

Invoices oefening

TOM LAPERRE

Inhoud

Inhoud	2
Requirements	3
Opbouw	3
Invoice Solution	3
Voor we van start gaan.	4
Invoice Solution – We gaan van start	4
Invoice DataLayer	6
De Business/Orchestration Layer	18
Presentation Layer	28
Beveiligen van de APIs	32
UnitTesting	40
Testen van de APIs	40

- AUTHENTICATIE WEGHALEN UIT INVOICE (beveiligen van de APIS naar nieuw project en implementeren in api gateway)
- Testen in layers implementeren

Requirements

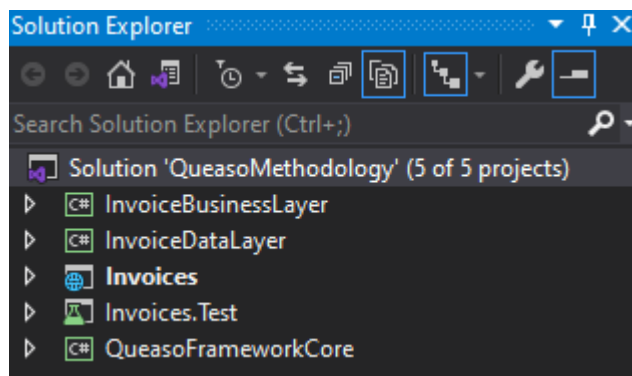
- Visual studio 2019 Community edition of hoger:
(<https://visualstudio.microsoft.com/downloads/>).
- Sqlserverlocaldb of standalone SQL:
(Developer edition: <https://www.microsoft.com/nl-nl/sql-server/sql-server-downloads>).
- Internetconnectie
(Package download)
- SSMS
(<https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studiossms?view=sql-server-ver15>)
- Postman
(<https://www.postman.com/downloads/>)

Opbouw

De opdracht gaan we oplossen in een 3-Tier Model. Het volledige project bestaat uit :

- Een **DataLayer**; verantwoordelijk voor alle database communicatie.
- Een **BusinessLayer** (OrchestrationLayer); alle logische bewerkingen op de data ifv de het businessmodel.
- Een **MVC** layer; waarin we alle methodes die we naar de buitenwereld willen openstellen definiëren.

Voor het testen van de applicatie gaan we op twee manieren te werk. Via unittests, hier gebruiken we NUNIT en via de Postman app voor de MVC tests. Alle unittesten steken we in een apart project



Invoice Solution

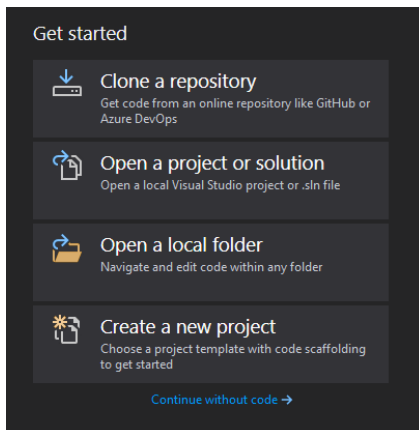
Dit is en blijft een oefening. Vraag steeds bij je klant naar de interne afspraken rond de code. Ofwel is deze er en dan volg je deze. Ofwel handel je volgens jou kennis 'Best practices'. Er doen bv. veel discussies de ronde wanneer en waar camelCase of PascalCase te gebruiken, wanneer enkel en wanneer meervoud, in welke taal enz. Verlies er geen tijd mee door te moeten herschrijven, bevraag het. Het zelfde is van toepassing voor waar worden variabelen gedeclareerd. Sommigen zweren bij, bovenaan in een class, sommige na een constructor en sommige binnen een '#region'. Veelal word hier niet meer naar gekeken. Het ziet er misschien iets netter uit in theorie, maar dit is old school. In de praktijk declareert men deze meestal, waar men ze nodig heeft. Het over en weer scrollen naar een declaratiesectie neemt op zich al tijd in, en die is er meestal niet. Het is veel handiger de F12 toets te leren gebruiken. Dit geldt trouwens voor alles shortcuts, het bespaart je enorm veel tijd eens je er een paar onder de knie hebt.

Azure DevOps

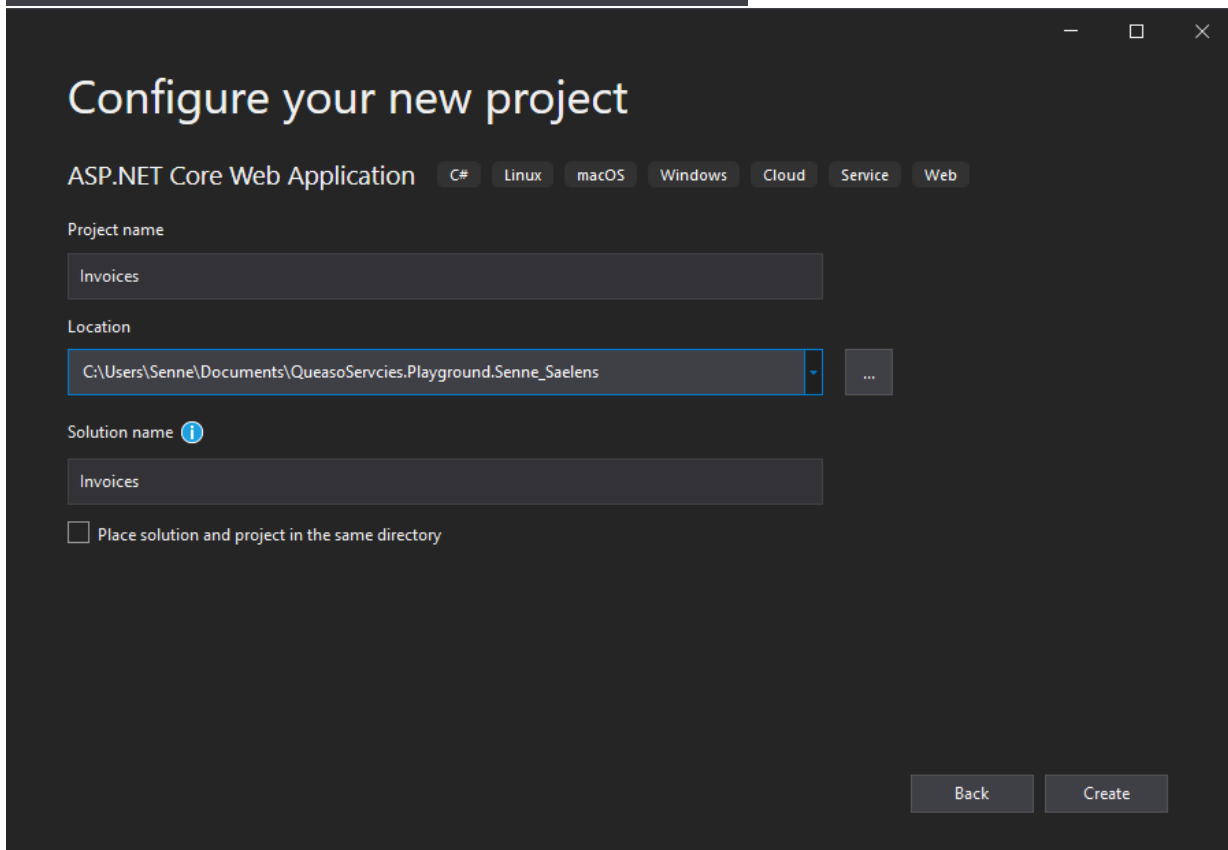
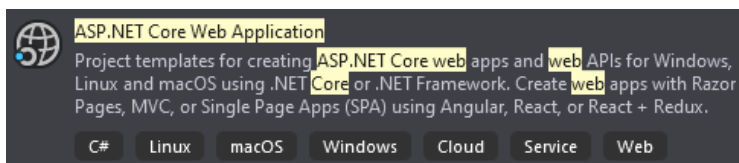
Van Filip heb je een account gekregen op Azure DevOps. We gaan eerst zorgen dat deze gelinkt staat aan Visual Studio. Zorg eerst dat je DevOps account geactiveerd is. Daarna log je in Visual Studio in met je gelinkt account. Normaal kan je nu vlot verbinden met DevOps.

Invoice Solution – We gaan van start

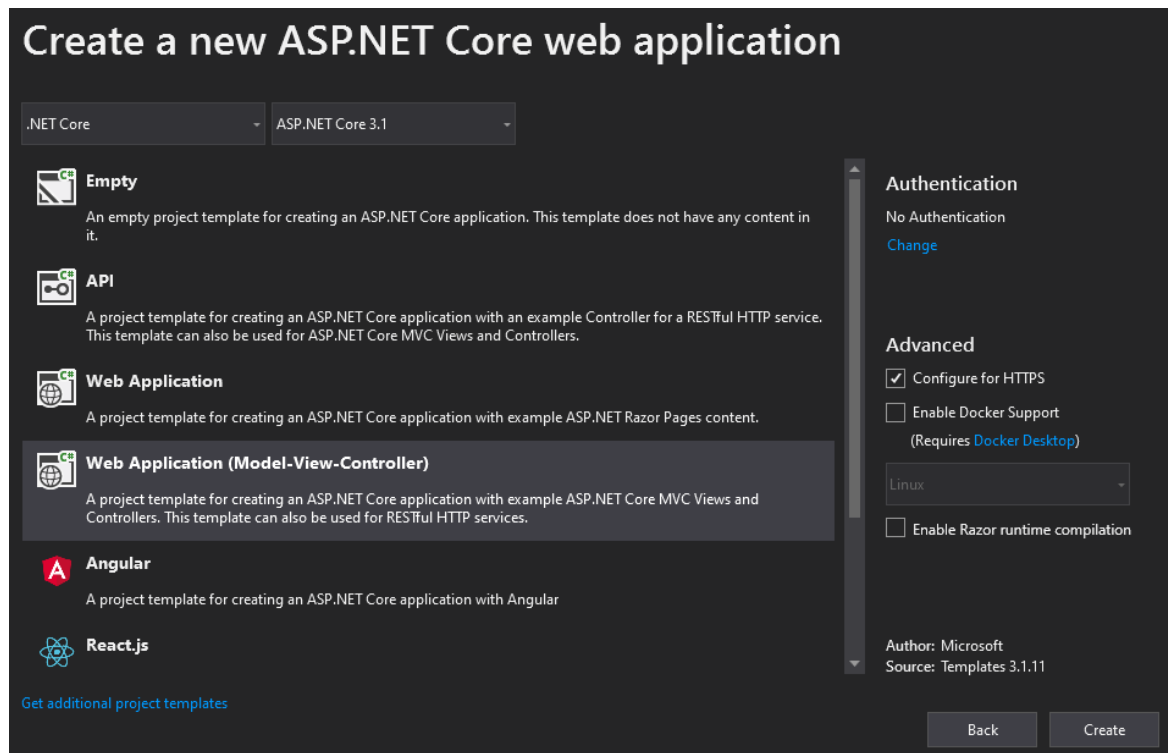
Open Visual Studio 2019 en klik op “Create a new project”.



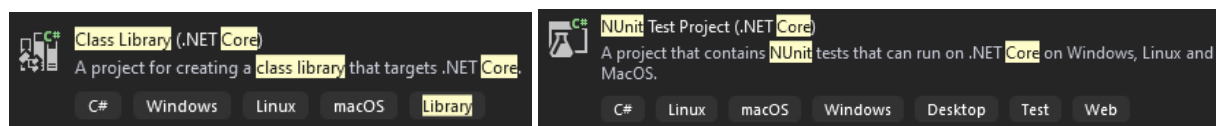
Maak een ASP.NET Core Web Application genaamd Invoices in een nieuwe Invoices solution. Plaats deze lokaal al in dezelfde map als QueasoMethodology.



Kies nu het type applicatie Web Application (Model-View-Controller) en bij de Advanced sectie vink je "Configure for HTTPS" aan.

