

.NET Advanced

N-Layered architectuur

Topics

- Concept
- Data Access Layer
- Business Logic/Service Layer
 - Custom exceptions voor datavalidatie
- API Layer
- Refactoring services registratie

N-Layered architectuur

Concept

Concept

Doel

- Het concept van de N-Layered architectuur kennen

Concept

Theorie

- Alle logica van een applicatie in 1 project is meestal geen goede keuze
- Applicatie opsplitsen in aparte lagen
 - Elke laag noemen we een **Layer**
 - Elke laag is verantwoordelijk voor specifieke aspecten van de functionaliteit
 - Aantal lagen alsook terminologie kan variëren

Concept

Theorie

- Een typische N-Layered architectuur bestaat uit volgende layers
 - **Data Access Layer**
 - Alle objecten en logica die te maken hebben met toegang tot de database
 - Alle objecten en logica omtrent EF Core komen in deze laag
 - **Business Logic/Service Layer**
 - Alle objecten en logica die te maken hebben met business rules (bv. korting bepalen voor een factuur) komen hierin
 - **API Layer**
 - Alle objecten en logica nodig om informatie over HTTP naar de presentatie laag te sturen
 - **Presentation Layer**
 - Alle objecten en logica om de gebruikersinterface uit te werken

Concept

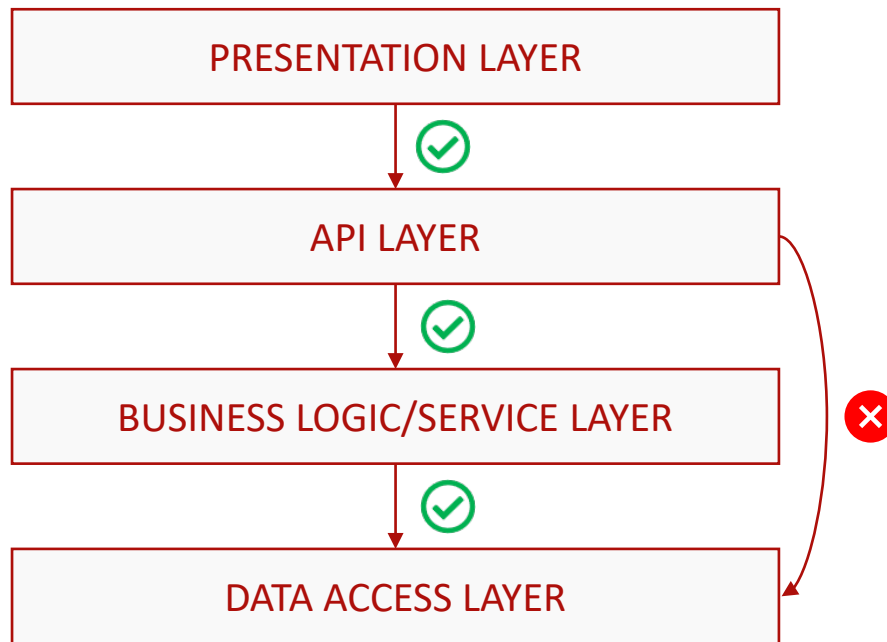
Theorie

- Waarom?
 - ⇒ Separation of concerns, bv:
 - Data Access Layer classes implementeren enkel logica voor data access
 - Business Logic Layer classes implementeren enkel logica voor business rules
 - ⇒ Layers of isolation
 - Wijzigingen in implementatie binnen een laag hebben geen impact op bovenliggende laag (bv. LINQ query's i.p.v. stored procedure gebruiken in DAL)
 - De ene laag hoeft niets te weten van de interne werking van de onderliggende laag

Concept

Theorie

- Closed principe wordt toegepast
 - Een laag communiceert enkel met de laag 1 niveau lager
 - Communicatie met een laag meer dan 1 laag dieper is niet toegestaan



Concept

Theorie

- Elke layer is een apart project in de solution van de app
- Welk projecttype gebruiken per layer?

