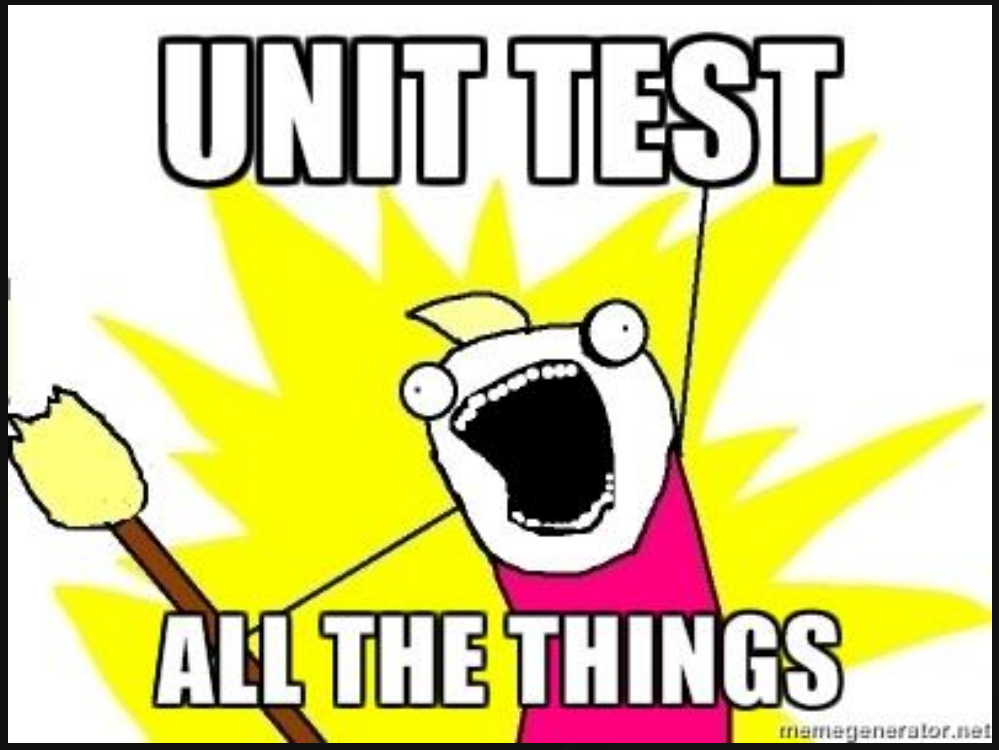
Unit tesztelhető ASP.NET Web API alkalmazás fejlesztése

Nagy Ákos



Szoftver verifikáció és validáció

Miről nem lesz szó?

- Mi az a Web API?

  Egyéb tárgyak az egyetemi képzés során

- Mi az a unit tesztelés?

  A tárgy (és egyéb egyetemi tárgyak) anyagában részletesen tárgyalva

Miről lesz szó?

- Mi az a Web API?
  … mert fontos az előadáshoz ☺
- Dependency injection
  … az eszköz, ami a unit tesztelhetőséget elősegíti
- Szoftver-architektúra, tervezési minták
  … a módszerek, amikkel be tudjuk vezetni a tesztelhetőséget

A cél...

...egy olyan ASP.NET Web API alkalmazás-architektúra bemutatása, ami elősegíti a funkcionalitás unit tesztelhetőségét.

ASP.NET Web API

Cél: REST architektúra implementálása .NET keretrendszerben

```
public class PhotosController:IController
{
  public Photo GetPhoto (int photoId) { ... }
}
```

www.superserver.com/Photos/GetPhoto?id=5

Dependency Injection

Probléma:

```
public class PersonController : IController
{
    PersonManager ps = new PersonManager();

    public void DoSomethingWithPerson()
    {
        ps.DoSomething( 42 );
    }
}
```

Dependency Injection

Probléma:

```
public class PersonController : IController
{
    IPersonManager ps = new PersonManager();

    public void DoSomethingWithPerson()
    {
        ps.DoSomething( 42 );
    }
}
```

Dependency Injection

```
public class PersonController : IController
{
  IPersonManager ps;

  public PersonController(IPersonManager ps)
  {
    this.ps=ps;
  }

  public void DoSomethingWithPerson()
  {
     ps.DoSomething( 42 );
  }
}
```

Dependency Injection

De akkor kinek a feladata a példányosítás?