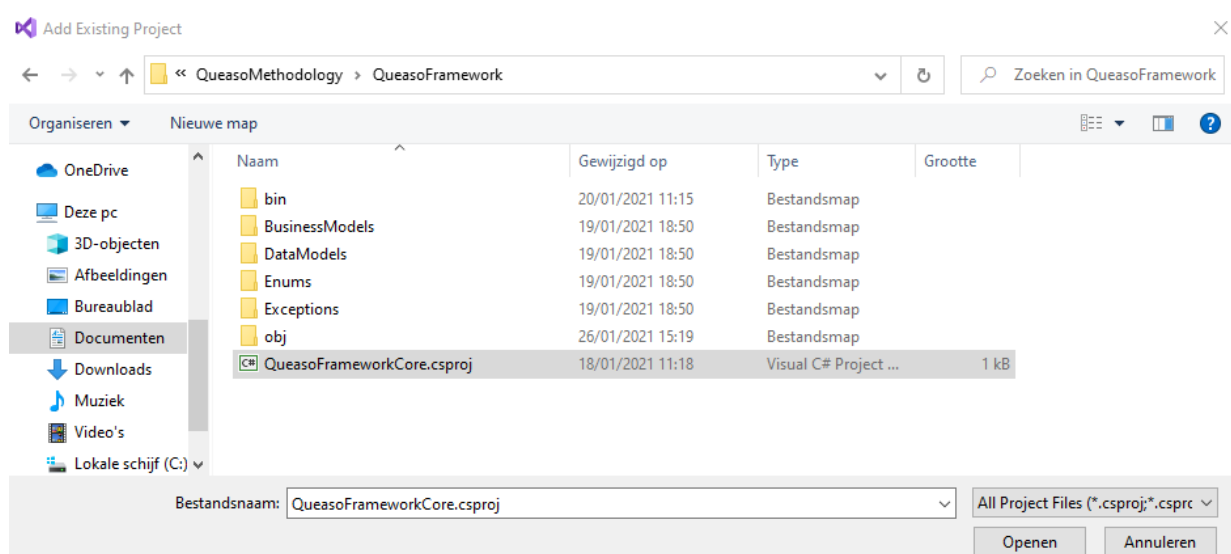


Voeg nu 2 Class Library (.NET Core) projecten toe genaamd InvoiceDataLayer en InvoiceBusinessLayer in dezelfde Solution. Voeg hierna een nieuw NUnit Core Test project aan de solution Invoices.Test toe.



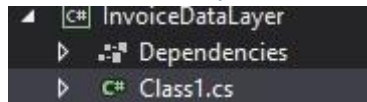
Nu voegen we nog het QueasoFrameworkCore project toe.

Rechtermuisklik op Solution > Add > Existing Project



Hiermee is de structuur van de solution af. We starten met de DataLayer.

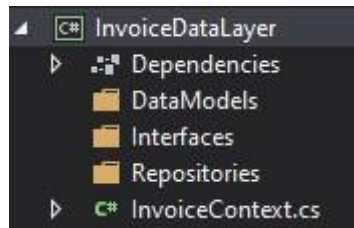
Invoice DataLayer



De default aangemaakte Class1 verwijderen we en we voorzien 3 mappen.

- **DataModels:** Hierin zullen de models van onze tabellen komen
- **Repositories:** Hierin alle methodes die we op de individuele tabellen zullen kunnen toepassen.
- **Interfaces:** Doorheen de volledige solution gaan we gebruik maken van DI (Dependency Injection) dus gaan we ook interfaces naar de implementeren voor de repositories.

Onder de root van project maken we een class 'InvoiceContext.cs' aan waarin we DbContext gaan configureren (CTRL+Shift+A InvoiceContext)

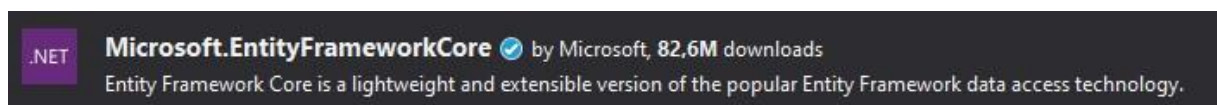


Lets code

Binnen InvoiceContext.cs maken we eerst onze class public en voegen de base constructors (ctor tab) toe om gebruik te maken van Entity framework Core, voeg daar DbContext aan toe als baseclass:

```
namespace InvoiceDataLayer
{
    4 references
    public class InvoiceContext: DbContext
    {
        0 references
        public InvoiceContext() { }
        0 references
        public InvoiceContext(DbContextOptions<InvoiceContext> options) : base(options) { }
    }
}
```

Selecteer de tekst DbContextOptions en druk CTRL+; of ALT+ENTER om de ontbrekende package 'Microsoft.EntityFrameworkCore' toe te voegen. Als alternatief: Tools-NuGet Packet Manager- Manage NuGet Packages for solution , Browse Microsoft.EntityFrameworkCore



En voeg de namespace toe.

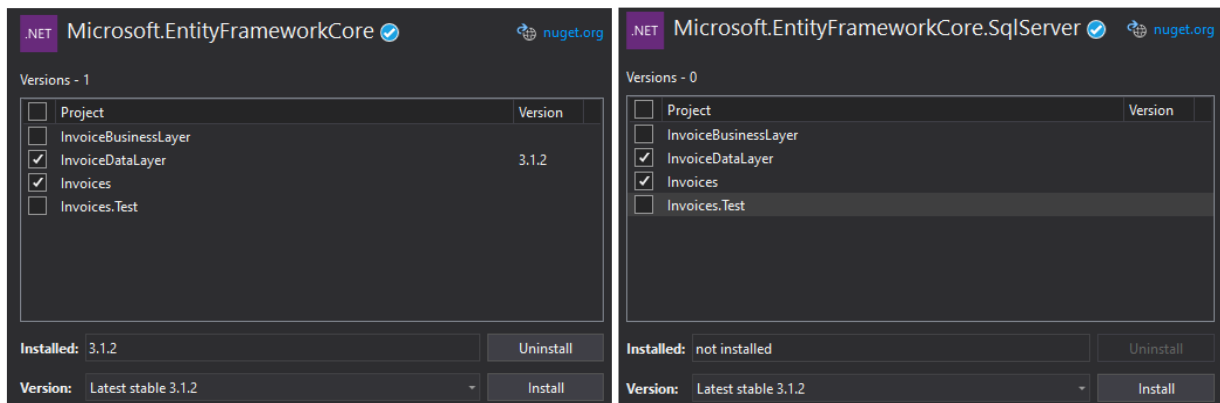
```
using Microsoft.EntityFrameworkCore;
```

Nu springen we even naar ons Invoice MVC project waar de te gebruiken database voor onze InvoiceContext gaan configureren. We passen hier voor in Startup.cs de services aan en geven aan welke provider en welke database we wensen te gebruiken/aanmaken/updaten.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();

    services.AddDbContext<InvoiceContext>(opts => opts.UseSqlServer(Configuration.GetConnectionString("InvoiceDb"),
        b => b.MigrationsAssembly("Invoices")).EnableSensitiveDataLogging());
}
```

Voeg de namespace InvoiceDataLayer toe. Maw we gaan een databasecontext toevoegen voor het type InvoiceContext en gaan hiervoor en SqlServer gebruiken. De configuratie voor de connectie komt uit ons configuratiebestand appsettings.json in de key InvoiceDb. Migratiescripts zullen worden opgeslaan in het project Invoices. Om dit aan opgestart te krijgen dienen we eerst de provider zijn framework (package) te voorzien, via NuGet Package Manager zoek je naar EntityFrameworkCore en voegt dit toe aan de 2 database gerelateerde projecten:



Pas nu ifv jou situatie de connectionstring aan. Ok dit is enkel voor de oefening dit doe je normaal niet, credentials komen dan van een safe keystore.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "InvoiceDb": "Server=WEBSVR;Database=Invoices;User ID=QueasoUsr;password=QueasoUsr;"
  }
}
```

Ook hier zal je meerdere dispuutjes horen, moet mijn db nu Invoice of Invoices noemen. Persoonlijk ga ik steeds uit van de logica, zitten daar meerdere dingen van het zelfde soort in, zeg en schrijf het dan ook zo.

Nu is duidelijk welke provider we gaan gebruiken voor onze DB, welke configuratie en waar migratiescripts moeten komen. Terug naar onze DataLayer om de inhoud van de database te bepalen.

Voor de eenvoud van de opdracht hebben we voor het maken van een factuur nood aan een paar zaken. Een factuurhoofding met daaraan gekoppeld een aantal factuurlijnen. Ontstaat er in het proces een fout, dan willen we deze bijhouden.

We gaan deze structuur nu eerst vastleggen. Maak daarvoor in de InvoiceDataLayer.DataModels volgende public classes aan (GetSet sneltoets => prop tab tab)

```

namespace InvoiceDataLayer.DataModels
{
    [Table("InvoiceHeader")]
    21 references
    public class DO_InvoiceHeader:DataObjectBase
    {
        [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        9 references
        public Guid Id { get; set; }
        2 references
        public decimal Amount { get; set; }
        2 references
        public decimal TotalAmount { get; set; }
        2 references
        public decimal VatAmount { get; set; }
        4 references
        public string VatNumber { get; set; }
        1 reference
        public bool IsPaid { get; set; }
        1 reference
        public double InvoiceNumber { get; set; }

        //navigation
        12 references
        public List<DO_InvoiceLine> Invoicelines { get; set; }
    }
}

```

```

namespace InvoiceDataLayer.DataModels
{
    [Table("InvoiceLine")]
    11 references
    public class DO_InvoiceLine:DataObjectBase
    {
        [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        1 reference
        public Guid Id { get; set; }
        3 references
        public decimal Amount { get; set; }
        2 references
        public string Description { get; set; }
        3 references
        public decimal LineAmount { get; set; }
        2 references
        public decimal PricePerUnit { get; set; } = 0;
        2 references
        public decimal Quantity { get; set; }
        3 references
        public decimal VatAmount { get; set; }
        2 references
        public decimal VatRate { get; set; }

        //navigation
        2 references
        public DO_InvoiceHeader InvoiceHeader { get; set; }
    }
}

```

```

namespace InvoiceDataLayer.DataModels
{
    [Table("InvoiceException")]
    10 references
    public class DO_InvoiceException:DataObjectBase
    {
        [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        2 references
        public Guid Id { get; set; }
        1 reference
        public int Type { get; set; }
        2 references
        public string Namespace { get; set; }
        2 references
        public string UseCase { get; set; }
        2 references
        public string Message { get; set; }
        2 references
        public string InputParameters { get; set; }
    }
}

```

Voor de creatie van een uniek nummer voor de factuur gebruiken we onderstaande class. Dit is maar één mogelijke implementatie manier, het kan uiteraard eenvoudiger.


```

[Table("InvoiceNumber")]
3 references
public class DO_InvoiceNumber : DataObjectBase
{
    0 references
    public int LastUsedNumber { get; set; } = 1;
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public Guid Id { get; set; }
    public static volatile DO_InvoiceNumber instance;
    public static object syncRoot = new object();
    0 references
    public static DO_InvoiceNumber Instance
    {
        get
        {
            if (instance == null)
            {
                lock (syncRoot)
                {
                    if (instance == null) { instance = new DO_InvoiceNumber(); }
                }
            }
            return instance;
        }
    }
}

```

Alle gemeenschappelijke velden/properties komen in een baseclass terecht:

```

namespace InvoiceDataLayer.DataModels
{
    0 references
    public abstract class DataObjectBaseInvoice
    {
        #region Creation
        0 references
        public string CreatedBy { get; set; }
        0 references
        public DateTime CreatedOn { get; set; } = DateTime.Now;
        #endregion

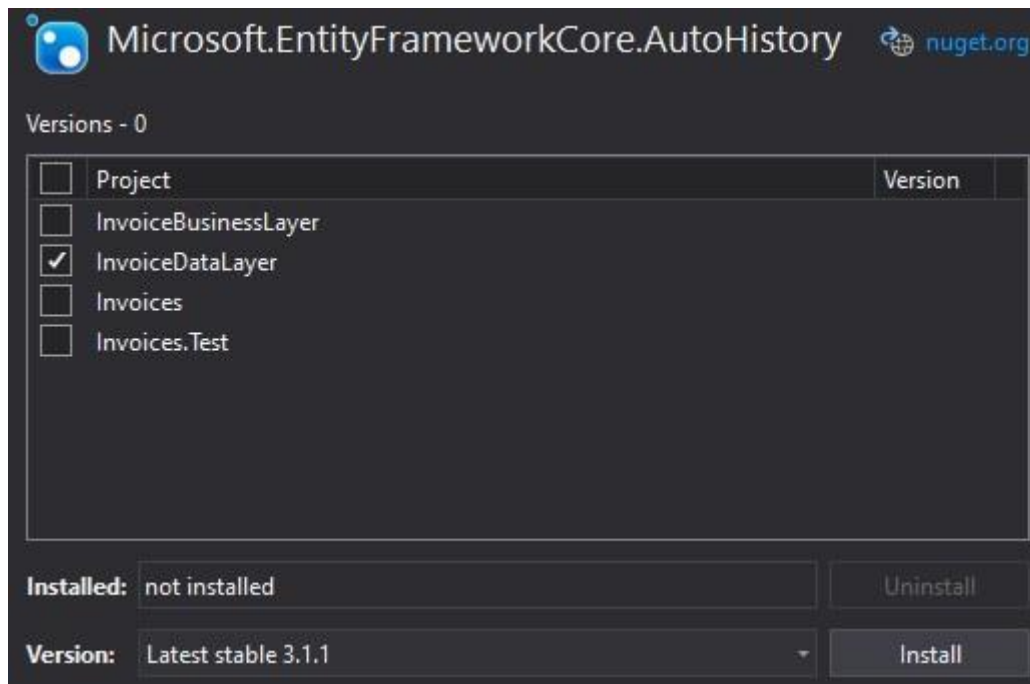
        #region Updated
        0 references
        public string UpdatedBy { get; set; }
        0 references
        public DateTime UpdatedOn { get; set; }
        #endregion

        #region Deleted
        0 references
        public bool IsDeleted { get; set; } = false;
        0 references
        public string DeletedBy { get; set; } = null;
        0 references
        public DateTime? DeletedOn { get; set; } = null;
        #endregion
    }
}

```

Hiermee zijn alle tabelstructuren vastgelegd en dienen we deze enkel nog te koppelen aan de DbContext in InvoiceContext.cs

We gaan per model een dataset aanmaken en base index voorzien op de Guid van het model. We gaan ook zorgen dat EF de change history gaat bijhouden.



