.NET Advanced

Repository en Unit Of Work pattern

AP HOGESCHOOL ANTWERPEN

AP.BE

Topics

- Repository pattern
- Unit Of Work pattern

.NET Advanced

Repository pattern

AP HOGESCHOOL ANTWERPEN

AP.BE

Repository pattern

Doel

- Het concept van het Repository pattern kennen kennen
- Het doel van het Repository pattern kennen
- Het Repository pattern kunnen implementeren

Repository pattern

Theorie

- Huidige opbouw exposed DbContext object aan BLL
  - Zorgt voor een afhankelijkheid tussen data access technologie en BLL
  - Wat als we van EF Core switchen naar bv. NHibernate?
- Scherm data access technologie af naar andere lagen toe
- Maak hiervoor gebruik van het Repository pattern

Repository pattern

Theorie

PeopleRepository

public IEnumerable<Person> GetAll()

```
public IEnumerable<Person> GetAll()
{
    return _pregaContext.People;
}
```

public Person GetById(int id)

public Person Create(Person person)

public void Update(Person person)

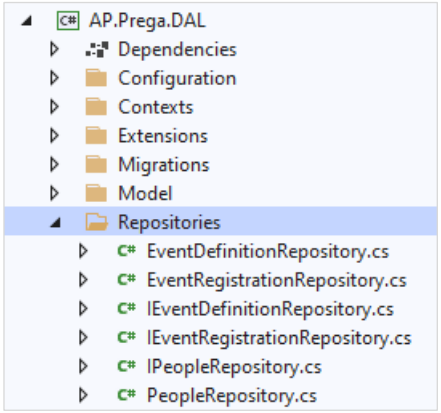
public void Delete(int id)

- Repository exposed enkel methoden naar buiten toe
- Enkel deze methoden zijn aanspreekbaar van buitenaf
- Implementatie van methoden maakt gebruik van data access technologie
- Data access technologie wordt op die manier afgeschermd en kan dus niet rechtstreeks aangesproken worden van buitenaf

Repository pattern

Theorie

- Plaatsing in Data Access Layer
  - Map Repositories met daarin interfaces en classes
  - Voor elke entiteit
    - Een interface
    - Een class die de interface implementeert



Repository pattern

Theorie

- Te voorzien in interface
  - Aantal standaard methoden bv. GetAll() - GetById() - Create() - Update() - Delete()
  - Kunnen aangevuld worden met specifieke methoden voor een entiteit
    - Bv. GetPeopleFromCity(string cityName)
- Te voorzien in class
  - Injecteren van DbContext via repository constructor
  - DbContext toewijzen aan private member variable
  - SaveChanges() van de DbContext aanroepen bij data manipulatie methoden Create() - Update() - Delete()

Repository pattern

Voorbeelden

Interface

```
public interface IPeopleRepository
{
    2 references
    public IEnumerable<Person> GetAll();
    4 references
    public Person GetById(int id);
    2 references
    public Person Create(Person person);
    2 references
    public void Update(Person modifiedPerson);
    2 references
    public void Delete(int id);
}
```

Implementatie

```
public class PeopleRepository : IPeopleRepository
{
    private PregaContext _pregaContext;

    1 reference
    public PeopleRepository(PregaContext pregaContext)
    {
        _pregaContext = pregaContext;
    }

    2 references
    public IEnumerable<Person> GetAll()
    {
        return _pregaContext.People;
    }

    4 references
    public Person GetById(int id)...

    2 references
    public Person Create(Person person)...

    2 references
    public void Update(Person modifiedPerson)...

    2 references
    public void Delete(int id)...
}
```

Injecteer de DbContext via de constructor

Repository pattern

Voorbeelden

Interface

```
public interface IPeopleRepository
{
    2 references
    public IEnumerable<Person> GetAll();
    4 references
    public Person GetById(int id);
    2 references
    public Person Create(Person person);
    2 references
    public void Update(Person modifiedPerson);
    2 references
    public void Delete(int id);
}
```

Implementatie Create - Update - Delete

```
public Person Create(Person person)
{
    _pregaContext.People.Add(person);
    _pregaContext.SaveChanges();

    return person;
}

2 references
public void Update(Person modifiedPerson)
{
    _pregaContext.People.Update(modifiedPerson);
    _pregaContext.SaveChanges();
}

2 references
public void Delete(int id)
{
    Person person = new() { Id = id };
    _pregaContext.Remove<Person>(person);
    _pregaContext.SaveChanges();
}
```