Layer	Projecttype
Data Access Layer	⇒ Class Library project (resulteert in een .dll file)
Business Logic Layer	⇒ Class Library project (resulteert in een .dll file)
Service Layer (API)	⇒ ASP.NET Core Web API project (resulteert in een .exe file)
Presentation Layer	⇒ Een UI project voor web, desktop maar kan evengoed een niet .NET project zijn zoals Angular, Vue, React, ...

Business Logic/Service Layer

Aandachtspunten

- N-Tier <> N-layered
- N-Tier
 - Fysiek gescheiden lagen (elk deel bv. op een aparte server)
- N-Layered
 - Logische lagen in de applicatie (bv. DAL, BLL, ...)

N-Layered architectuur

Data Access Layer

Data Access Layer



Doel

- Het juiste project type voor de Data Access Layer kunnen kiezen
- De Data Access Layer kunnen structureren

Data Access Layer

Theorie

- ⇒ Maak een nieuw **Class Library** project aan in je solution voor de Data layer
 - **Geen console app** aangezien dit een apart .exe bestand oplevert!
- ⇒ Gebruik volgende naamgevingsconventie **[company].[application].[component]** (Inclusief de punten als scheidingsteken)
 - ⇒ Waarbij je component vervangt door **DAL** (afkorting voor **Data Access Layer**)
 - ⇒ Of vervangt door **Data**

Data Access Layer

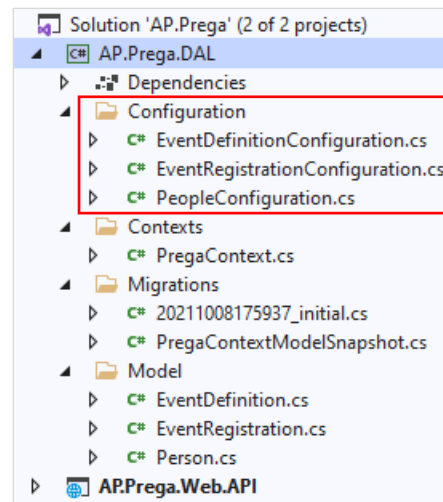
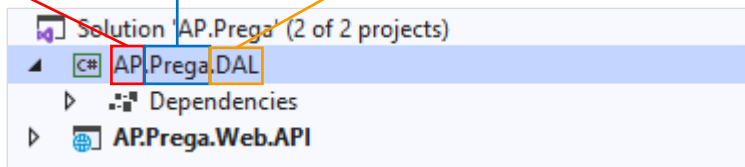
Theorie

- Wat komt er in de Data Access Layer?
 - Map met Entity Configuration classes (Zie les 3)
 - Map met DbContext class(es) (Zie les 2)
 - Map met Entity classes (Zie les 2)
 - Map met Migrations (Gegenereerd bij aanmaken migration)
- Welke reference(s) naar andere projecten?
 - Geen
- Welke NuGet packages
 - Alle packages benodigd voor EF Core (zie les 2)

Data Access Layer

Voorbeelden

Company Application Component
AP Prega DAL



N-Layered architectuur

Business Logic/Service Layer

Business Logic/Service Layer



Doel

- Het juiste project type voor de Business Logic/Service Layer kunnen kiezen
- De Business Logic/Service Layer kunnen structureren
- Gepaste interfaces kunnen definiëren voor service classes
- Service classes kunnen implementeren

Business Logic/Service Layer

Theorie

- ⇒ Maak een nieuw **Class Library** project aan in je solution voor deze layer
- ⇒ Gebruik volgende naamgevingsconventie **[company].[application].[component]** (Inclusief de punten als scheidingsteken)
 - ⇒ Waarbij je component vervangt door **BLL** (afkorting voor **B**usiness **L**ogic **L**ayer)
 - ⇒ Of vervangt door **Service**

Business Logic/Service Layer

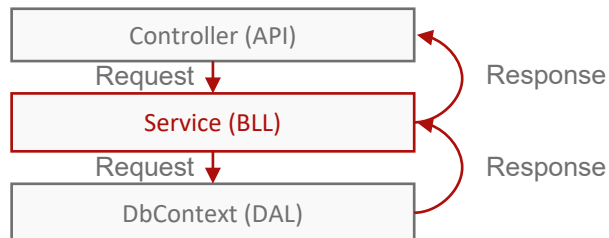
Theorie

- Wat komt er in de Business Logic/Service Layer?
 - Map met Interfaces voor de Service classes
 - Map met Service classes
- Welke reference(s) naar andere projecten?
 - Project reference naar de Data Access Logic Layer

