



**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Webes alkalmazások fejlesztése

9. előadás

Webszolgáltatások tesztelése (ASP.NET Core)

Cserép Máté

mcserep@inf.elte.hu

<http://mcserep.web.elte.hu>

Webszolgáltatások tesztelése

Tesztelés

- A webszolgáltatások tesztelése elvégezhető
 - manuálisan, kliens oldalon, a kérések küldését biztosító program, így böngésző vagy célszoftver (pl. *Postman*, *Insomnia*, *Fiddler*) segítségével
 - automatikusan, kliens oldalon, a kérések küldését biztosító osztály (pl. `HttpClient`) segítségével
 - automatikusan, szerver oldalon, a vezérlő műveleteinek közvetlen tesztelésével
 - a szolgáltatás tesztelését célszerű felügyelt környezetben, a teszten belül elvégezni
 - zárjuk ki a külső tényezőket (pl. adatbázis)

Webszolgáltatások tesztelése

Memóriabeli adatbázis

- Az adatbáziskontextus függőségi befecskendezés révén paraméterezhető a használt adatbázis motorral:
 - `services.AddDbContext<MyDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));`
- Használhatunk ideiglenes memóriában tárolt adatbázist is a teszteléshez:
 - `var options = new DbContextOptionsBuilder<MyDbContext>().UseInMemoryDatabase("TestDb").Options;
var context = new MyDbContext(options);`

Webszolgáltatások tesztelése

MSTest, xUnit, NUnit

- A platformfüggetlen egységtesztekhez használhatjuk többek között az *MSTest*, az *NUnit* vagy az *xUnit* keretrendszert is.
 - az *MSTest* valamint az *xUnit* natívan támogatott a Visual Studio 2017 által,
 - az *NUnit* a megfelelő NuGet csomag révén telepíthető.

MSTest	NUnit	xUnit	
[TestClass]	[TestFixture]	<i>n/a</i>	Teszt osztály.
[TestMethod]	[Test]	[Fact]	Teszteset (metódus).
[TestInitialize]	[Setup]	<i>ctor</i>	Tesztesetek inicializálása.
[TestCleanup]	[TearDown]	IDisposable	Tesztesetek takarítása.

Webszolgáltatások tesztelése

Példa

Feladat: Teszteljük az utazási ügynökség webszolgáltatását.

- egy külön *xUnit* tesztprojektben létrehozunk a tesztkörnyezetet biztosító osztályt (**TravelAgencyTest**), ezen belül pedig a vezérlők tesztéseit.
- az entitásmodellhez egy memóriabeli adatbázist használunk
 - az adatbázist minden teszteset előtt inicializáljuk egy minta adathalmazzal.
 - az adatbázist minden teszteset végrehajtása után semmisítjük meg.

Webszolgáltatások tesztelése

Mock objektumok

- Amennyiben függőséggel rendelkező programegységet tesztelünk, a függőséget helyettesítjük annak szimulációjával, amit *mock objektumnak* nevezünk
 - megvalósítja a függőség interfészét, egyszerű, hibamentes funkcionalitással
 - használatukkal a teszt valóban a megadott programegység funkcionalitását ellenőrzi, nem befolyásolja a függőségben felmerülő esetleges hiba
- Mock objektumokat manuálisan is létrehozhatunk, vagy használhatunk erre alkalmas programcsomagot
 - pl. *NSubstitute*, *Moq* letölthetőek NuGet segítségével

Webszolgáltatások tesztelése

Mock objektumok

- Pl. :

```
interface IDependency // függőség interfésze
{
    Boolean Check(Double value);
    Double Compute();
}
...
class DependencyImplementation : IDependency
    // a függőség egy megvalósítása
{
    public Boolean Check(Double value) { ... }
    public Double Compute() { ... }
}
```

Webszolgáltatások tesztelése

Mock objektumok

- Pl. :

```
class Dependant { // osztály függőséggel
    private IDependency _dependency;

    public Dependant(IDependency d) {
        _dependency = d;
    } // konstruktor befecskendezéssel helyezzük be
    // a függőséget
    ...
}
...
Dependant d =
    new Dependant(new DependencyImplementation());
// megadjuk a konkrét függőséget
```


Webszolgáltatások tesztelése

Mock objektumok

- Pl. :

```
class DependencyMock : IDependency
    // mock objektum
{
    // egy egyszerű viselkedést adunk meg
    public Double Compute() { return 1; }
    public Boolean Check(Double value) {
        return value >= 1 && value <= 10;
    }
}
...
Dependant d = new Dependant(new DependencyMock());
    // a mock objektumot fecskendezzük be a függő
    // osztálynak
```

Webszolgáltatások tesztelése

Mock objektumok

- *Moq* segítségével könnyen tudunk interfészekből mock objektumokat előállítani
 - a **Mock** generikus osztály segítségével példányosíthatjuk a szimulációt, amely az **Object** tulajdonsággal érhető el, és alapértelmezett viselkedést produkál, pl.:

```
Mock<IDependancy> mock =  
    new Mock<IDependancy>();  
    // a függőség mock objektuma  
Dependant d = new Dependant(mock.Object);  
    // azonnal felhasználható
```

- a **Setup** művelettel beállíthatjuk bármely tagjának viselkedését (**Returns (...)**, **Throws (...)**, **Callback (...)**), a paraméterek szabályozhatóak (**It**)

Webszolgáltatások tesztelése

Mock objektumok

- pl. :

```
mock.Setup(obj => obj.Compute()).Returns(1);  
    // megadjuk a viselkedést, mindig 1-t ad  
    // vissza
```

```
mock.Setup(obj =>  
        obj.Check(It.IsInRange<Double>(0, 10,  
            Range.Inclusive)))  
    .Returns(true);
```

```
mock.Setup(obj => obj.Check(It.IsAny<Double>()))  
    .Returns(false);  
    // több eset a paraméter függvényében
```

...

- lehetőségünk van a hívások nyomkövetésére (**Verify(...)**)

Webszolgáltatások tesztelése

Az Entity Framework mock-olása

- Hasonló módon az Entity Framework interfésze is mockolható, `DbSet` típusonként, például:

```
var cityData = new List<City> {  
    new City { Id = 1, Name = "TESTCITY" }  
};
```

```
IQueryable<City> qCityData =  
    cityData.AsQueryable();
```

```
cityMock = new Mock<DbSet<City>>();
```

Webszolgáltatások tesztelése

Az Entity Framework mock-olása

```
cityMock.As<IQueryable<City>>().Setup(mock =>  
    mock.ElementType).Returns(qCityData.ElementType);
```

```
cityMock.As<IQueryable<City>>().Setup(mock =>  
    mock.Expression).Returns(qCityData.Expression);
```

```
cityMock.As<IQueryable<City>>().Setup(mock =>  
    mock.Provider).Returns(qCityData.Provider);
```

```
cityMock.As<IQueryable<City>>().Setup(mock =>  
    mock.GetEnumerator()).Returns(  
    cityData.GetEnumerator());  
// a korábban megadott listát fogjuk visszaadni
```