De volledige class ziet er dan als volgt uit:

```
public class InvoiceContext: DbContext
{
    // References
    public InvoiceContext() { }
    // References
    public InvoiceContext(DbContextOptions<InvoiceContext> options) : base(options) { }

    // References
    public DbSet<DO_InvoiceHeader> InvoiceHeader { get; set; }
    // References
    public DbSet<DO_InvoiceLine> InvoiceLine { get; set; }
    // References
    public DbSet<DO_InvoiceNumber> InvoiceNumber { get; set; }
    // References
    public DbSet<DO_InvoiceException> InvoiceException { get; set; }

    // References
    protected override void OnModelCreating(ModelBuilder mb)
    {
        mb.EnableAutoHistory(int.MaxValue);
        mb.Entity<DO_InvoiceHeader>().HasIndex(i => i.Id);
        mb.Entity<DO_InvoiceLine>().HasIndex(i => i.Id);
        mb.Entity<DO_InvoiceNumber>().HasIndex(i => i.Id);
        mb.Entity<DO_InvoiceException>().HasIndex(i => i.Id);

        base.OnModelCreating(mb);
    }
}
```

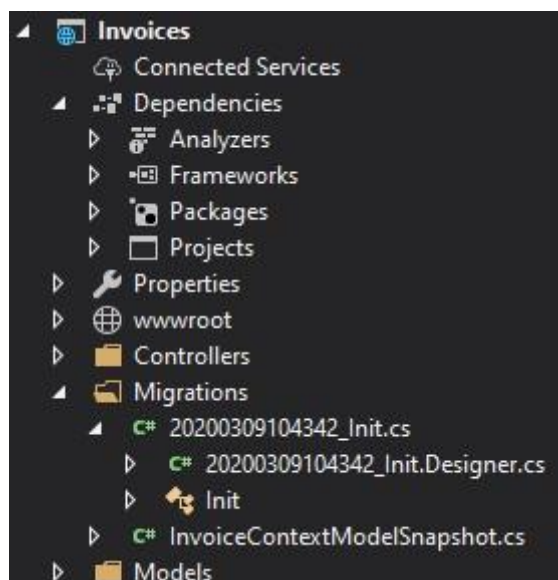
Hiermee kan Entity Framework weg om de database te bouwen, te migreren naar vorige en nieuwe versies en er mee te communiceren. Build de solution (CTRL + B) en indien foutloos, start NuGet Package Manager – Packet Manager Console.

Om de migratiescript te genereren gaan we in het migratieproject “b.MigrationsAssembly(“Invoices”))” gebruik maken van het framework Tools pakket (dus eerst package installeren).

Dan binnen PM console “Add-Migration Init” uitvoeren op het default project “Invoices”

```
PM> Add-Migration Init
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[10400]
    Sensitive data logging is enabled. Log entries and exception messages may include sensitive application data.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'Amount' on entity type 'DO_InvoiceHeader'. This will use the default
    SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'TotalAmount' on entity type 'DO_InvoiceHeader'. This will use the default
    the SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'VatAmount' on entity type 'DO_InvoiceHeader'. This will use the default
    the SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'Amount' on entity type 'DO_InvoiceLine'. This will use the default
    SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'LineAmount' on entity type 'DO_InvoiceLine'. This will use the default
    the SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'PricePerUnit' on entity type 'DO_InvoiceLine'. This will use the default
    the SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'Quantity' on entity type 'DO_InvoiceLine'. This will use the default
    SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'VATAmount' on entity type 'DO_InvoiceLine'. This will use the default
    SQL server column type that can accommodate all the values using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column 'VATRate' on entity type 'DO_InvoiceLine'. This will use the default
    SQL server column type that can accommodate all the values using 'HasColumnType()'.
To undo this action, use Remove-Migration.
PM>
```

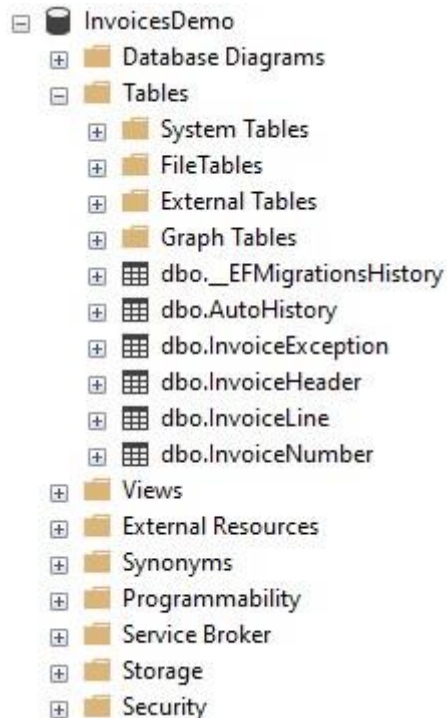
De migratiestructuur werd aangemaakt:



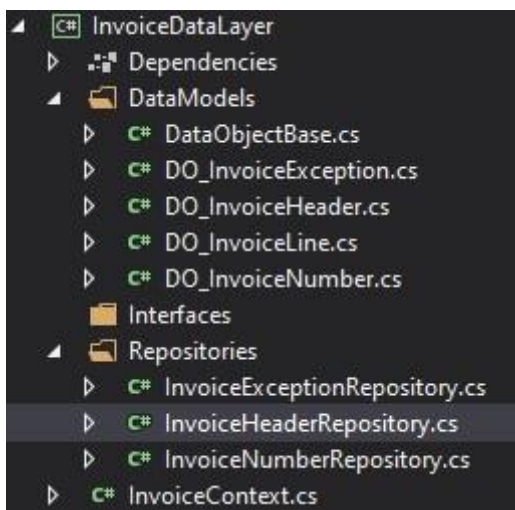
Met de volgende stap kunnen de database aanmaken. Terug binnen PM Console, voer je ‘UpdateDatabase’ uit.

```
PM> Update-Database
Build started...
Build succeeded.
```

Via SSMS kan je dit nazien, de database en structuur werd aangemaakt. Herhaal Add-Migration aNameX, update-database telkens je één of meerdere aanpassingen deed binnen InvoiceDataLayer. De updates zullen vlot verlopen zolang je niet in de knoop geraakt met keys en foreign keys. Is dat het geval, dan is het meestal het handigste de migration folder en je database gewoon weg te gooien en terug een Initiële Add-Migration Init, update-database uit te voeren.



De database is nu opgebouwd, maar je kan er nog niet mee doen. Voor een monolithische App voorzien we meestal de CRUD methodes. Men kan dit indien nodig afslanken uit uitbreiden if van de noden. Alle nodige methodes gaan we aanmaken in de folder InvoiceDataLayer.Repositories gegroepeerd per DBSet. We maken bijgevolg terug public classes aan SHIFT+ALT+A voor de InvoiceHeader, InvoiceException en InvoiceNumber. Voor InvoiceLine hebben we eigenlijk geen class nodig daar we aan de collection van InvoiceLines kunnen via de InvoiceHeader.



We beginnen met InvoiceHeaderRepository. Om met onze Database te kunnen communiceren hebben we nood aan een instantie van onze DbContext. Dus in een de constructor (ctor tab tab)

vragen gewoon aan de services om via DI ons een instantie hiervan ter beschikking te stellen. En dan zijn we vertrokken, we kennen de instantie toe aan een private variabele en kunnen aan alle EF methods ifv onze noden.

```
namespace InvoiceDataLayer.Repositories
{
    3 references
    public class InvoiceHeaderRepository
    {
        private InvoiceContext _invoiceContext;
        2 references
        public InvoiceHeaderRepository(InvoiceContext invoiceContext)
        {
            _invoiceContext = invoiceContext;
        }
    }
}
```

We hebben in ons geval 3 methods nodig, een header maken, updaten en er één ophalen aan de hand van zijn Id. Voor elke method gaan we dit zo veel mogelijk asynchroon doen. In functie van de oefening maken we hier gebruik van Linq. Op dit niveau begint het echter van belang te zijn wat het doel is. Maken we hier een backend die aan de andere kant een front-end heeft type desktop applicatie of eerder Xamarin, Angular? Linq is perfect ok voor responses van een paar tientallen records. Is de result van de query hoger, of is het echt de bedoeling mobile te gaan ontwikkelen, dan is het beter iets verder te ontwikkelen en queries uit te voeren via sql stored procedures, met paging selectors en Dapper ipv Linq. In ons geval zal een InvoiceHeader geen duizenden InvoiceLines terug geven, dus Linq is ok. Onze 3 methoden zien er dan zo uit (CTRL + ; of ALT+ENTER voor de ontbrekende namespaces automatisch toe te voegen).

```
using InvoiceDataLayer.DataModels;
using System;
using System.Threading.Tasks;
using System.Linq;
using Microsoft.EntityFrameworkCore;

namespace InvoiceDataLayer.Repositories
{
    1 reference
    public class InvoiceHeaderRepository
    {
        private InvoiceContext _invoiceContext;

        0 references
        public InvoiceHeaderRepository(InvoiceContext invoiceContext)
        {
            _invoiceContext = invoiceContext;
        }

        0 references
        public async Task<DO_InvoiceHeader> CreateInvoiceHeader(DO_InvoiceHeader entity)
        {
            _invoiceContext.Add(entity);
            await _invoiceContext.SaveChangesAsync();
            return entity;
        }

        0 references
        public async Task<DO_InvoiceHeader> UpdateInvoiceHeader(DO_InvoiceHeader entity)
        {
            _invoiceContext.Update(entity);
            await _invoiceContext.SaveChangesAsync();
            return entity;
        }

        0 references
        public DO_InvoiceHeader GetInvoiceHeaderById(Guid Id)
        {
            return _invoiceContext.InvoiceHeader.Include(l => l.InvoiceLines).FirstOrDefault(i => i.Id == Id);
        }
    }
}
```

Het doel van InvoiceNumberRepository, is het ophalen van een nieuw invoice nummer en dit opslaan in de database.

```

public class InvoiceNumberRepository
{
    private readonly InvoiceContext _invoiceContext;
    0 references
    public InvoiceNumberRepository(InvoiceContext invoiceContext)
    {
        _invoiceContext = invoiceContext;
    }
    0 references
    public int GetNextNumber()
    {
        try
        {
            var invoiceNumber = _invoiceContext.InvoiceNumber.FirstOrDefault();
            if (invoiceNumber != null)
            {
                invoiceNumber.LastUsedNumber += 1;
                _invoiceContext.Entry<DO_InvoiceNumber>(invoiceNumber).State = EntityState.Modified;
                Save();
                return invoiceNumber.LastUsedNumber;
            }
            else
            {
                var newInvoiceNumber = new DO_InvoiceNumber();
                newInvoiceNumber.CreatedBy = ""; //Todo
                newInvoiceNumber.UpdatedBy = ""; //Todo
                newInvoiceNumber.LastUsedNumber = 1;
                _invoiceContext.InvoiceNumber.Add(newInvoiceNumber);
                Save();
                return newInvoiceNumber.LastUsedNumber;
            }
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }
    2 references
    public void Save()
    {
        _invoiceContext.SaveChanges();
    }
}