Business Logic/Service Layer

Theorie

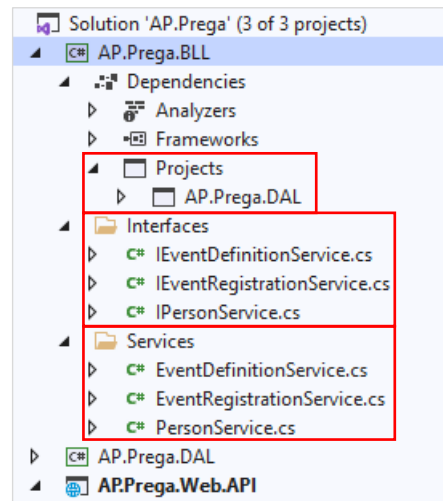
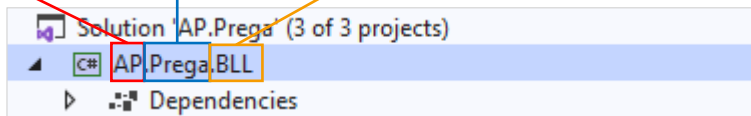
- Opbouw van de laag
 - Interfaces
 - Specificeren welke methods geïmplementeerd moeten worden
 - Faciliteren Dependency Injection in het API project en betere testbaarheid code
 - Interface per service class
 - Service classes
 - Elke class implementeert de bijhorende interface en bevat business logic code
 - Fungeren ook als doorgeefluik tussen DAL en API



Business Logic/Service Layer

Voorbeelden

Company Application Component
AP Prega BLL



Business Logic/Service Layer

Voorbeelden

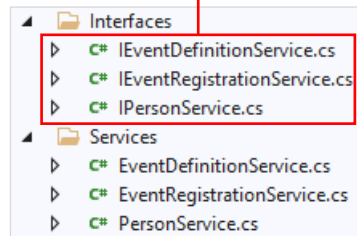
Interface

```
1 using AP.Prega.DAL.Model;
2 using System.Collections.Generic;
3
4 namespace AP.Prega.BLL.Interfaces
5 {
6     public interface IPeopleService
7     {
8         public List<Person> GetAll();
9
10        public Person GetById(int id);
11
12        public Person Add (Person person);
13
14        public void Delete(int id);
15
16        public void Update(Person person);
17    }
18 }
19
```

Let op de naamgeving van de interface

⇒ I[naam entiteit]Service

Interface per service class



Business Logic/Service Layer

Voorbeelden

Implementatie

```
1  using AP.Prega.BLL.Interfaces;
2  using AP.Prega.DAL.Contexts;
3  using AP.Prega.DAL.Model;
4  using System.Collections.Generic;
5  using System.Linq;
6
7  namespace AP.Prega.BLL.Services
8  {
9      2 references
10     public class PeopleService : IPeopleService
11     {
12         private PregaContext _pregaContext;
13
14         0 references
15         public PeopleService(PregaContext pregaContext)
16         {
17             _pregaContext = pregaContext;
18         }
19
20         2 references
21         public List<Person> GetAll()
22         {
23             return _pregaContext.People.ToList();
24         }
25     }
26 }
```

Service class implementeert de bijhorende interface

DbContext wordt geïnjecteerd via de constructor en vervolgens toegewezen aan een private member variable

Gebruik de private DbContext variable om informatie op te halen of terug te sturen

N-Layered architectuur

Custom exceptions voor datavalidatie

Custom exceptions voor datavalidatie



Doel

- In functie van datavalidatie
 - Het concept van custom exceptions kennen
 - Custom exceptions kunnen creëren
 - Custom en standaard exceptions kunnen throwen

Custom exceptions voor datavalidatie

Theorie

- Datavalidatie gebeurt deels in de front-end en zeker in de back-end
 - Vermijden van fouten door te laten bij rechtstreeks gebruik API zonder front-end
- Wordt in de business logic layer geïmplementeerd aan de back-end zijde
- Bij niet voldoen aan een validatie wordt een exception gethrowed
- Exception kan standaard of custom exception zijn

