



**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Webes alkalmazások fejlesztése

10. előadás

Webszolgáltatások tesztelése (ASP.NET Core)

Cserép Máté

mcserep@inf.elte.hu

<https://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
 - manuálisan, az API-hoz generált tesztelő asztali vagy webes felület segítségével (pl. *Swagger UI*)
 - 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.
 - Mind a három teszt keretrendszer natívan támogatott a Visual Studio 2019 által.

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.
 - a memóriabeli adatbázis kezeléséhez a projekthez adjuk a **Microsoft.EntityFrameworkCore.InMemory** NuGet csomagot

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 mockolása

- 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>>();
```

- Ahhoz, hogy a *mockolt* `DbSet` használható legyen, minimálisan az `ElementType`, `Expression` és `Provider` tulajdonságokat, valamint a `GetEnumerator()` metódust kell konfigurálnunk.

Webszolgáltatások tesztelése

Az Entity Framework mockolása

- Pl.:

```
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
```

Webszolgáltatások tesztelése

Az Entity Framework mockolása

- Hozzáadó és törlő műveletek kezelése:

```
_buildingMock.Setup(mock =>
    mock.Add(It.IsAny<Building>()))
    .Callback<Building>( building => {
        buildingData.Add(building); }
); // beállítjuk, hogy mi történjen hozzáadáskor
```

```
_buildingMock.Setup(mock =>
    mock.Remove(It.IsAny<Building>()))
    .Callback<Building>(building => {
        buildingData.Remove(building);
    });
); // beállítjuk, hogy mi történjen törléskor
```

Webszolgáltatások tesztelése

Példa

Feladat: Teszteljük az utazási ügynökség webszolgáltatását.

- a meglévő *xUnit* tesztprojektben hozzuk létre a *Moq* alapú egységtesztelését a webszolgáltatás vezérlőknek (**TravelAgencyMockTest**)
 - a projekthez adjuk hozzá a **Moq** NuGet csomagot
 - a városok és épületek **DbSet**-jét egy-egy memóriában tárolt listára mockoljuk
 - Az adatbázis kontextus (**TravelAgencyContext**) **Cities** és **Buildings** tulajdonságát ezen mockolt **DbSet**-ekre konfiguráljuk

Webszolgáltatások tesztelése

Integrációs tesztelés

- Több komponens együttes viselkedésének ellenőrzését integrációs tesztnek nevezzük. A külső tényezők, így pl. a hálózati kapcsolatból eredő hibák kiküszöbölésére a kliens és szerver együttes tesztelésekor is törekedni kell.
 - Mivel a szolgáltatás webszervert igényel, az *ASP.NET Core* biztosít egy könnyűsúlyú webszervert (**TestServer**), amely lehetővé teszi a szolgáltatás futtatását közvetlenül a memóriában, hálózati kapcsolat igénybevétele nélkül
 - A webszervert egy **HostBuilder** segítségével, egy a megszokott módon egy *startup* osztállyal (**TestStartup**) konfiguráljuk, amely eltérhet az éles konfigurációtól (pl. *in-memory* adatbázis).
 - A klienst (**HttpClient**) a szerverhez kapcsoltn példányosítjuk (**GetTestClient**), így minden kliensbeli kérés a memóriában hajtódik végre.

Webszolgáltatások tesztelése

Integrációs tesztelés

- Pl.:

```
var hostBuilder = new HostBuilder()
    .UseTestServer()
    .UseStartup<TestStartup>()
    .UseEnvironment("Development");
// betöltjük a szolgáltatás konfigurációját

IHost server = hostBuilder.Start();
// szerver példányosítása és indítása

HttpClient client = server.GetTestClient();
// kliens csatlakoztatása a szerverhez

var response = await client.GetAsync("api/...");
// a kérés a memóriában fut le
```

Webszolgáltatások tesztelése

Példa

Feladat: Teszteljük az utazási ügynökség webszolgáltatását.

- a meglévő *xUnit* tesztprojektben hozzuk létre egy integrációs teszt projektet a kliens perzisztencia rétegének tesztelésére (**TravelAgencyIntegrationTest**)
 - a **TestStartup** osztályban konfiguráljuk a szolgáltatást, az entitásmodellhez memóriabeli adatbázist használunk
 - a teszt szerver használatához a projekthez adjuk a **Microsoft.AspNetCore.TestHost** NuGet csomagot
 - a teszt szerver futtatásához a projekt *.csproj* állományában az SDK-t **Microsoft.NET.Sdk**-ről **Microsoft.NET.Sdk.Web**-re módosítsuk

```
<Project Sdk="Microsoft.NET.Sdk.Web">
```