DI – container

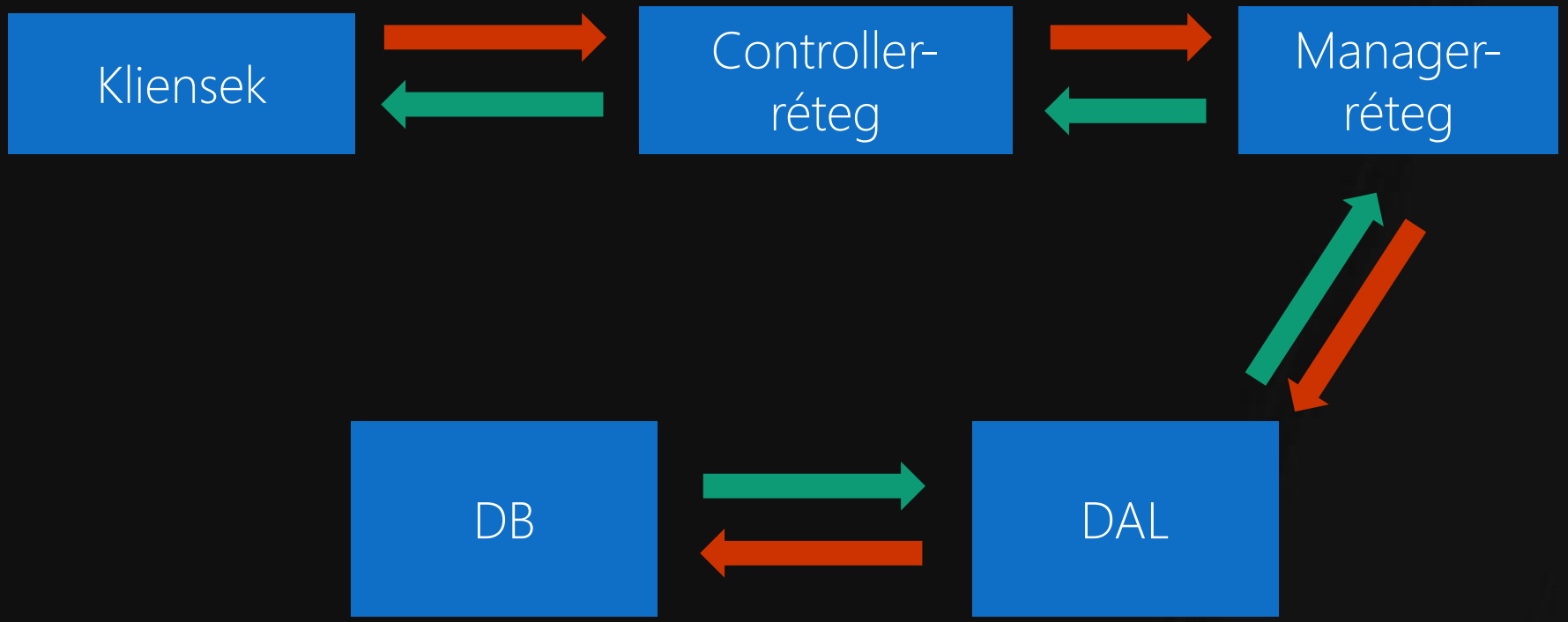
- Konfigurációban megadjuk, hogy ha adott típusú referencia kell, akkor milyen konkrét típust hozzon létre.
- Automatikusan, rekurzívan feloldja a konstruktorparamétereket és elvégzi az átadást
- Életciklusmenedzsment (?)

Dependency Injection

Miért jó?

