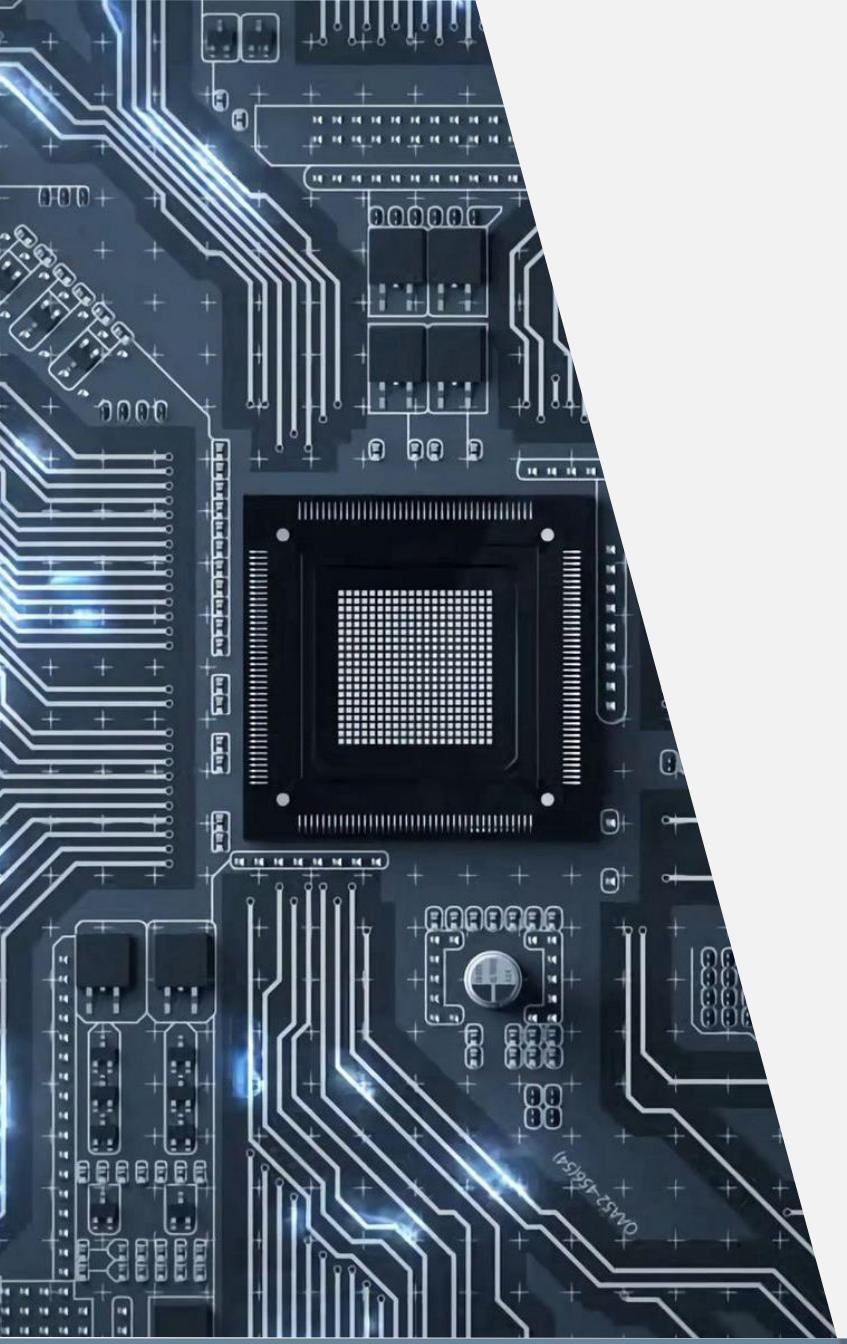
ShareValue

Mocking Around with .NET Unit Tests

DE STANDAARD VOORBIJ



Programma

- Wie ben ik?
- Given-When-Then
- FluentAssertions
- Moq
- AutoFixture
- EntityFrameworkCoreMock
- Demo
- Tips and tricks
- Vragen

Wie ben ik?

- Marco Kreeft
- Developer / Competence Lead @ ShareValue
- 36 jaar
- Samenwonend
- 2 katten
- Honk- en softbal
- F1
- Ajax

Given-When-Then

Bij unit testen is Given-When-Then een conventie voor het benoemen van tests op een gestructureerde en consistente manier. Het helpt bij het organiseren en beschrijven van de verschillende onderdelen van een test, waardoor de leesbaarheid en begrijpelijkheid van de test verbetert.

```
[Test]
0 references
public async Task GivenUserId_WhenGetByIdAsync_ThenUserShouldBeReturned()
{
    // Arrange
    var mockRepository = new Mock<IUserRepository>();
    var user = _fixture.Create<User>();
    mockRepository.Setup(expression: r :IUserRepository => r.GetByIdAsync(1)).ReturnsAsync(user);
    var service = new UserService(mockRepository.Object);

    // Act
    var result :User = await service.GetUserById(1);

    // Assert
    result.Should().BeEquivalentTo(user);
    mockRepository.Verify(expression: x :IUserRepository => x.GetByIdAsync(1), Times.Once);
}
```

FluentAssertions

Een zeer uitgebreide set van uitbreidingsmethoden die het mogelijk maken om op een meer natuurlijke manier de verwachte uitkomst van een TDD (Test Driven Development) of BDD (Behavior-driven development)-stijl unit testen te specificeren.

```
// Assert
result.Should().BeEquivalentTo(user);
result.Id.Should().Be(user.Id);
result.Email.Should().Be(user.Email);
result.Name.Should().Be(user.Name);:IUserRepository expression:
```