Roep SaveChanges() aan bij data manipulatie operaties

Repository pattern

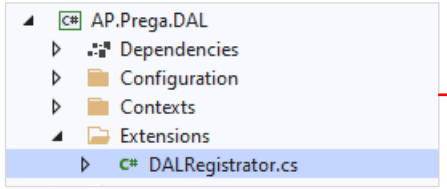
Theorie

- Repositories moeten ook geregistreerd worden bij de DI Container
- Uit te werken via extension method in Data Access Layer
- Algemene registratie methode voorzien die onderliggende private methods aanroept
- Algemene methode aanroepen vanuit ConfigureServices() method in Startup.cs van API
- Manier van werken analoog zoals bij registratie service classes

Repository pattern

Voorbeelden

Registreer de repositories in de DAL




```
public static IServiceCollection RegisterDataAccessServices(this IServiceCollection services)
{
    services.RegisterContexts();
    services.RegisterRepositories();

    return services;
}
```

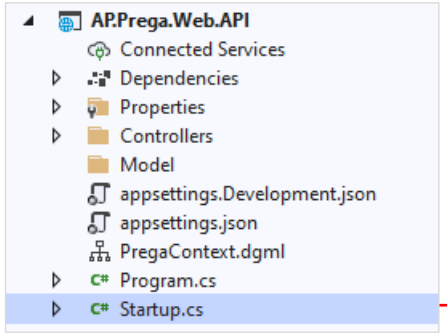
```
private static IServiceCollection RegisterRepositories(this IServiceCollection services)
{
    services.AddScoped<IEventDefinitionRepository, EventDefinitionRepository>();
    services.AddScoped<IEventRegistrationRepository, EventRegistrationRepository>();
    services.AddScoped<IPeopleRepository, PeopleRepository>();

    return services;
}
```

Repository pattern

Voorbeelden

Registreer de repositories




```
public void ConfigureServices(IServiceCollection services)
{

    services.AddControllers();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "AP.Prega.Web.API", Version = "v1" });
    });

    services.RegisterDataAccessServices();
    services.RegisterServices();
```

.NET Advanced

Unit Of Work pattern

AP HOGESCHOOL ANTWERPEN

AP.BE

Unit Of Work pattern

Doel

- Het concept van het Unit Of Work pattern kennen
- Het doel van het Unit Of Work pattern kennen
- Het Unit Of Work pattern kunnen implementeren
- Het Unit Of Work pattern kunnen combineren met het Repository pattern