- Konfigurálható példányosítás
- Éles kódban: modul példányosítása, ami adatbázisműveleteket végez
- Teszteléskor: olyan modult példányosítunk, ami nem db műveleteket végez, hanem csak egy mock/fake

Architektúra



Controller-réteg

Nem unit tesztelhető

Replikálni kellene az ASP.NET Web API pipeline-t is hozzá

De emiatt nem is feltétlenül kell unit tesztelni

Feladata a felhasználói kérések fogadása és a válaszok előállítása ➔ **Integrációs teszt hatásköre**

Persze akkor itt nem lehet üzleti logika

Manager-réteg

Ide kerül az üzleti logika

Üzleti szabályok betartása, üzleti folyamatok, tranzakciók

A DAL-ból veszi az adatokat

Unit tesztelhető, de szüksége van adatokra!!

Hogyan szerezzünk adatokat teszt közben? ➔ DAL feloldása DI-vel

DAL

Tipikusan a CRUD műveleteket valósítja meg valamilyen ORM technológiával (Entity Framework, NHibernate)

DE! Ezek tipikusan önálló keretrendszerek, nem mindig használhatóak jól DI megközelítésben

Repository tervezési minta

```
public class PhotoRepository : IPhotoRepository
{
  private EfContext ctx;

  public PhotoRepository(EfContext ctx)
  {
    this.ctx = ctx;
  }

  public Photo GetPhoto( int id ) { /* EF specifikus kód, ctx. */  }

  public int InsertPhoto(Photo newPhoto)
  { /* EF specifikus kód, ctx. */  }

  public void UpdatePhoto (int id, Photo newPhoto)
  { /* EF specifikus kód, ctx. */  }

  public void DeletePhoto (int id)
  { /* EF specifikus kód, ctx. */  }
}
```

Repository tervezési minta

Ez már DI-vel használható

Éles kódban PhotoRepository-t kötünk az IPhotoRepository interfészhez, tesztkörnyezetben TestPhotoRepositoryt

Éles kódban PhotoRepository-t kötünk az IPhotoRepository interfészhez, tesztkörnyezetben TestPhotoRepositoryt

DE! Hogyan tudunk több entitást is érintő tranzakciókat megvalósítani?

Unit Of Work tervezési minta

```
public class UnitOfWork : IUnitOfWork
{
  public IPhotoRepository PhotoRepository { get; private set; }
  public IUserRepository UserRepository { get; private set; }
  private EfContext ctx;

  public UnitOfWork(
      IPhotoRepository photoRep,
      IUserRepository userRep,
      EfContext ctx)

{
  this.PhotoRepository = photoRep;
  this.UserRepository = userRep;
  this.ctx = ctx;
}

  public void Commit()
  {
    ctx.SaveChanges();
  }
}
```

Unit Of Work tervezési minta

Feloldás:

IUnitOfWork ➔UnitOfWork, ehhez kell
IPhotoRepository ➔ PhotoRepository, ehhez kell
EfContext ➔EfContext
IUserRepository ➔ UserRepository, ehhez kell
EfContext ➔EfContext
EfContext ➔ EfContext

Az EfContext 3x oldódik fel, tehát ez csak akkor fog működni, ha minden feloldás ugyanazt az objektumot adja!

DI életciklus-menedzsment

Feloldáskor jöjjön-e létre objektum vagy sem?

Ninject:

- Singleton scope
- Thread scope
- Transient scope
- Custom scope
- Request scope