Moq

Moq is een open-source mocking framework voor .NET, waarmee het eenvoudiger wordt om mock objecten te maken en te gebruiken in unit tests. Het is ontworpen om het testen van code te vergemakkelijken door het mogelijk te maken om nep objecten te maken die de echte objecten vervangen waarvan de code afhankelijk is.

```
// Arrange
var mockRepository = new Mock<IUserRepository>();
var user = _fixture.Create<User>();
mockRepository.Setup(expression: r :IUserRepository => r.GetByIdAsync(1)).ReturnsAsync(user);
var service = new UserService(mockRepository.Object);
```

It.IsAny

It.IsAny is een methode die gebruikt om een flexibele mock op te zetten. Het stelt je in staat om een mock te maken van een methode die een bepaald type argument verwacht, zonder een specifieke waarde te hoeven opgeven voor dat argument. In plaats daarvan kun je It.IsAny<T> gebruiken om aan te geven dat elk argument van het juiste type geaccepteerd wordt.

```
mockRepository.Setup(r => r.UpdateAsync(It.IsAny<User>()))
    .Returns(Task.CompletedTask);
```

```
var mockRepository = new Mock<IUserRepository>();
mockRepository.Setup(r => r.GetByIdAsync(It.IsAny<int>()))
    .ReturnsAsync(new User { Id = 1, Name = "John Doe" });
```

```
mockSet.Setup(expression: m :DbSet<TEntity> => m.AddAsync(entity: It.IsAny<TEntity>(), cancellationToken: default)) // ISetup<DbSet<…>,ValueTask<…>>
    .Callback<TEntity, CancellationToken>((t :TEntity, _) =>
    {
        var temp :List<TEntity> = data.ToList();
        temp.Add(t);

        data = temp.AsQueryable();
    });
```

Verify

Verify wordt gebruikt om te verifiëren dat een bepaalde methode is aangeroepen met specifieke argumenten en/of een specifiek aantal keren.

```
// Assert
result.Should().BeEquivalentTo(user);
mockRepository.Verify(x => x.GetByIdAsync(1), Times.Once);:IUserRepository expression:
```

```
// Verify FindAsync has been called and not the synchonous Find
mockSet.Verify( expression: x :DbSet<User> => x.FindAsync(firstUser.Id), Times.Once);
mockSet.Verify( expression: x :DbSet<User> => x.Find(firstUser.Id), Times.Never);
```

```
_dbContextMock.Verify( expression: x :ApplicationContext => x.Users.Update(_user), Times.Once);
```

AutoFixture

AutoFixture is een open source bibliotheek voor het automatisch genereren van testdata. Met AutoFixture kan testdata gegenereerd worden die aan de testbehoeften voldoen zonder handmatig waarden te hoeven definiëren.

Create / CreateMany

Create is een van de belangrijkste functies van Autofixture, waarmee je een instantie van een willekeurig type kunt maken met automatisch gegenereerde waarden voor de eigenschappen van het object.

CreateMany kan gebruikt worden om meerdere instanties van een type te genereren.

```
var fixture = new Fixture();
var user = fixture.Create<User>();

var users = fixture.CreateMany<User>();
```

Build

Het verschil tussen Create en Build is dat Create automatisch willekeurige waarden toewijst aan alle eigenschappen van het object, terwijl Build meer controle geeft over de eigenschappen die worden ingesteld.

```
var fixture = new Fixture();
var user = fixture.Build<User>()
    .With(x => x.Name, "John")
    .With(x => x.Email, "test@test.nl")
    .Create();
```

```
var fixture = new Fixture();
var user = fixture.Build<User>()
    .With(x => x.Name, "John")
    .With(x => x.Email, "test@test.nl")
    .CreateMany();
```

Customize

Met customize kun je de manier waarop AutoFixture objecten genereert aanpassen. Je kunt bijvoorbeeld specifieke waarden toewijzen aan bepaalde eigenschappen van een object, of aangepaste algoritmes schrijven om een object te genereren. Hiermee kan je herhaling voorkomen.

```
1 reference
public class UserCustomization : ICustomization
{
    0 references
    public void Customize(IFixture fixture)
    {
        fixture.Customize<User>(composerTransformation: c :ICustomizationComposer<User> => c
            .Without(x :User => x.Id)
            .With(propertyPicker: x :User => x.Name, value: "Peter"));
    }
}
```

```
var allPeters :List<User> = _fixture.Customize(new UserCustomization()).CreateMany<User>().ToList();
```

EntityFrameworkCoreMock

EntityFrameworkCoreMock is een bibliotheek die wordt gebruikt om unit tests te schrijven voor applicaties die gebruik maken van Entity Framework Core als ORM framework. Hiermee kan je DbContextMock objecten maken die de werking van een echte DbContext simuleren, zodat unit tests uitgevoerd kunnen worden zonder daadwerkelijk toegang te hebben tot een database.

Demo

ShareValue

Tips and tricks

- Unit test, geen integratie test
- Given-When-Then
- Mocking frameworks
- Gebruik data-driven tests
- Wees niet bang om te refactoren om de code testbaarder te maken
- Test happy EN unhappy (crappy) flow
- DRY (Don't repeat yourself)

Vragen

ShareValue

ShareValue B.V.

Kampenringweg 45A
2803 PE Gouda

T +31 (0)182 516 468
E info@sharevalue.nl

Volg ons op

github.com/ShareValueNL/SV.Demo.MockingAround

github.com/marcokreeft87

marcokreeft

Tijd voor de borrel!

ShareValue

www.sharevalue.nl