Custom exceptions voor datavalidatie

Theorie

- Custom exceptions worden gedefiniëerd in de Business Logic/Service Layer
 - Maak eventueel een aparte folder hiervoor aan
- Voorzie een beschrijvende naam (Engels!)
- Laat custom Exception class overerven van Exception class
- Voorzie indien nodig een specifieke constructor waaraan benodigde info wordt meegegeven

Custom exceptions voor datavalidatie

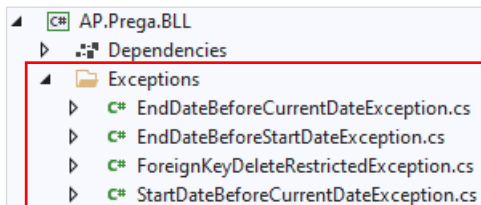
Voorbeelden

Custom exception

```
public class EndDateBeforeCurrentDateException : Exception
{
    0 references
    public EndDateBeforeCurrentDateException()
    {
    }

    2 references
    public EndDateBeforeCurrentDateException(DateTime endDate)
        : base($"Enddate {endDate} lies before the current date.")
    {
    }
}
```

Erft over van Exception



```
AP.Prega.BLL
├── Dependencies
└── Exceptions
    ├── EndDateBeforeCurrentDateException.cs
    ├── EndDateBeforeStartDateException.cs
    ├── ForeignKeyDeleteRestrictedException.cs
    └── StartDateBeforeCurrentDateException.cs
```

Plaats de custom exceptions in een map

Custom exceptions voor datavalidatie

Voorbeelden

Validaties in de Create() method van de EventDefinitionService class

```
public EventDefinition Create(EventDefinition eventDefinition)
{
    if (eventDefinition.StartsOn < DateTime.Now)
    {
        throw new StartDateBeforeCurrentDateException(eventDefinition.StartsOn);
    }

    if (eventDefinition.EndsOn < DateTime.Now)
    {
        throw new EndDateBeforeCurrentDateException(eventDefinition.EndsOn);
    }

    if (eventDefinition.EndsOn < eventDefinition.StartsOn)
    {
        throw new EndDateBeforeStartDateException(eventDefinition.StartsOn, eventDefinition.EndsOn);
    }

    if (eventDefinition.Venue.Length > 80)
    {
        throw new FormatException("Venue length exceeded 80 characters");
    }
}
```

Custom exceptions

Standaard exception

N-Layered architectuur

API Layer

API Layer



Doel

- De API Layer kunnen structureren
- Controllers kunnen implementeren
- Custom en standaard exceptions kunnen opvangen en omzetten naar een gepast ActionResult

API Layer

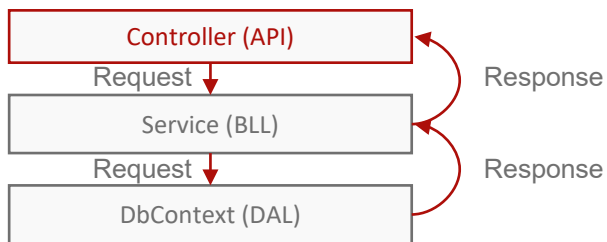
Theorie

- Wat komt er in de API Layer?
 - Controllers
- Welke reference(s) naar andere projecten?
 - Project reference naar de Data Access Logic Layer
 - Project reference naar de Business Logic/Service Layer