```

Het doel van de InvoiceExceptionRepository is het ophalen en opslaan van een of meerdere Exceptions.

```

public class InvoiceExceptionRepository
{
    private readonly InvoiceContext _invoiceContext;

    0 references
    public InvoiceExceptionRepository(InvoiceContext invoiceContext)
    {
        _invoiceContext = invoiceContext;
    }

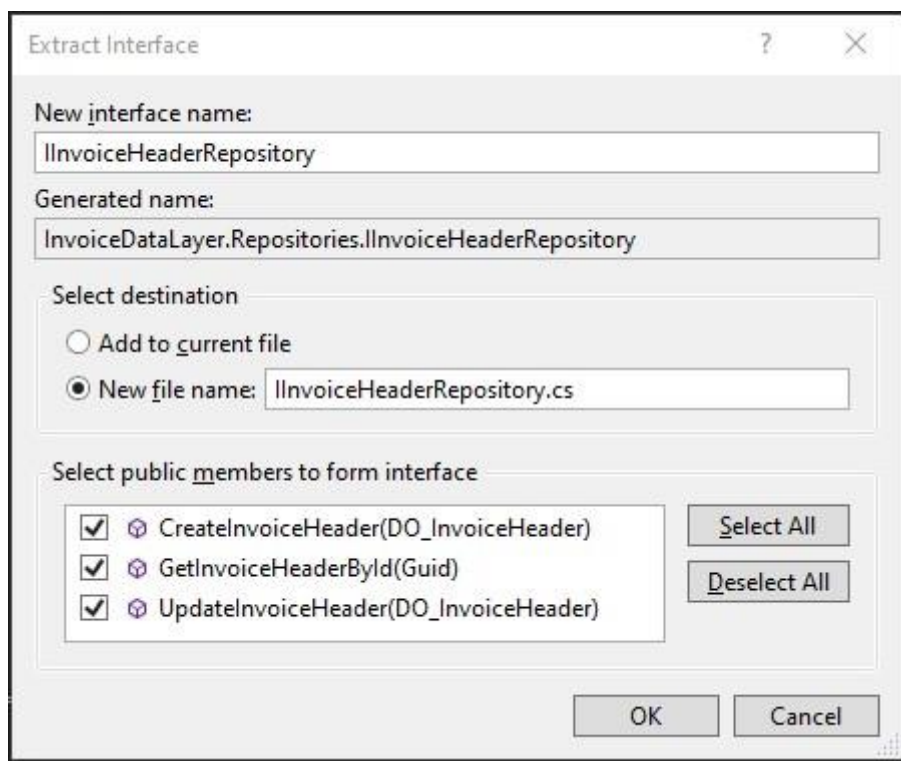
    0 references
    public IEnumerable<DO_InvoiceException> GetExceptions()
    {
        return _invoiceContext.InvoiceException.ToList();
    }

    0 references
    public async Task CreateInvoiceException(DO_InvoiceException entity)
    {
        _invoiceContext.InvoiceException.Add(entity);
        await _invoiceContext.SaveChangesAsync();
    }
}

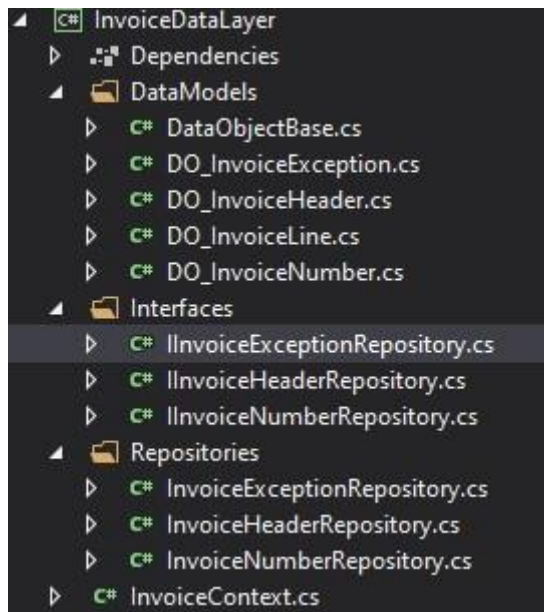
```

Voor onze classes kunnen we nu een interface voorzien en deze ter beschikking stellen via dependency injection (DI)

Sta binnen 'public class **InvoiceHeaderRepository**' in het woord InvoiceHeaderRepository, druk dan CTRL+; en kies in de dropdown 'Extract Interface'



Herhaal voor InvoiceNumberRepository en InvoiceExceptionRepository. Verplaats de 3 interfaces naar de map Interfaces.



Onze DataLayer is bijna af, enkel dienen we nu aan onze applicatie duidelijk maken welke klasse er gebruik dient te worden indien we een interface van een bepaald type vragen. Dit doen we in ons MVC Invoices project. Om geen onoverzichtelijke file te maken, gaan we alle configuratie voor DI in een aparte file onderbrengen. In Startup.cs voegen daartoe volgende toe:

```
services.ConfigureDI Services();
```

En we voegen een map toe Extensions met daarin een nieuwe class

```
namespace Invoices.Extensions
{
    References
    public static class AppServices
    {
        1 reference
        public static void ConfigureDI Services(this IServiceCollection diService)
        {
            #region Context

            #endregion

            #region Repository
            diService.AddScoped<IInvoiceExceptionRepository, InvoiceExceptionRepository>();
            diService.AddScoped<IInvoiceHeaderRepository, InvoiceHeaderRepository>();
            diService.AddScoped<IInvoiceNumberRepository, InvoiceNumberRepository>();
            diService.AddScoped<IApplicationUser, ApplicationUser>();
            #endregion

            #region Orchestration

            #endregion

            #region Controllers

            #endregion
        }
    }
}
```

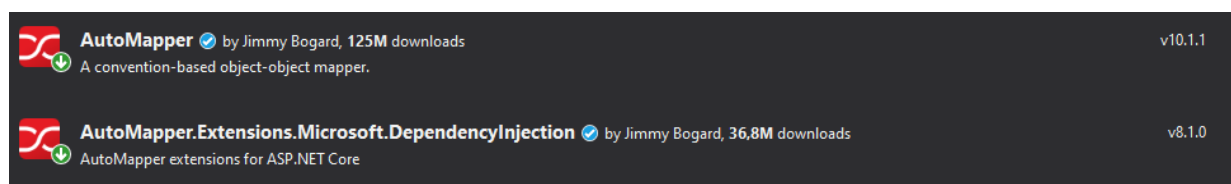

Onze DataLayer is klaar, we kunnen nu ook de gemaakte CRUD methodes uitvoeren op de onderliggende database. Als we nu binnen de BusinessLayer een interface vragen zullen we een instantie krijgen van het gepaste type.

Normaal zouden we nu unittest uitvoeren, we houden dit voor iets later om verwarring te voorkomen.

De Business/Orchestration Layer

Hoeveel layers heb je nodig? Wel dit hangt af van de situatie. Het is meestal NOT DONE om meer logica dan nodig in de PresentationLayer te steken, enkel primaire controle om input objecten, eventueel nog bijkomende authenticatie controle en een return van eventuele data na een call op de business layer. In ons geval kiezen we dus niet, om bijvoorbeeld nog een extra service layer toe te voegen, in een meer complexe toepassing zou dit best het geval kunnen zijn. Er is dus geen eenduidigheid. In deze BusinessLayer gaan we alle logica onderbrengen die de functionele en/of technische analist verwacht en uitgetekend heeft via use cases. Zie enterprise architect van Queaso voor deze oefening.

Het is niet toegelaten om objecten van uit de DataLayer direct aan te spreken in de BusinessLayer. We werken enkel met objecten vanuit de BusinessLayer en gebruikmakend van AutoMapper worden deze omgevormd naar het juiste formaat waarmee de DataLayer dan verder kan. AutoMapper configureren we binnen .Net Core als volgt: installeer volgende packages



In Invoices Startup.cs voegen we de service toe:

```
services.ConfigureDI();
services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());
```

Nu we een mapping service hebben kunnen we deze configureren. Maak in Invoices een nieuwe map aan Automapper, met een nieuwe class AutoMapperConfig.cs

```
namespace Invoices.AutoMapper
{
    1 reference
    public class AutoMapperConfig : Profile
    {
        0 references
        public AutoMapperConfig()
        {
            CreateMap<BO_InvoiceLine, DO_InvoiceLine>();
            CreateMap<DO_InvoiceLine, BO_InvoiceLine>();

            CreateMap<DO_InvoiceHeader, BO_InvoiceHeader>();
            CreateMap<BO_InvoiceHeader, DO_InvoiceHeader>();

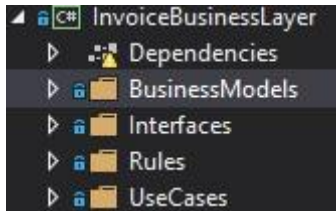
            CreateMap<DO_InvoiceHeader, BO_InvoiceHeader>()
                .ForMember(desc => desc.InvoiceLines, do_ => do_.MapFrom(scr => scr.InvoiceLines));
            CreateMap<BO_InvoiceHeader, DO_InvoiceHeader>()
                .ForMember(desc => desc.InvoiceLines, do_ => do_.MapFrom(scr => scr.InvoiceLines));

            CreateMap<FrameworkException, DO_InvoiceException>();
            CreateMap<DO_InvoiceException, FrameworkException>();
        }
    }
}
```

We maken hier dus steeds een mapping in de twee richtingen voor de Business- en de DataLayer. De modellen zijn echter nog niet gemaakt in de BusinessLayer, maar eens deze objecten bestaan zal AutoMapper de juiste vertaalslag kunnen maken.

Queaso heeft een baseclass (BusinessObjectBase) gemaakt waarmee een aantal businessrules gecontroleerd kunnen worden. Binnen deze BusinessLayer wordt deze geïmplementeerd. Voeg een referentie toe naar het project QueasoFrameworkCore.

Net zoals in de DataLayer maken we nu in het business project de folderstructuur:



We beginnen met de regels op te stellen, in de rules folder maken we InvoiceBusinessRule.cs

```
public class InvoiceBusinessRule : BusinessRule
{
    0 references
    public InvoiceBusinessRule CalculateTotalLine(string propertyName, decimal quantity, decimal amount, out decimal calculatedTotal)
    {
        this.PropertyName = propertyName;
        calculatedTotal = 0;
        try
        {
            calculatedTotal = quantity * amount;
        }
        catch (Exception ex)
        {
            this.Passed = false;
            SetFailedMessage($"An error occurred while setting property {propertyName}: {ex.Message}");
        }
        return this;
    }
    0 references
    public InvoiceBusinessRule CalculateVatAmount_Line(string propertyName, decimal vatRate, decimal amountWithoutVat, out decimal calculatedVatAmount)
    {
        this.PropertyName = propertyName;
        calculatedVatAmount = 0;
        try
        {
            calculatedVatAmount = amountWithoutVat / 100 * vatRate;
        }
        catch (Exception ex)
        {
            this.Passed = false;
            SetFailedMessage($"An error occurred while setting property {propertyName}: {ex.Message}");
        }
        return this;
    }
}
```

```

public InvoiceBusinessRule CalculateTotal_Invoice(string propertyName, List<BO_InvoiceLine> InvoiceLines, out decimal calculatedTotal)
{
    this.PropertyName = propertyName;
    calculatedTotal = 0;
    try
    {
        foreach (var line in InvoiceLines)
        {
            calculatedTotal += line.Amount;
        }
    }
    catch (Exception ex)
    {
        this.Passed = false;
        SetFailedMessage($"An error occurred while setting property {propertyName}: {ex.Message}");
    }
    return this;
}

//reference
public InvoiceBusinessRule CalculateVatAmount_Invoice(string propertyName, List<BO_InvoiceLine> InvoiceLines, out decimal calculatedVatAmount)
{
    this.PropertyName = propertyName;
    calculatedVatAmount = 0;
    try
    {
        foreach (var line in InvoiceLines)
        {
            calculatedVatAmount += line.Amount;
        }
    }
    catch (Exception ex)
    {
        this.Passed = false;
        SetFailedMessage($"An error occurred while setting property {propertyName}: {ex.Message}");
    }
    return this;
}

public InvoiceBusinessRule CheckValidityVatNumber(string propertyName, string vatNumber)
{
    this.PropertyName = propertyName;
    if (!string.IsNullOrEmpty(vatNumber) && !vatNumber.ToUpper().StartsWith("BE0"))
    {
        this.Passed = false;
        SetFailedMessage($"Property {propertyName}: Does not contain a valid VatNumber");
    }
    return this;
}

