut = new CarController(repositoryMock.Object, mapperMock.Object, loggerMock.Object);
    repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns((Car)null);
    // Act
    var testResponse = sut.GetCarById(It.IsAny<Guid>());
    //Assert
    var type = testResponse.Result.GetType();
    Assert.Equal("NotFoundObjectResult", type.Name);
}
```

Papildomas SpecimenBuilder

Kartais tai, kaip Autofixture sukuria objektus mum gali netikti, tokiu atveju reikia susikurti savo SpecimenBuilder, kuriame nurodysime kaip objektas turi atrodyt

SpecimenBuilder implementacija

```
public class CarSpecimenBuilder : ISpecimenBuilder
{
    public object Create(object request, ISpecimenContext context)
    {
        if (request is Type type && type == typeof(Car))
        {
            return new Car { Color = "White", Id = Guid.NewGuid(), Model = "Bmw" };
        }
        return new NoSpecimen();
    }
}
```

SpecimenBuilder implementacija

```
private readonly Fixture _fixture = new Fixture();

public CarControllerTests()
{
    _fixture.Customizations.Add(new CarSpecimenBuilder());
}

[Theory, AutoData]
public void Repository_Returns_Car_Object_Correctly()
{

    // Arrange
    var car = _fixture.Create<Car>();
    var repositoryMock = new Mock<ICarRepository>();
    var mapperMock = new Mock<IMapper>();
    var loggerMock = new Mock<ILogger<CarController>>();
    var sut = new CarController(repositoryMock.Object, mapperMock.Object, loggerMock.Object);
    repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns(car);
    // Act
    var testResponse = sut.GetCarById(It.IsAny<Guid>());
    //Assert
    Assert.Equal(testResponse.Value, car);
}
```

Kūrimas AutoDataAttribute

Norint dar labiau suoptimizuoti kodą, galime susikurti savo AutoDataAttribute.

Prieš tai matėme, kad Theory atributas naudojo taip pat AutoData atributą

```
[Theory, AutoData]
public void Repository_Returns_Car_Object_Correctly(Car car)
{
```

Bet AutoData atributas nenaudoja mūsų sukurto SpecimenBuilder’io.

Susikuriame savo atributą.

Kūrimas AutoDataAttribute

Taip atrodys klasė:

Paveldėdama iš AutoDataAttribute, ją galime naudoti kaip atributą su Theory.

Konstruktoriuje nurodome, kad būtų naudojamas base ir pridedame grąžinimą Fixture objekto, kurį prieš tai kūrėme testo klasėje.

```
public class CustomDataAttribute : AutoDataAttribute
{
    public CustomDataAttribute() : base(() =>
    {
        var fixture = new Fixture();
        fixture.Customizations.Add(new CarSpecimenBuilder());
        return fixture;
    })
    { }
}
```

Kūrimas AutoDataAttribute

Pakeičiame AutoData į CustomDataAttribute ir grąžiname Car objektą į parametrus.

Dabar šį objektą sukurs tokį, kokį nurodėme SpecimenBuilderyje ir pačiuose testuose Fixture kurti nebereikės.

```
public class CarControllerTests
{

    [Theory, CustomDataAttribute]
    public void Repository_Returns_Car_Object_Correctly(Car car)
    {
        // Arrange
        var repositoryMock = new Mock<ICarRepository>();
        var mapperMock = new Mock<IMapper>();
        var loggerMock = new Mock<ILogger<CarController>>();
        var sut = new CarController(repositoryMock.Object, mapperMock.Object, loggerMock.Object);
        repositoryMock.Setup(x => x.GetCar(It.IsAny<Guid>())).Returns(car);
        // Act
        var testResponse = sut.GetCarById(It.IsAny<Guid>());
        //Assert
        Assert.Equal(testResponse.Value, car);
    }
```

Fun fact!

Galime rašyti atributo pavadinimą be žodžio Attribute, šiuo atveju tai bus CustomData

```
[Theory, CustomData]
public void Repository_Returns_Car_Object_Correctly(Car car)
{
```

Užduotis nr. 1

- Ištestuokite 3 paskaitos programą.
- Susikurkite tiek savo SpecimenBuilder’į, tiek atributą.