Introduction

- Mocking framework for unit testing
- Strongly typed
- .NET
- Available se nuget package
Why mocking framework?

• Testing classes with interface dependencies
• Hand writing is repetitive, slow, painful, boring, agonizing, annoying, awful, ...
• Moq will do that for us
Inner workings

• Dynamic assembly on runtime
• Assembly name: “DynamicProxyGenAssembly2”
  • => [assembly: InternalsVisibleTo(“DynamicProxyGenAssembly2”)]
Example - Arrange

- Setup(lambda expression)
- Returns/Throws/Raises
Class It

- `Is<>(Predicate)`
- `IsAny<>()`

```csharp
mockStream
    .Setup(stream => stream.Write(It.IsAny<string>()))
    .Returns<string>(text => text.StartsWith("magic"));

mockStream
    .Setup(stream => stream.Write(It.IsAny<string>(text =>
        string.Equals(text, "magic", StringComparison.CurrentCultureIgnoreCase))));

mockStream
    .Setup(stream => stream.Write(It.IsNotNull<string>()))
    .Returns(true);
```
Linq toMocks

• “from the universe of mocks, get me one/those that behave like this”
• Allow for easily implemented recursive mocks

```csharp
ControllerContext context = Mock.Of<ControllerContext>(ctx =>
    ctx.HttpContext.Request.IsAuthenticated == true &&
    ctx.HttpContext.Response.ContentType == "application/xml");
```
Example - Act

- Object property
- That’s it

```csharp
Mock<ITextStream> mockStream = new Mock<ITextStream>();
ITextStream stream = mockStream.Object;
```
Example - Assert

- Method Verify(expression, Times)

```csharp
mockStream.VerifyAll();
mockStream.Verify(stream => stream.Read(), "Read was not called");
mockStream.Verify(stream => stream.Read(), Times.Once());

mockDB
    .Setup(db => db.Update(It.IsAny<string>()))
    .Returns(true);

// Act - should not modify DB
mockDB.Verify(db => db.Update(It.IsAny<string>()), Times.Never);
```
Question time