```

Nu werken we onze modellen uit in de BusinessModels folder, net zoals bij de DataLayer hebben we nu BO_ InvoiceHeader, InvoiceLine en InvoiceNumber nodig. We voorzien BusinessRule die checkt dat er daadwerkelijk een Id is voor de InvoiceLine. Daartoe voegen we een inherit van BusinessObjectBase toe.

```

public class BO_InvoiceLine : BusinessObjectBase
{
    0 references
    public BO_InvoiceLine()
    {
    }
    0 references
    public BO_InvoiceLine(Guid invoiceHeaderId, decimal pricePerUnit, decimal quantity, string description)
    {
        this.InvoiceHeaderID = invoiceHeaderId;
        this.PricePerUnit = pricePerUnit;
        this.Quantity = quantity;
        this.VatRate = VatRate;
        this.Description = description;
    }
    private decimal _Amount = 0;
    private decimal _VatAmount = 0;
    private decimal _LineAmount = 0;

    2 references
    private Guid Id { get; set; }
    1 reference
    public Guid InvoiceHeaderID { get; set; }
    1 reference
    public decimal PricePerUnit { get; set; }
    1 reference
    public decimal Quantity { get; set; }
    0 references
    public decimal Amount { get => _Amount; set => _Amount = value; }
    0 references
    public decimal VatAmount { get => _VatAmount; set => _VatAmount = value; }
    0 references
    public decimal LineAmount { get => _LineAmount; set => _LineAmount = value; }
    2 references
    public decimal VatRate { get; set; }
    1 reference
    public string Description { get; set; }

    1 reference
    public override bool AddBusinessRules()
    {
        BusinessRules.Add(new InvoiceBusinessRule().IsRequired(nameof(this.Id), this.Id));
        return base.AddBusinessRules();
    }
}

```

Idem voor InvoiceHeader, deze dient te voldoen aan een aantal BusinessRules

```

public class BO_InvoiceHeader : BusinessObjectBase
{
    0 references
    public override bool AddBusinessRules()
    {
        BusinessRules.Add(new InvoiceBusinessRule().IsRequired(nameof(this.VatNumber), this.VatNumber));
        BusinessRules.Add(new InvoiceBusinessRule().MaxLength(nameof(this.VatNumber), this.VatNumber, 50));
        BusinessRules.Add(new InvoiceBusinessRule().CheckValidityVatNumber(nameof(this.VatNumber), this.VatNumber));

        BusinessRules.Add(new InvoiceBusinessRule().IsRequired(nameof(this.InvoiceLines), this.InvoiceLines));

        BusinessRules.Add(new InvoiceBusinessRule().CalculateTotal_Invoice(nameof(this.Amount), this.InvoiceLines, out this.amount));
        BusinessRules.Add(new InvoiceBusinessRule().CalculateVatAmount_Invoice(nameof(this.VatAmount), this.InvoiceLines, out this.vatAmount));
        BusinessRules.Add(new InvoiceBusinessRule().GetSum(nameof(this.TotalAmount), new List<decimal>() { this.amount, this.vatAmount }, out this.totalAmount));

        return base.AddBusinessRules();
    }
}

```

```

#region ctor
0 references
public BO_InvoiceHeader()
{
    this.Id = new Guid();
    this.VatNumber = string.Empty;
    this.InvoiceLines = new List<BO_InvoiceLine>();
    this.BusinessRules = new List<QueasoCore.BusinessModels.Rules.BusinessRule>();
}
0 references
public BO_InvoiceHeader(string vatNumber)
{
    this.Id = new Guid();
    this.VatNumber = vatNumber;
    this.InvoiceLines = new List<BO_InvoiceLine>();
    this.BusinessRules = new List<QueasoCore.BusinessModels.Rules.BusinessRule>();
}
#endregion

```

```

#region prop

decimal amount = 0;
decimal vatAmount = 0;
decimal totalAmount = 0;

2 references
public Guid Id { get; set; }
1 reference
public decimal Amount { get => amount; set => amount = value; }
1 reference
public decimal VatAmount { get => vatAmount; set => vatAmount = value; }
1 reference
public decimal TotalAmount { get => totalAmount; set => totalAmount = value; }
0 references
public int InvoiceNumber { get; set; }
8 references
public string VatNumber { get; set; }
0 references
public bool IsPaid { get; set; }
0 references
public int LastUsedNumber { get; set; }
9 references
public List<BO_InvoiceLine> InvoiceLines { get; set; }
#endregion

```

```

#region Methode

/// <summary>
/// Adds an BO_Invoiceline to the header
/// </summary>
/// <param name="invoiceLine"></param>
0 references
public void AddInvoiceLineToHeader(BO_InvoiceLine invoiceLine)
{
    if (InvoiceLines == null)
    {
        InvoiceLines = new List<BO_InvoiceLine>();
    }
    this.InvoiceLines.Add(invoiceLine);
}
#endregion

```

```
namespace InvoiceBusinessLayer.BusinessModels
{
    0 references
    public class BO_InvoiceNumber
    {
        0 references
        public int LastUsedNumber { get; set; }
    }
}
```

Nu de BusinessLayer af is kunnen we AutoMapper binnen het Invoices MVC project verder uitwerken. Leg een dependency naar de BusinessLayer en importeer de namespace. Om de mapping van de exceptions naar de DataLayer te kunnen uitvoeren voeg je ook de namespace QueasoCore.Exceptions toe:

```
0 references
public AutoMapperConfig()
{
    #region BO-DO
    CreateMap<BO_InvoiceLine, DO_InvoiceLine>();
    CreateMap<DO_InvoiceLine, BO_InvoiceLine>();

    CreateMap<DO_InvoiceHeader, BO_InvoiceHeader>();

    CreateMap<BO_InvoiceHeader, DO_InvoiceHeader>();

    CreateMap<DO_InvoiceHeader, BO_InvoiceHeader>()
        .ForMember(desc => desc.InvoiceLines, do_ => do_.MapFrom(scr => scr.InvoiceLines));
    CreateMap<BO_InvoiceHeader, DO_InvoiceHeader>()
        .ForMember(desc => desc.InvoiceLines, do_ => do_.MapFrom(scr => scr.InvoiceLines));

    CreateMap<FrameworkException, DO_InvoiceException>();
    CreateMap<DO_InvoiceException, FrameworkException>();

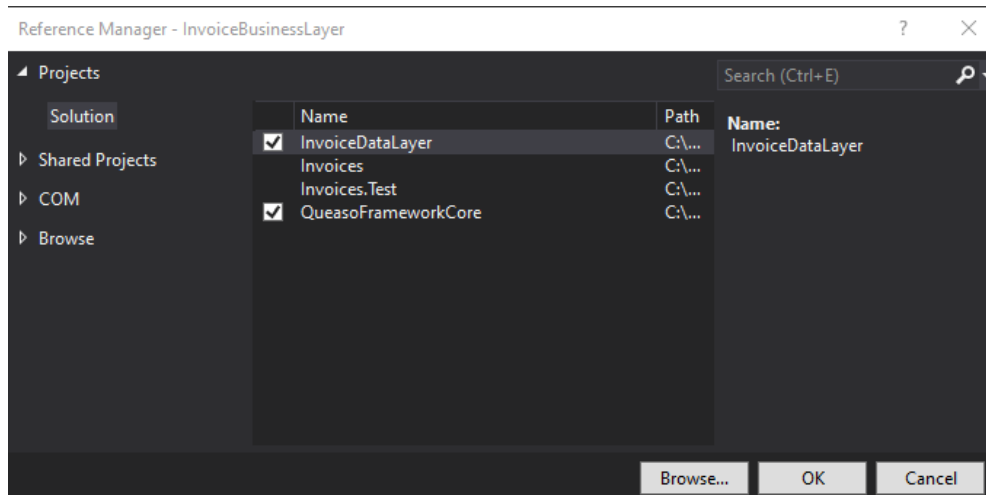
    CreateMap<BO_InvoiceNumber, DO_InvoiceNumber>();
    CreateMap<DO_InvoiceNumber, BO_InvoiceNumber>();

    #endregion

    #region Presentation_BO
    CreateMap<AddInvoiceLineToInvoiceHeaderInput, BO_InvoiceLine>();
    CreateMap<BO_InvoiceLine, AddInvoiceLineToInvoiceHeaderInput >();

    #endregion
}
```

Nu de mappings in orde zijn kunnen we de Use Cases, zoals opgesomd in het enterprise architect invoices project, implementeren. Deze plaatsen we in een folder 'UseCases'. Zoals gezegd is dit in sommige gevallen de naam 'Services'. We maken er een nieuwe klasse aan en leggen ook een reference naar onze DataLayer:



```
using AutoMapper;
using InvoiceBusinessLayer.BusinessModels;
using InvoiceDataLayer.DataModels;
using InvoiceDataLayer.Repositories;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace InvoiceBusinessLayer.UseCases
{
    2 references
    public class InvoiceUseCases
    {
        IInvoiceHeaderRepository _invoiceHeaderRepository;
        IInvoiceNumberRepository _invoiceNumberRepository;
        IInvoiceExceptionRepository _invoiceExceptionRepository;
        IMapper _mapper;
        0 references
        public InvoiceUseCases(IMapper mapper)
        {
            _mapper = mapper;
        }
        0 references
        public InvoiceUseCases(IMapper mapper,
                               IInvoiceHeaderRepository invoiceHeaderRepository,
                               IInvoiceNumberRepository invoiceNumberRepository,
                               IInvoiceExceptionRepository invoiceExceptionRepository)
        {
            _mapper = mapper;
            _invoiceHeaderRepository = invoiceHeaderRepository;
            _invoiceNumberRepository = invoiceNumberRepository;
            _invoiceExceptionRepository = invoiceExceptionRepository;
        }
    }
}
```

We implementeren de UseCases:


```

0 references
public async Task<BO_InvoiceHeader> UC_301_001_CreateInvoiceHeader(string vatNumber)
{
    try
    {
        var invoiceHeaderToCreate = new BO_InvoiceHeader(vatNumber);
        invoiceHeaderToCreate.IsPaid = false;
        invoiceHeaderToCreate.InvoiceNumber = _invoiceNumberRepository.GetNextNumber();
        if (invoiceHeaderToCreate.Valid)
        {
            DO_InvoiceHeader dataObj = _mapper.Map<DO_InvoiceHeader>(invoiceHeaderToCreate);
            dataObj.CreatedBy = "ToDo";
            dataObj.UpdatedBy = "ToDo";
            DO_InvoiceHeader newInvoiceHeader = await _invoiceHeaderRepository.CreateInvoiceHeader(dataObj);
            invoiceHeaderToCreate.Id = dataObj.Id;
        }
        else if (invoiceHeaderToCreate.BrokenRules.Count != 0)
        {
            foreach (var item in invoiceHeaderToCreate.BrokenRules)
            {
                FrameworkException exception = new FrameworkException(invoiceHeaderToCreate, "UC_301_001_CreateInvoiceHeader", "", FrameworkExceptionType.BusinessRuleViolation); //ToDo
                SaveInvoiceException(exception);
            }
        }
        return invoiceHeaderToCreate;
    }
    catch (Exception ex)
    {
        var exception = new FrameworkException("UC_301_001_CreateInvoiceHeader", ex.Message, ex, FrameworkExceptionType.Error);
        SaveInvoiceException(exception);
        throw exception;
    }
}