Unit Of Work pattern

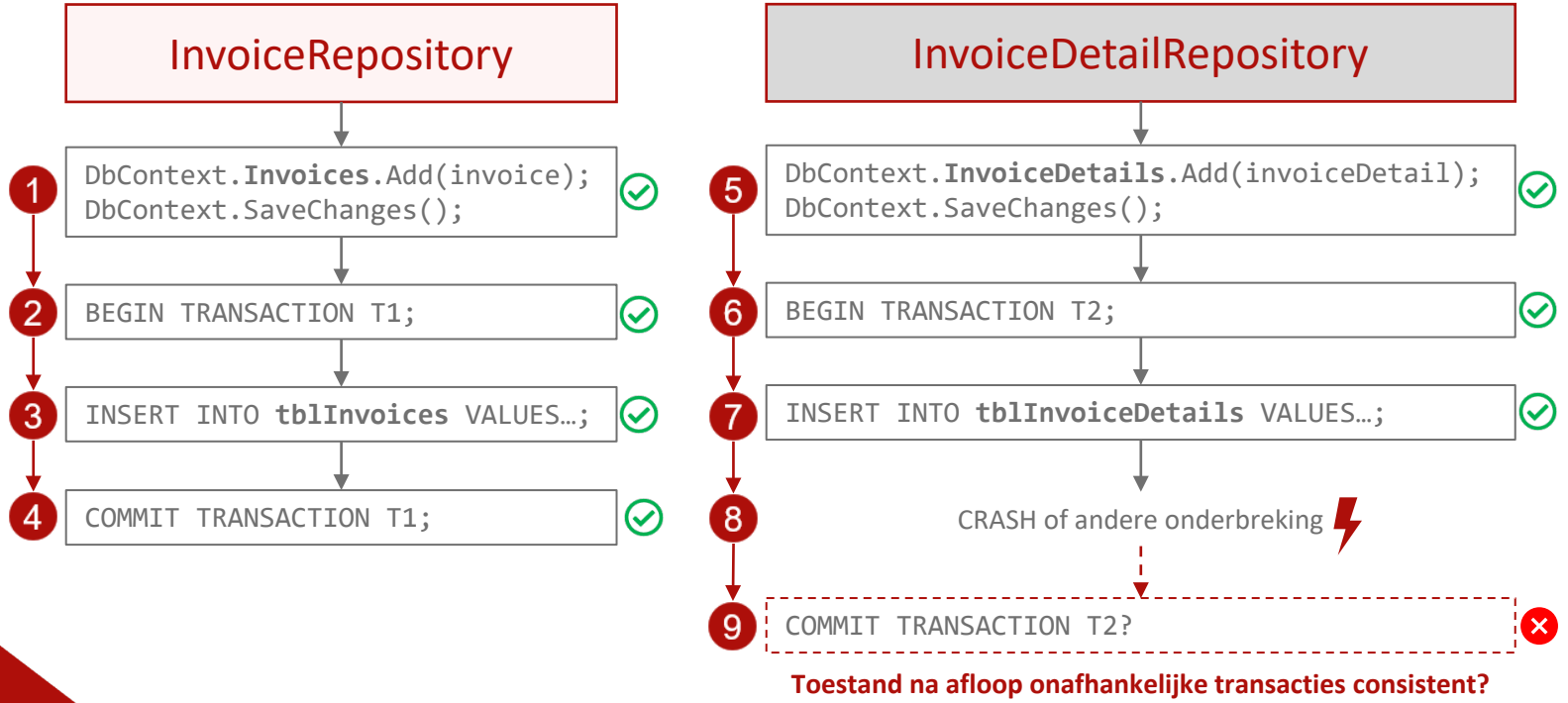
Theorie

- Potentiële problemen bij uitwerking repository pattern tot hiertoe

1. Bulk operaties bv. create van x-aantal records
   - Database roundtrip per create
2. Transactie die als 1 geheel moet uitgevoerd worden
   - Elke aanroep van SaveChanges() wordt in een transactie uitgevoerd.
   - SaveChanges per aparte entiteit/repository aanroepen resulteert in onafhankelijk van elkaar uitgevoerde transacties en niet in één allesomvattende transactie

Unit Of Work pattern

Theorie



Toestand na afloop onafhankelijke transacties consistent?

Unit Of Work pattern

Theorie

- Oplossing
  - Zorgen dat alle data manipulaties in een keer worden uitgevoerd ongeacht het aantal entiteiten/repositories dat onderdeel uitmaakt van deze bewerking
  - Dus 1x SaveChanges() aanroepen

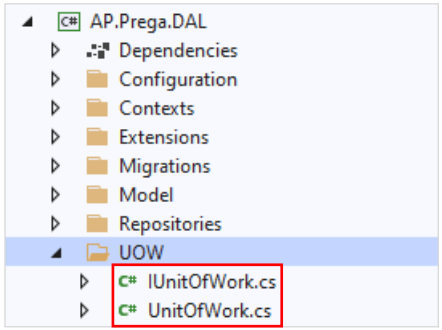
⇨ Unit Of Work pattern zorgt hier voor

- In kleine applicaties 1 UnitOfWork class uitwerken
- In grotere applicaties meerdere UnitOfWork classes in functie van bij elkaar horende entiteiten/repositories

Unit Of Work pattern

Theorie

- Plaatsing in Data Access Layer
  - Interface en class
  - Eventueel in aparte map geplaatst



Unit Of Work pattern

Theorie

- Te voorzien in Interface
  - Commit() method (soms ook Complete() of SaveChanges() als method name)
  - Property per repository met enkel “Getter”
- Te voorzien in Class
  - Injecteren DbContext en Repositories via constructor
  - Toewijzen aan private member variable
  - Private member variables per repository
  - “Getters” per repository die respectievelijke private member variables teruggeven
  - Commit() method die de SaveChanges() method van de DbContext uitvoert

Unit Of Work pattern

Voorbeelden

Interface

```
public interface IUnitOfWork
{
    10 references
    public int Commit();

    8 references
    public IEventDefinitionRepository EventDefinitionRepository { get; }
    8 references
    public IEventRegistrationRepository EventRegistrationRepository { get; }
    8 references
    public IPeopleRepository PeopleRepository { get; }
}
```