DI életciklus-menedzsment

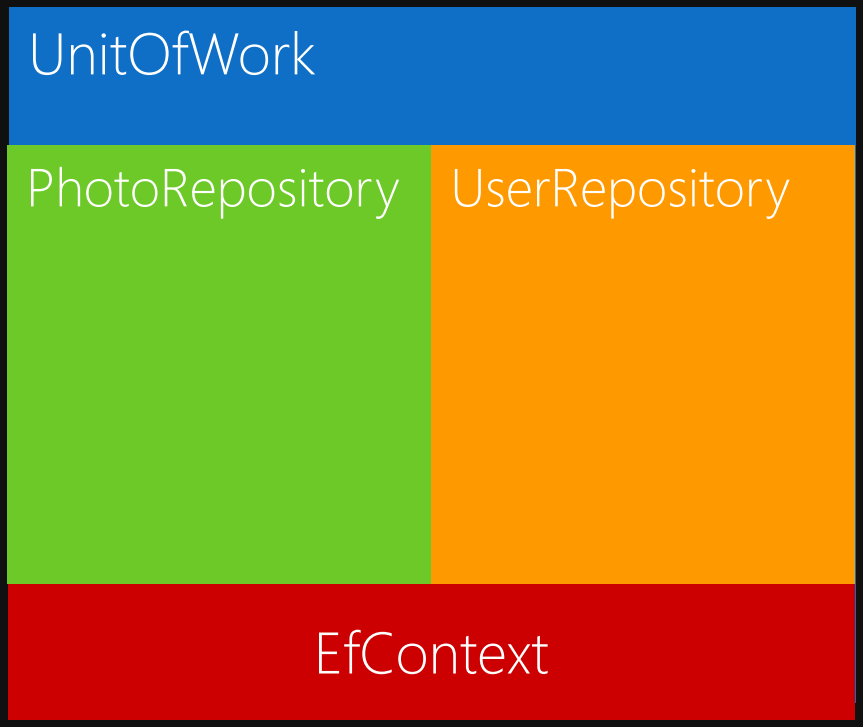
UnitOfWork, Repository-k, Managerek, Controllerek:
Nincs állapotuk ➔ Transient scope

EfContext:
Van állapota ➔ Request scope

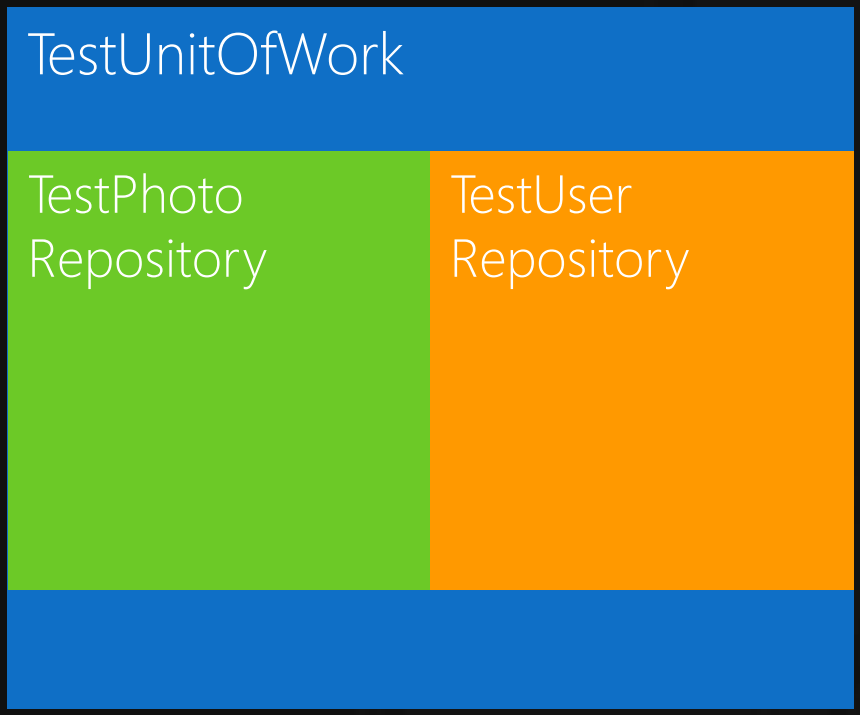
Unit tesztelés

Manager funkciók tesztje

Éles környezet



Unit teszt környezet



Miről volt szó?

- Mi az a Web API?
  … mert fontos az előadáshoz ☺
- Dependency injection
  … az eszköz, ami a unit tesztelhetőséget elősegíti
- Szoftver-architektúra, tervezési minták
  … a módszerek, amikkel be tudjuk vezetni a tesztelhetőséget

Köszönöm a figyelmet!