```

```

public async Task<BO_InvoiceHeader> UC_301_002_AddInvoiceLineToHeader(Guid invoiceHeaderId, BO_InvoiceLine invoiceLine)
{
    try
    {
        if (invoiceLine.Valid)
        {
            BO_InvoiceHeader invoiceHeader = UC_301_003_GetInvoiceById(invoiceHeaderId);
            invoiceHeader.AddInvoiceLineToHeader(invoiceLine);
            if (invoiceHeader.Valid)
            {
                DO_InvoiceHeader invoiceToUpdate = _invoiceHeaderRepository.GetInvoiceHeaderById(invoiceHeader.Id);
                DO_InvoiceLine invoiceLineToAdd = new DO_InvoiceLine()
                {
                    CreatedBy = "ToDo",
                    UpdatedBy = "ToDo",
                    Amount = invoiceLine.Amount,
                    Description = invoiceLine.Description,
                    InvoiceHeader = invoiceToUpdate,
                    LineAmount = invoiceLine.LineAmount,
                    PricePerUnit = invoiceLine.PricePerUnit,
                    Quantity = invoiceLine.Quantity,
                    VATAmount = invoiceLine.VatAmount,
                    VATRate = invoiceLine.VatRate
                };
                if (invoiceToUpdate.InvoiceLines == null) invoiceToUpdate.InvoiceLines = new List<DO_InvoiceLine>();
                invoiceToUpdate.Id = invoiceHeader.Id;
                invoiceToUpdate.Amount = invoiceHeader.Amount;
                invoiceToUpdate.InvoiceNumber = invoiceHeader.InvoiceNumber;
                invoiceToUpdate.IsPaid = invoiceHeader.IsPaid;
                invoiceToUpdate.VatAmount = invoiceHeader.VatAmount;
                invoiceToUpdate.VatNumber = invoiceHeader.VatNumber;
                invoiceToUpdate.TotalAmount = invoiceHeader.TotalAmount;
                invoiceLineToAdd.InvoiceHeader = invoiceToUpdate;
                invoiceToUpdate.InvoiceLines.Add(invoiceLineToAdd);

                await _invoiceHeaderRepository.UpdateInvoiceHeader(invoiceToUpdate);

                return invoiceHeader;
            }
        }
        else if (invoiceLine.BrokenRules.Count != 0)
        {
            foreach (var item in invoiceLine.BrokenRules)
            {
                FrameworkException exception = new FrameworkException(invoiceLine, "UC_301_002_AddInvoiceLineToHeader", "", FrameworkExceptionType.BusinessRuleViolation);
                SaveInvoiceException(exception);
            }
        }
        return null;
    }
    catch (Exception ex)
    {
        FrameworkException exception = new FrameworkException("UC_301_002_AddInvoiceLineToHeader", ex.Message, ex, FrameworkExceptionType.Error); //ToDo
        SaveInvoiceException(exception);
        throw exception;
    }
}

```

```

public BO_InvoiceHeader UC_301_003_GetInvoiceById(Guid id)
{
    try
    {
        return _mapper.Map<BO_InvoiceHeader>(_invoiceHeaderRepository.GetInvoiceHeaderById(id));
    }
    catch (Exception ex)
    {
        FrameworkException exception = new FrameworkException("UC_301_003_GetInvoiceById", ex.Message, ex, FrameworkExceptionType.Error); //ToDo
        SaveInvoiceException(exception);
        throw exception;
    }
}

```

```

6 references
public void SaveInvoiceException(FrameworkException exception, BO_AuthenticatedUser authenticatedUser)
{
    //var dataobject = _mapper.Map<DO_InvoiceException>(exception);
    DO_InvoiceException dataobject = new DO_InvoiceException();
    dataobject.Id = exception.Id;
    dataobject.Type = Convert.ToInt32(exception.Type);
    dataobject.Namespace = exception.Namespace;
    dataobject.UseCase = exception.UseCase;
    dataobject.Message = exception.Message;
    dataobject.InputParameters = exception.InputParameters;
    dataobject.Namespace = exception.Namespace;
    dataobject.UseCase = exception.UseCase;
    dataobject.Message = exception.Message;
    dataobject.InputParameters = exception.InputParameters;

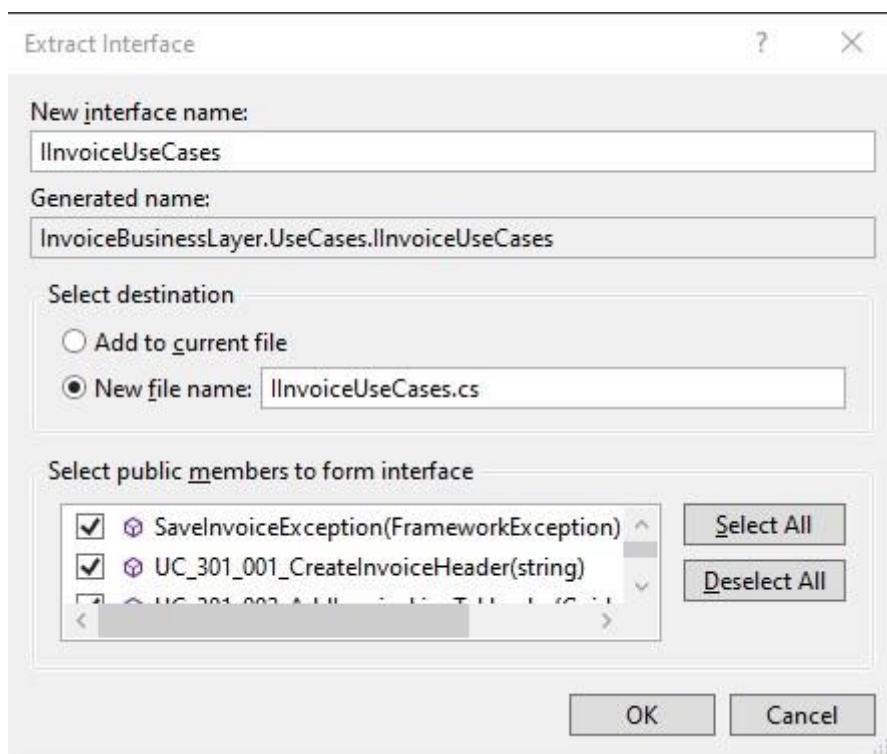
    dataobject.CreatedOn = DateTime.Now;
    dataobject.UpdatedOn = DateTime.Now;
    dataobject.DeletedOn = DateTime.Now;
    dataobject.IsDeleted = false;

    dataobject.CreatedBy = authenticatedUser.userName;
    dataobject.UpdatedBy = authenticatedUser.userName;
    dataobject.DeletedBy = authenticatedUser.userName;

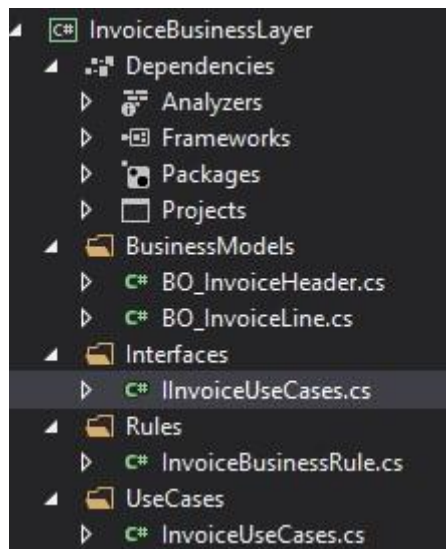
    _invoiceExceptionRepository.CreateInvoiceException(dataobject, _mapper.Map<DO_AuthenticatedUser>(authenticatedUser));
}

```

De UseCases zijn klaar, we bouwen er een interface voor zodat we er via DI in de presentation layer aan kunnen. CTRL+; op de class naam => extract interface:



We verplaatsen de interface naar de Interfaces folder



We voegen nu, net als bij de repositories, de interface toe aan AppServices.cs voor DI.

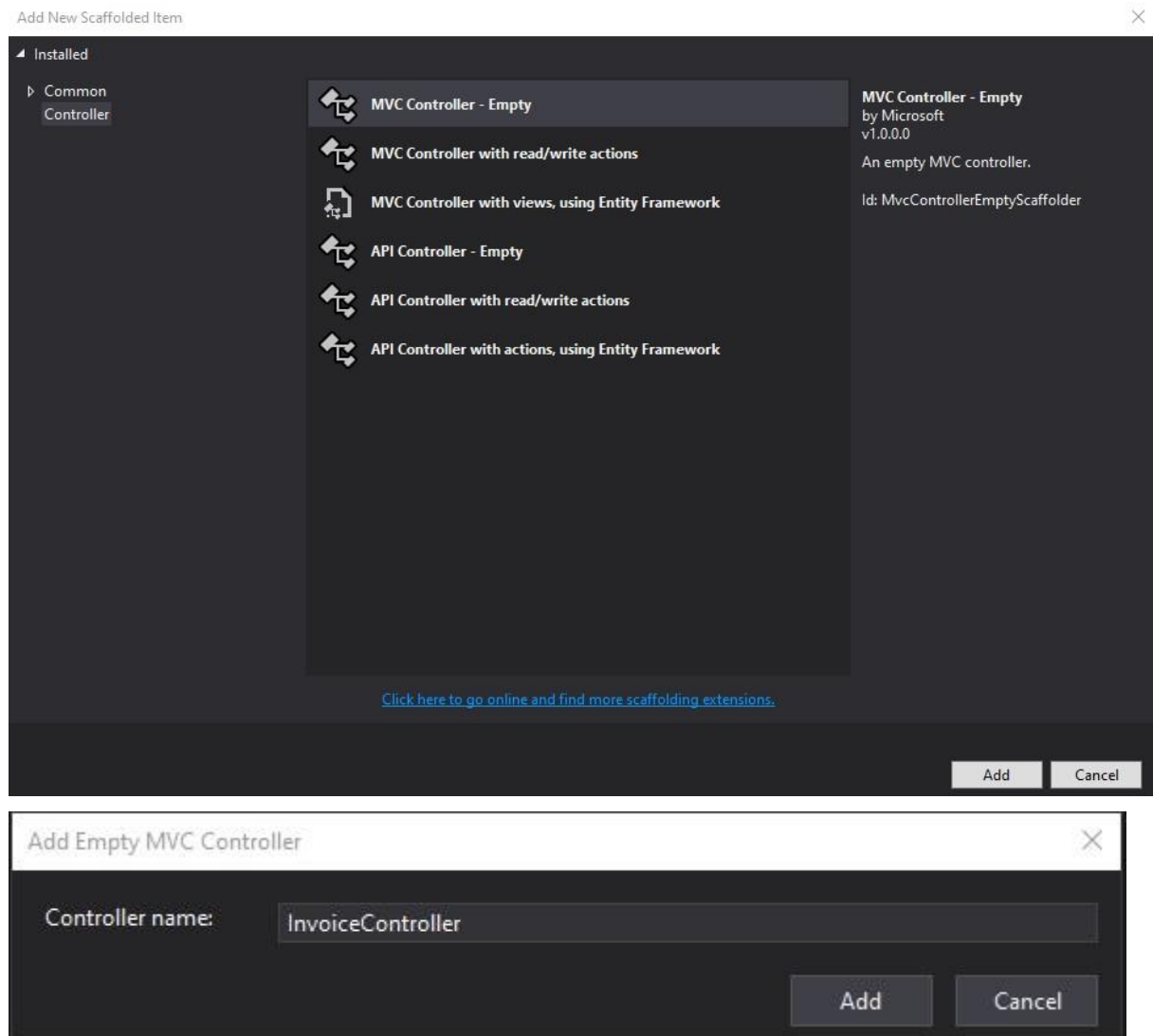
```
#region Orchestration
diService.AddScoped<IInvoiceUseCases, InvoiceUseCases>();
#endregion
```

Net zoals bij de DataLayer kan nu gekozen worden om unittest te schrijven op de BusinessLayer. We houden dit voor de duidelijkheid tot later.

We kunnen nu verder gaan naar de laatste 'Tier' voor onze backend.

Presentation Layer

Hiervoor werken we op ons Invoices MVC project waar we een nieuwe lege controller toevoegen voor Invoices : Controllers map - Add Controller.