API Layer

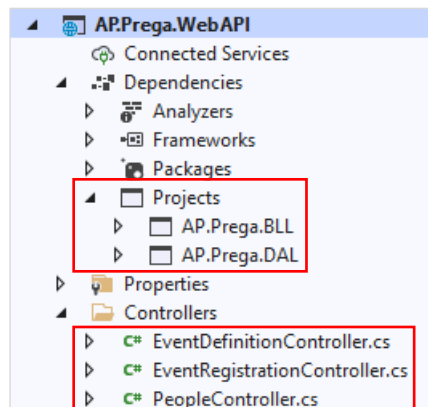
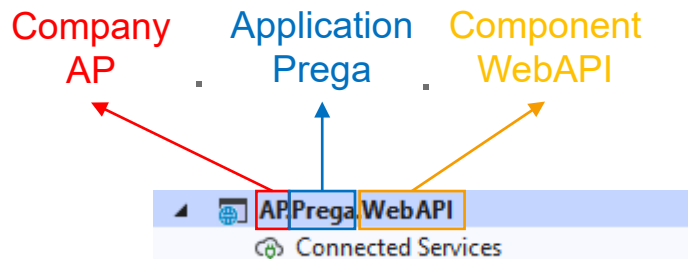
Theorie

- Opbouw van de laag
 - Controllers
 - Implementeren logica om requests te verwerken
 - Implementeren logica om juiste respons terug te geven in functie van resultaat
 - Communiceren met de Service classes van de Business Logic/Service Layer



API Layer

Voorbeelden



API Layer



Voorbeelden

Implementatie

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class PeopleController : ControllerBase
{
    private IPeopleService _peopleService;

    0 references
    public PeopleController(IPeopleService peopleService)
    {
        _peopleService = peopleService;
    }

    // GET: api/<PeopleController>
    [HttpGet]
    0 references
    public IActionResult GetAll()
    {
        var result = _peopleService.GetAll();

        if (result.Any() == false)
        {
            return NotFound();
        }

        return Ok(result);
    }
}
```

Service class van de BLL laag wordt geïnjecteerd via de constructor en vervolgens toegewezen aan een private member variable

Gebruik de private Service variable om te communiceren met de Business Logic/Service Layer

API Layer

Voorbeelden

Omzetting Exception naar ActionResult

```
public void Delete(int id)
{
    Person person = _pregaContext.People.Find(id);
    if (person != null)
    {
        _pregaContext.Remove<Person>(person);
        _pregaContext.SaveChanges();
    }
    else
    {
        throw new KeyNotFoundException("");
    }
}
```

```
// DELETE api/<PeopleController>/5
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    try
    {
        _peopleService.Delete(id);
        return NoContent();
    }
    catch (KeyNotFoundException ex)
    {
        return NotFound();
    }
}
```

- We controleren of de person met het gegeven id bestaat
- Indien niet throwen we een KeyNotFoundException
- Deze exception wordt opgevangen in de controller en omgezet naar ActionResult NotFound

N-Layered architectuur

Refactoring services registratie

Refactoring registratie services

Doel

- Registreren van Layer specifieke services binnenin de layer

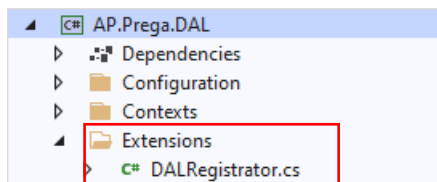
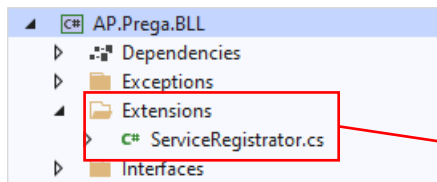
Refactoring registratie services

Theorie

- Vermijden van cluttering van Configuration() method in Startup.cs
- Registratie van services via extension method in static class van betreffende Layer
 - Bv. ServiceRegistrar class in Business Logic/Service Layer
 - Eventueel in map Extensions plaatsen
- Wat registreren?
 - (Voorlopig) enkel DbContext in Data Logic Layer
 - Alle service classes in Business Logic/Service Layer

Refactoring registratie services

Voorbeelden



```
public static class ServiceRegistrator
{
    1 reference
    public static IServiceCollection RegisterServices(this IServiceCollection services)
    {
        services.AddScoped<IEventDefinitionService, EventDefinitionService>();
        services.AddScoped<IEventRegistrationService, EventRegistrationService>();
        services.AddScoped<IPeopleService, PeopleService>();

        return services;
    }
}
```

Oefeningen

Deel 7 - Oefenbundel 1

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

N-Layered architectuur



Bronnen

- Microsoft
 - <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/names-of-assemblies-and-dlls>
- Oreilly
 - <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>