PV179 #3

Patrik Behrami, Lukáš Daubner

Obsah

- Repository pattern
  - Motivace
  - Příklad
  - Praktická ukázka
- Unit of work pattern
  - Motivace
  - Příklad
  - Praktická ukázka
- Samostatná práce 1 & 2
- Závěr a diskuze

Repository pattern - motivace

- Poskytuje vrstvu abstrakce mezi konzumentem dat a „tvrdou“ implementací přístupu k datům (=v našem případě EF)
- Repository jsou v našem případě třídy, které zapouzdřují funkčnost a logiku přístupu k datům
- Typická implementace koresponduje s vytvořenými DAL modely (=1 repository rozhraní a implementace pro každý model nebo…to si řekneme na konci ☺)
- **!Jako obvykle, tento přístup není tzv. silver bullet!**
- Po ukázce „správného“ řešení na konci cvičení si řekneme nevýhody

Repository pattern - příklad

```
public interface IAnimalRepository
{
    4 references
    Task<IEnumerable<Animal>> GetAnimalsAsync();
    1 reference
    Task<Animal> GetAnimalByIdAsync(Guid id);
    2 references
    Task InsertAnimalAsync(Animal animal);
    1 reference
    Task DeleteAnimalAsync(Guid animalId);
    2 references
    void UpdateAnimal(Animal animal);
    1 reference
    Task SaveAsync();
}
```

Repository – praktická ukázka

Unit of Work – motivace

- Je používán pro sjednocení logického celku několika operací prováděných nad datovým úložištěm tak, aby vždy všechny prošly nebo všechny selhaly (=simulace databázové transakce)
- V našem podání objekt, který udržuje veškeré změny na datech v průběhu jedné „transakce“ a následně zajišťuje, že se změny persistují nebo zahodí
- Často kombinován s Repository patternem
- V našem případě (při využití EF), implementace UoW například poskytuje našim repozitářům sdílet instanci DBContextu – i díky tomu dokáže zajistit atomičnost operací a korektní dispose spojení k databázi po ukončení „transakce“

Unit of Work - příklad

```
2 references
public interface IUnitOfWork : IAsyncDisposable
{
    7 references
    public IAnimalRepository AnimalRepository { get; }

    3 references
    public IZookeeperRepository ZookeeperRepository { get; }

    3 references
    public ISponsorRepository SponsorRepository { get; }

    1 reference
    public ISponsorshipRepository SponsorshipRepository { get; }

    2 references
    public Task CommitAsync();
}
```

Unit of Work - praktická ukázka

Samostatná práce – úvod

- Lab03_Tasks
  - Zoo.DAL - Vytvořený DAL s modely, migracemi, data seedem
  - Zoo.App – Konzolová aplikace konzumující Zoo.DAL, která orchestruje čtení a zapisování dat

Samostatná práce – zadání první části

1. Zajistěte, aby v konzolové aplikaci neexistovala závislost implementace na DbContextu
   - Repository pattern
2. Zajistěte, že všechny změny dat provedené během jedné operace se buďto provedou, nebo selžou
   - Unit Of Work pattern

Čas: ~25 min

Samostatná práce – ukázka řešení první části

Samostatná práce – zadání druhé části

- Pokud jste nestihli udělat plné řešení, použijte Lab03_Better

1. Zredukujte existující kód tak, aby zbytečně nebyl duplikován pro stejné operace nad různými objekty
   - Nápověda 1: Využijeme generik

```
1 reference
public async Task DeleteAnimalAsync(Guid animalId)
{
    var animalToDelete = await _context.Animal.SingleOrDefaultAsync(predicate: anim => anim.Id == animalId);
    _context.Animal.Remove(entity: animalToDelete);
}

1 reference
public async Task<Animal> GetAnimalByIdAsync(Guid id)
{
    return await _context.Animal.SingleOrDefaultAsync(predicate: anim => anim.Id == id);
}

4 references
public async Task<IEnumerable<Animal>> GetAnimalsAsync()
{
    return await _context.Animal.ToListAsync();
}
```

```
1 reference
public async Task DeleteSponsorAsync(Guid sponsorId)
{
    var sponsorToDelete = await _context.Sponsor.SingleOrDefaultAsync(predicate: anim => anim.Id == sponsorId);
    _context.Sponsor.Remove(entity: sponsorToDelete);
}

1 reference
public async Task<Sponsor> GetZookepeerByIdAsync(Guid id)
{
    return await _context.Sponsor.SingleOrDefaultAsync(predicate: anim => anim.Id == id);
}

1 reference
public async Task<IEnumerable<Sponsor>> GetSponsorsAsync()
{
    return await _context.Sponsor.ToListAsync();
}
```

Samostatná práce – zadání druhé části

1. Zredukujte existující kód tak, aby zbytečně nebyl duplikován pro stejné operace nad různými objekty
   - Nápověda 1: Využijeme generik
   - Nápověda 2: Generic repository pattern
   - Nápověda 3: Použití DbSet lze kombinovat s generiky

Čas: ~20 min

```
5 references
public class GenericRepository<TEntity> : IGenericRepository<TEntity> where TEntity : BaseEntity
{
    private ZooDbContext _dbContext;
    private DbSet<TEntity> _dbSet;

    4 references
    public GenericRepository(ZooDbContext dbContext)
    {
        _dbContext = dbContext;
        _dbSet = _dbContext.Set<TEntity>();
    }

    1 reference
    public async Task Delete(Guid idToDelete)
    {
        var entityToDelete = await _dbSet.FindAsync(keyValues: idToDelete);

        if (_dbContext.Entry(entity: entityToDelete).State == EntityState.Detached)
        {
            _dbContext.Attach(entity: entityToDelete);
        }

        _dbSet.Remove(entity: entityToDelete);
    }

    5 references
    public async Task<IEnumerable<TEntity>> Get()
    {
        return await _dbSet.ToListAsync();
    }
```

Samostatná práce – ukázka řešení druhé části

Repository a Unit of Work – závěr a diskuze

- Nad všemi design patterny je potřeba přemýšlet jako nad jednou z možných cest, nikoliv zázračnou magickou formulí, která vyřeší všechny problémy
- Naše repozitáře v tomto Labu jsou velmi zjednodušené – nepovolují např. přenesení filtrování a pagingu až na DB úroveň
  - Tzn. je nutné vždy vytáhnout všechna data do paměti, což je samozřejmě v běžné aplikaci nemyslitelné
- O užitečnosti Repository a speciálně Generic repository patternu v kombinaci s moderními ORM se vedou diskuze (odkaz 1, odkaz 2, odkaz 3)
- Možnost alternativy či doplňku k repozitářům – query objects