Via DI geven we alle interfaces een private readonly variable. Hieronder zijn er een paar meer opgenomen dan nodig. Deze zijn voorzien om onze API calls later secure te kunnen maken, voorlopig doen we er nog niets mee:

```

namespace Invoices.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class InvoiceController : Controller
    {
        private readonly IConfiguration _config;
        private readonly UserManager<IdentityUser> _userManager;
        private readonly SignInManager<IdentityUser> _signInManager;

        private readonly IApplicationUser _applicationUser;
        private readonly IInvoiceUseCases _invoiceUseCases;
        private readonly IMapper _mapper;

        0 references
        public InvoiceController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager,
            IConfiguration configuration, IApplicationUser applicationUser, IInvoiceUseCases invoiceUseCases, IMapper mapper)
        {
            _mapper = mapper;
            _config = configuration;
            _userManager = userManager;
            _signInManager = signInManager;
            _applicationUser = applicationUser;
            _invoiceUseCases = invoiceUseCases;
        }
    }
}

```

Voeg een klasse ApplicationUser toe in de map Extentions en voeg hiervoor dependency injection toe aan AppServices.

```

2 references
public class ApplicationUser
{
    private readonly IHttpContextAccessor _httpContextAccessor;
    0 references
    public ApplicationUser(IHttpContextAccessor httpContextAccessor)
    {
        _httpContextAccessor = httpContextAccessor;
    }
    0 references
    public string Name => this.GetContextUserName();
    1 reference
    private string GetContextUserName()
    {
        return _httpContextAccessor.HttpContext.User.Claims.First(c => c.Type == "Name").Value;
    }
}

```

```
diService.AddScoped<IApplicationUser, ApplicationUser>();
```

En om aan onze HTTP context te geraken voegen we het volgende toe in startup:

```
services.TryAddSingleton<IHttpContextAccessor, HttpContextAccessor>();
```

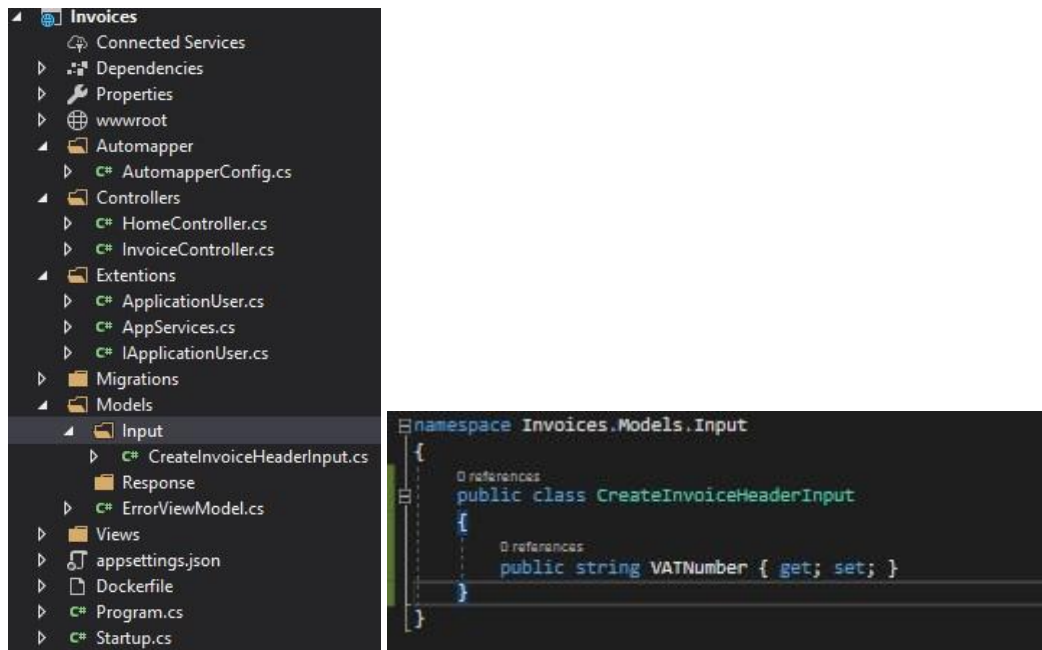
Onze eerste Api call maakt een InvoiceHeader aan en het resultaat van het object wordt onder de vorm van een Json string teruggestuurd.

```

[AllowAnonymous] // [Authorize]
[HttpPost("CreateInvoiceHeader")]
0 references
public async Task<IActionResult> CreateInvoiceHeader(CreateInvoiceHeaderInput createInvoiceHeaderInput)
{
    if (!ModelState.IsValid)
    {
        return StatusCode(400);
    }
    var result = await _invoiceUseCases.UC_301_001_CreateInvoiceHeader(createInvoiceHeaderInput.VATNumber);
    return Ok(result);
}

```

Om dat te zien of de post in het correcte formaat is maken we een class CreateInvoiceHeaderInput in de Models.Input van het project.



Idem om een nieuwe lijn aan de invoice toe te voegen:

```
[AllowAnonymous] // [Authorize]
[HttpPost("AddInvoiceLineToInvoiceHeader")]
public async Task<ActionResult> AddInvoiceLineToInvoiceHeader(AddInvoiceLineToInvoiceHeaderInput createInvoiceHeaderInput)
{
    if (!ModelState.IsValid)
    {
        return StatusCode(400);
    }
    var result = await _invoiceUseCases.UC_301_002_AddInvoiceLineToHeader(createInvoiceHeaderInput.InvoiceHeaderID, _mapper.Map<BO_InvoiceLine>(createInvoiceHeaderInput));
    return Ok();
}
```

```
namespace Invoices.Models.Input
{
    1reference
    public class AddInvoiceLineToInvoiceHeaderInput
    {
        1reference
        public Guid InvoiceHeaderID { get; set; }
        0references
        public decimal Amount { get; set; }
        0references
        public string Description { get; set; }
        0references
        public Guid InvoiceLineID { get; set; }
        0references
        public decimal LineAmount { get; set; }
        0references
        public decimal PricePerUnit { get; set; }
        0references
        public decimal Quantity { get; set; }
        0references
        public decimal VATAmount { get; set; }
        0references
        public decimal VATRate { get; set; }
    }
}
```

Voor de mapping voegen we aan AutoMapperConfig.cs het volgende toe:

```
#region Presentation-BO
CreateMap<AddInvoiceLineToInvoiceHeaderInput, BO_InvoiceLine>();
CreateMap<BO_InvoiceLine, AddInvoiceLineToInvoiceHeaderInput>();
#endregion
```

Voor de laatste methode om een volledige invoice op te halen:

```

[AllowAnonymous] //[Authorize]
[HttpPost("GetInvoiceByName")]
public IActionResult GetInvoiceByName(GetInvoiceByNameInput getInvoiceByNameInput)
{
    if (!ModelState.IsValid)
    {
        var result = _invoiceUseCases.UC_301_003_GetInvoiceById(getInvoiceByNameInput.InvoiceHeaderId);
        return Ok(result);
    }
    return Ok();
}

```

```

namespace Invoices.Models.Input
{
    public class GetInvoiceByNameInput
    {
        public Guid InvoiceHeaderId { get; set; }
    }
}

```

De volledige applicatie zou nu moeten draaien.

Daar we niet wensen dat iedereen zomaar onze API's kan aanroepen, gaan we nu security implementeren.

Beveiligen van de APIs

Er zijn uiteraard vele manieren om te beveiligen. De in onze oefening geïmplementeerde API security is volledig gebaseerd op deze youtube video:

<https://www.youtube.com/watch?v=I56YLbAVAfo&list=WL&index=33>

Het laat ons toe een gebruiker te registreren via API en in- en uit te loggen. Na een succesvolle login ontvangen we een Token, die we verder kunnen gebruiken in de reeds gemaakte InvoiceHeader en InvoiceLine API's. Vervang eerst in de InvoiceController [AllowAnonymous] door [Authorize]

```
[Authorize] // [AllowAnonymous]
[HttpPost("CreateInvoiceHeader")]
```

Door de implementatie worden extra tabellen toegevoegd aan de database om vanaf de eerste user zich aanmeldt.

In de InvoiceController voegen we volgende methods toe:

```
[HttpPost("Register")]
public async Task<ActionResult> Register(RegisterUserInput registerUserInput)
{
    IdentityUser identityUser = new IdentityUser(registerUserInput.userName);
    //if (!_signInManager.IsSignedIn())
    var lookupUser = _userManager.FindByNameAsync(registerUserInput.userName);
    if (lookupUser.Result == null)
    {
        var newAccount = await _userManager.CreateAsync(identityUser, registerUserInput.password);
        if (newAccount.Succeeded)
        {
            var invoiceClaims = new List<Claim>() { new Claim(ClaimTypes.Name, registerUserInput.userName) };
            var invoiceIdentity = new ClaimsIdentity(invoiceClaims, "Invoice Identity");
            ClaimsIdentity invoiceIdentity = new ClaimsIdentity(invoiceClaims, CookieAuthenticationDefaults.AuthenticationScheme);
            ClaimsPrincipal newPrincipal = new ClaimsPrincipal(new[] { invoiceIdentity });
            var signInResult = await _signInManager.PasswordSignInAsync(registerUserInput.userName, registerUserInput.password, false, false);
            if (signInResult != null)
            {
                return Ok("Registration successfull");
            }
            //await HttpContext.SignInAsync(newPrincipal);
        }
    }
    else
    {
        var signInResult = await _signInManager.PasswordSignInAsync(registerUserInput.userName, registerUserInput.password, false, false);
        if (signInResult.Succeeded)
        {
            return Ok("user was already registered - loggedin");
        }
        else
        {
            return Ok("login failed for user");
        }
    }
    return Ok("not signed in");
}
```

De input class RegisterUserInput:

```
public class RegisterUserInput
{
    5 references
    public string userName { get; set; }
    3 references
    public string password { get; set; }
}
```



```

[HttpPost("Login")]
0 references
public async Task<IActionResult> Login(RegisterUserInput registerUserInput)
{
    IActionResult response = Unauthorized();
    var user = await AuthenticateUser(registerUserInput);
    if (user != null)
    {
        var token = GenerateJsonWebToken(user);
        response = Ok($"token = {token}");
    }
    return response;
}

[HttpPost("Logout")]
0 references
public IActionResult Logout()
{
    _signInManager.SignOutAsync();
    return Ok();
}

```

Installeer :



System.IdentityModel.Tokens.Jwt by Microsoft, 213M downloads
Includes types that provide support for creating, serializing and validating JSON Web Tokens.

v6.8.0



Microsoft.IdentityModel.JsonWebTokens by Microsoft, 125M downloads
Includes types that provide support for creating, serializing and validating JSON Web Tokens.

v6.8.0

```

1 reference
private string GenerateJsonWebToken(RegisterUserInput user)
{
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Json:Secret"]));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha512);

    var invoiceClaims = new List<Claim>() { new Claim(ClaimTypes.Name, user.userName) };

    var token = new JwtSecurityToken(
        issuer: _config["Json:Issuer"],
        audience: _config["Json:Issuer"],
        claims: invoiceClaims,
        expires: DateTime.Now.AddMinutes(2),
        signingCredentials: credentials
    );
    return new JwtSecurityTokenHandler().WriteToken(token);
}

1 reference
private async Task<RegisterUserInput> AuthenticateUser(RegisterUserInput registerUserInput)
{
    var signInResult = await _signInManager.PasswordSignInAsync(registerUserInput.userName, registerUserInput.password, false, false);
    if (signInResult.Succeeded)
    {
        return registerUserInput;
    }
    return null;
}

```

Let op dat je bij `_config` de `Microsoft.Extensions.Configuration` gebruikt en niet de `AutoMapper.Configuration`.

```

//using AutoMapper.Configuration;
using Microsoft.Extensions.Configuration;

```

Hiermee zijn de basisfunctionaliteiten voor registratie en authenticatie voorzien. De nodige services om deze te gebruiken moeten nog voorzien worden in `Startup.cs`

Bij `Configure` voegen we `UseAuthentication` toe:

```

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}

```



Voor cross origine support voegen we aan ConfigureServices

```

services.AddCors(conf => conf.AddPolicy("CorsPol", builder =>
    builder.AllowAnyOrigin()
        .AllowAnyHeader()
        .AllowAnyMethod()
        .AllowCredentials()
        .Build()));

```

Voor Authenticatie:


Microsoft.AspNetCore.Authentication.JwtBearer
 by Microsoft, 39,5M downloads
 v3.1.2

ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.

```

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(o => o.TokenValidationParameters = new TokenValidationParameters()
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuersSigningKey = true,

        ValidIssuer = Configuration["Json:Issuer"],
        ValidAudience = Configuration["Json:Issuer"],
        IssuersSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Json:Secret"]))
    });

```

En we voegen de nodige waarden toe aan onze appsettings.json. Als demo! deze moeten van een secure locatie komen voor productie.

```

"Json": {
  "Secret": "TheKeyOfInvoices",
  "Issuer": "Queaso.be"
}

```

Voor het configureren van de extra tabellen:


Microsoft.AspNetCore.Identity.EntityFrameworkCore
 by Microsoft, 30,1M downloads
 v3.1.2

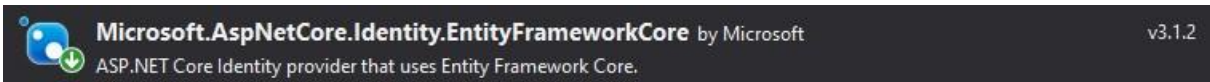
ASP.NET Core Identity provider that uses Entity Framework Core.

```

services.AddIdentity<IdentityUser, IdentityRole>(
    config => {
        config.Password.RequiredLength = 4;
        config.Password.RequireDigit = false;
        config.Password.RequireUppercase = false;
        config.Password.RequireNonAlphanumeric = false;
    }
)
.AddEntityFrameworkStores<InvoiceContext>()
.AddDefaultTokenProviders();

```

Op onze DataLayer moeten we enkel aangeven dat onze baseclass niet meer de gewone DbContext is, maar IdentityDbContext. Daarvoor installeren we op de DataLayer volgend package



En passen dan aan:

```

namespace InvoiceDataLayer
{
    13 references
    public class InvoiceContext : IdentityDbContext //Using DB authentication (services.AddIdentity...)
    //public class InvoiceContext:DbContext //Normal Db
    {