- Definieer een property voor elke repository
- Gebruik Interface als type voor elke repository
- Repositories mogen niet van buitenaf ingesteld worden, dus enkel getter voorzien

Unit Of Work pattern

Voorbeelden

Implementatie

```
public class UnitOfWork : IUnitOfWork
{
    private PregaContext _pregaContext;
    private IEventDefinitionRepository _eventDefinitionRepository;
    private IEventRegistrationRepository _eventRegistrationRepository;
    private IPeopleRepository _peopleRepository;

    0 references
    public UnitOfWork(
        PregaContext pregaContext,
        IEventDefinitionRepository eventDefinitionRepository,
        IEventRegistrationRepository eventRegistrationRepository,
        IPeopleRepository peopleRepository)
    {
        _pregaContext = pregaContext;
        _eventDefinitionRepository = eventDefinitionRepository;
        _eventRegistrationRepository = eventRegistrationRepository;
        _peopleRepository = peopleRepository;
    }
```

Injecteer de DbContext en Repositories via de constructor

Unit Of Work pattern

Voorbeelden

Implementatie

```
public IEventDefinitionRepository EventDefinitionRepository
{
    get
    {
        return _eventDefinitionRepository;
    }
}
public IEventRegistrationRepository EventRegistrationRepository ...
8 references
public IPeopleRepository PeopleRepository ...
```

Property bevat enkel een “Getter”

Property per repository

```
public int Commit()
{
    return _pregaContext.SaveChanges();
}
```

SaveChanges() wordt niet langer vanuit de methods in elke individuele repository uitgevoerd maar in de Commit() method van de UnitOfWork class

Unit Of Work pattern

Theorie

- Wat verandert er?
- Repositories
  - SaveChanges() method call verdwijnt uit alle methods
- Service classes
  - UnitOfWork interface injecteren i.p.v. DbContext
  - Alle methodes in service classes gebruiken repositories via UnitOfWork instance
  - Commit() methode vanuit de gepaste service class method aanroepen
- UnitOfWork
  - Registreren van de UnitOfWork class bij de DI container (scoped)

Unit Of Work pattern

Voorbeelden

Registreer de Unit(s) Of Work in de DAL

```
public static IServiceCollection RegisterDataAccessServices(this IServiceCollection services)
{
    services.RegisterContexts();
    services.RegisterRepositories();
    services.RegisterUnitsOfWork();

    return services;
}
```

```
private static IServiceCollection RegisterUnitsOfWork(this IServiceCollection services)
{
    services.AddScoped<IUnitOfWork, UnitOfWork>();

    return services;
}
```

Unit Of Work pattern

Voorbeelden

Implementatie service class

```
public EventDefinitionService(IUnitOfWork unitOfWork)
{
    _unitOfWork = unitOfWork;
}
```

Injecteer de UnitOfWork in de service class via de constructor

```
public IEnumerable<EventDefinition> GetAll()
{
    return _unitOfWork.EventDefinitionRepository.GetAll();
}
```

Get operaties in de service class gebeuren via de repositories van de UnitOfWork instance

Unit Of Work pattern

Voorbeelden

Implementatie

```
public EventDefinition Create(EventDefinition eventDefinition)
{
    if (eventDefinition.StartsOn < DateTime.Now)...

    if (eventDefinition.EndsOn < DateTime.Now)...

    if (eventDefinition.EndsOn < eventDefinition.StartsOn)...

    if (eventDefinition.Venue.Length > 80)...

    eventDefinition = _unitOfWork.EventDefinitionRepository.Create(eventDefinition);

    foreach(EventRegistration eventRegistration in eventDefinition.EventRegistrations)
    {
        _unitOfWork.EventRegistrationRepository.Create(eventRegistration);
    }

    _unitOfWork.Commit();

    return eventDefinition;
}
```

- We willen een event en bijhorende pre-registraties bewaren
- Via de 2 repositories worden de nodige records toegevoegd in de onderliggende shared DbContext
- Via de Commit() method wordt onderliggend de SaveChanges() method uitgevoerd van de shared DbContext

Oefeningen

Deel 8 - Oefenbundel 1

- Surf naar Digitap
- Download de opgave
- Los de vragen op

Repository en Unit Of Work pattern

Bronnen

- Microsoft
  - https://docs.microsoft.com/en-us/ef/core/saving/transactions
- Programming with Mosh
  - https://www.youtube.com/watch?v=rtXpYpZdOzM