```

Tot zover de configuratie voor security. Aanpassen van de BusinessLayer methods om de userinfo (in ons geval enkel de naam) door te geven is nu de volgende stap. Ook hier is er veel discussie over de beste manier implementatie. Voor de oefening kiezen we voor een object RegisterUserInput voor het aanmelden.

```

namespace Invoices.Models.Input
{
    5 references
    public class RegisterUserInput
    {
        7 references
        public string userName { get; set; }
        4 references
        public string password { get; set; }
    }
}

```

Bij iedere methode roepen we de login 'GetCurrentUser' aan die ons het object teruggeeft of null. Voeg deze toe aan CustomerController.

```

0 references
private CON_AuthenticatedUser GetcurrentUser()
{
    var identity = HttpContext.User.Identity as ClaimsIdentity;
    IList<Claim> claims = identity.Claims.ToList();
    CON_AuthenticatedUser User = new CON_AuthenticatedUser() { Id = Guid.Parse(claims[0].Value), userName = claims[1].Value };
    return User;
}

```

```

namespace Invoices.Models.Input
{
    3 references
    public class CON_AuthenticatedUser
    {
        1 reference
        public Guid Id { get; set; }
        1 reference
        public string userName { get; set; }
    }
}

```

We voorzien deze op elke layer:

```
namespace InvoiceBusinessLayer.BusinessModels
{
    1 reference
    public class BO_AuthenticatedUser
    {
        0 references
        public Guid Id { get; set; }
        0 references
        public string userName { get; set; }
    }
}

namespace InvoiceDataLayer.DataModels
{
    0 references
    public class DO_AuthenticatedUser
    {
        0 references
        public Guid Id { get; set; }
        0 references
        public string userName { get; set; }
    }
}
```

En passen AutoMapperConfig.cs aan

```
CreateMap<CON_AuthenticatedUser, BO_AuthenticatedUser>();
CreateMap<BO_AuthenticatedUser, CON_AuthenticatedUser>();
CreateMap<BO_AuthenticatedUser, DO_AuthenticatedUser>();
CreateMap<DO_AuthenticatedUser, BO_AuthenticatedUser>();
```

We passen onze IInvoiceUseCase interface aan zodat een AuthenticatedUser meegegeven moet worden:

```
namespace InvoiceBusinessLayer.UseCases
{
    4 references
    public interface IInvoiceUseCases
    {
        0 references
        void SaveInvoiceException(FrameworkException exception, BO_AuthenticatedUser authenticatedUser);
        1 reference
        Task<BO_InvoiceHeader> UC_301_001_CreateInvoiceHeader(string vatNumber, BO_AuthenticatedUser authenticatedUser);
        1 reference
        Task<BO_InvoiceHeader> UC_301_002_AddInvoiceLineToHeader(Guid invoiceHeaderId, BO_InvoiceLine invoiceLine, BO_AuthenticatedUser authenticatedUser);
        1 reference
        BO_InvoiceHeader UC_301_003_GetInvoiceById(Guid id, BO_AuthenticatedUser authenticatedUser);
    }
}
```

De methodes op de BusinessLayer die deze implementeren passen we ook aan. Nu kunnen we tegelijk de velden CreatedBy , UpdatedBy juist zetten (de todo comments in de code)

```
public async Task<BO_InvoiceHeader> UC_301_001_CreateInvoiceHeader(string vatNumber, BO_AuthenticatedUser authenticatedUser)
{
    try
    {
        var invoiceHeaderToCreate = new BO_InvoiceHeader(vatNumber);
        invoiceHeaderToCreate.IsPaid = false;
        invoiceHeaderToCreate.InvoiceNumber = _invoiceNumberRepository.GetNextNumber();
        if (invoiceHeaderToCreate.Valid)
        {
            DO_InvoiceHeader dataObj = _mapper.Map<DO_InvoiceHeader>(invoiceHeaderToCreate);
            dataObj.CreatedBy = authenticatedUser.userName;
            dataObj.UpdatedBy = authenticatedUser.userName;
            DO_InvoiceHeader newInvoiceHeader = await _invoiceHeaderRepository.CreateInvoiceHeader(dataObj);
            invoiceHeaderToCreate.Id = dataObj.Id;
        }
        else if (invoiceHeaderToCreate.BrokenRules.Count != 0)
        {
            foreach (var item in invoiceHeaderToCreate.BrokenRules)
            {
                FrameworkException exception = new FrameworkException(invoiceHeaderToCreate, "UC_301_001_CreateInvoiceHeader", "", FrameworkExceptionType.BusinessRuleViolation);
                SaveInvoiceException(exception, authenticatedUser);
            }
        }
        return invoiceHeaderToCreate;
    }
    catch (Exception ex)
    {
        var exception = new FrameworkException("UC_301_001_CreateInvoiceHeader", ex.Message, ex, FrameworkExceptionType.Error);
        SaveInvoiceException(exception, authenticatedUser);
        throw exception;
    }
}
```



```

public async Task<BO_InvoiceHeader> UC_301_002_AddInvoiceLineToHeader(Guid invoiceHeaderId, BO_InvoiceLine invoiceLine, BO_AuthenticatedUser authenticatedUser)
{
    try
    {
        if (invoiceLine.Valid)
        {
            BO_InvoiceHeader invoiceHeader = UC_301_003_GetInvoiceById(invoiceHeaderId, authenticatedUser);
            invoiceHeader.AddInvoiceLineToHeader(invoiceLine);
            if (invoiceHeader.Valid)
            {
                DO_InvoiceHeader invoiceToUpdate = _invoiceHeaderRepository.GetInvoiceHeaderById(invoiceHeader.Id);
                DO_InvoiceLine invoiceLineToAdd = new DO_InvoiceLine()
                {
                    CreatedBy = authenticatedUser.userName,
                    UpdatedBy = authenticatedUser.userName,
                    Amount = invoiceLine.Amount,
                    Description = invoiceLine.Description,
                    InvoiceHeader = invoiceToUpdate,
                    LineAmount = invoiceLine.LineAmount,
                    PricePerUnit = invoiceLine.PricePerUnit,
                    Quantity = invoiceLine.Quantity,
                    VATAmount = invoiceLine.VatAmount,
                    VATRate = invoiceLine.VatRate
                };
                if (invoiceToUpdate.InvoiceLines == null) invoiceToUpdate.InvoiceLines = new List<DO_InvoiceLine>();
                invoiceToUpdate.Id = invoiceHeader.Id;
                invoiceToUpdate.Amount = invoiceHeader.Amount;
                invoiceToUpdate.InvoiceNumber = invoiceHeader.InvoiceNumber;
                invoiceToUpdate.IsPaid = invoiceHeader.IsPaid;
                invoiceToUpdate.VatAmount = invoiceHeader.VatAmount;
                invoiceToUpdate.VatNumber = invoiceHeader.VatNumber;
                invoiceToUpdate.TotalAmount = invoiceHeader.TotalAmount;
                invoiceLineToAdd.InvoiceHeader = invoiceToUpdate;
                invoiceToUpdate.InvoiceLines.Add(invoiceLineToAdd);

                await _invoiceHeaderRepository.UpdateInvoiceHeader(invoiceToUpdate);

                return invoiceHeader;
            }
        }
        else if (invoiceLine.BrokenRules.Count != 0)
        {
            foreach (var item in invoiceLine.BrokenRules)
            {
                FrameworkException exception = new FrameworkException(invoiceLine, "UC_301_002_AddInvoiceLineToHeader", "", FrameworkExceptionType.BusinessRuleViolation);

                SaveInvoiceException(exception, authenticatedUser);
            }
        }
        return null;
    }
    catch (Exception ex)
    {
        FrameworkException exception = new FrameworkException("UC_301_002_AddInvoiceLineToHeader", ex.Message, ex, FrameworkExceptionType.Error); //Todo
        SaveInvoiceException(exception, authenticatedUser);
        throw exception;
    }
}

public BO_InvoiceHeader UC_301_003_GetInvoiceById(Guid id, BO_AuthenticatedUser authenticatedUser)
{
    try
    {
        return _mapper.Map<BO_InvoiceHeader>(_invoiceHeaderRepository.GetInvoiceHeaderById(id));
    }
    catch (Exception ex)
    {
        FrameworkException exception = new FrameworkException("UC_301_003_GetInvoiceById", ex.Message, ex, FrameworkExceptionType.Error); //Todo
        SaveInvoiceException(exception, authenticatedUser);
        throw exception;
    }
}

6 references
public void SaveInvoiceException(FrameworkException exception, BO_AuthenticatedUser authenticatedUser)
{
    var dataobject = _mapper.Map<DO_InvoiceException>(exception);
    _invoiceExceptionRepository.CreateInvoiceException(dataobject, _mapper.Map<DO_AuthenticatedUser>(authenticatedUser));
}

```

Voor de Exceptions:

```

namespace InvoiceDataLayer.Repositories
{
    4 references
    public interface IInvoiceExceptionRepository
    {
        1 reference
        Task CreateInvoiceException(DO_InvoiceException entity, DO_AuthenticatedUser authenticatedUser);
        1 reference
        IEnumerable<DO_InvoiceException> GetExceptions();
    }
}

```

```

namespace InvoiceDataLayer.Repositories
{
    2 references
    public class InvoiceExceptionRepository : IInvoiceExceptionRepository
    {
        private InvoiceContext _invoiceContext;
        2 references | 2/2 passing
        public InvoiceExceptionRepository(InvoiceContext invoiceContext)
        {
            _invoiceContext = invoiceContext;
        }

        1 reference
        public IEnumerable<DO_InvoiceException> GetExceptions()
        {
            return _invoiceContext.InvoiceException.ToList();
        }

        2 references
        public async Task CreateInvoiceException(DO_InvoiceException entity,
            DO_AuthenticatedUser authenticatedUser)
        {
            entity.CreatedBy = authenticatedUser.userName;
            _invoiceContext.InvoiceException.Add(entity);

            await _invoiceContext.SaveChangesAsync();
        }
    }
}

```

Nu passen we onze methods op de InvoiceController aan. Bij elke request ondervragen we de user:

```

[Authorize]
[HttpPost("CreateInvoiceHeader")]
0 references
public async Task<IActionResult> CreateInvoiceHeader(CreateInvoiceHeaderInput createInvoiceHeaderInput)
{
    if (!ModelState.IsValid)
    {
        return StatusCode(400);
    }
    var result = await _invoiceUseCases.UC_301_001_CreateInvoiceHeader(createInvoiceHeaderInput.VATNumber,
        _mapper.Map<BO_AuthenticatedUser>(GetCurrentUser()));
    return Ok(result);
}

[Authorize] // [AllowAnonymous]
[HttpPost("AddInvoiceLineToInvoiceHeader")]
0 references
public async Task<IActionResult> AddInvoiceLineToInvoiceHeader(AddInvoiceLineToInvoiceHeaderInput createInvoiceHeaderInput)
{
    if (!ModelState.IsValid)
    {
        return StatusCode(400);
    }
    var result = await _invoiceUseCases.UC_301_002_AddInvoiceLineToHeader(createInvoiceHeaderInput.InvoiceHeaderID, _mapper.Map<BO_InvoiceLine>(createInvoiceHeaderInput),
        _mapper.Map<BO_AuthenticatedUser>(GetCurrentUser()));
    return Ok();
}

[Authorize] // [AllowAnonymous]
[HttpPost("GetInvoiceByName")]
0 references
public IActionResult GetInvoiceByName(GetInvoiceByNameInput getInvoiceByNameInput)
{
    if (!ModelState.IsValid)
    {
        var result = _invoiceUseCases.UC_301_003_GetInvoiceById(getInvoiceByNameInput.InvoiceHeaderId,
            _mapper.Map<BO_AuthenticatedUser>(GetCurrentUser()));
        return Ok(result);
    }
    return Ok();
}

```

Om de authenticated user ook in de testen toe te voegen doe je het volgende:

```

namespace Customers.Test
{
    5 references
    public class TestBase : TestWithSqlite
    {
        public IMapper mapper;
        protected CON_AuthenticatedUser userCon;
        protected BO_AuthenticatedUser userBo;
        protected DO_AuthenticatedUser userDo;

        [SetUp]
        0 references
        public void Setup()
        {
            customerContext.Database.EnsureCreated();
            _mapper = new MapperConfiguration(c => c.AddProfile<AutomapperConfig>()).CreateMapper();

            userCon = new CON_AuthenticatedUser() { Id = new Guid(), userName = "Queaso" };
            userBo = _mapper.Map<BO_AuthenticatedUser>(userCon);
            userDo = _mapper.Map<DO_AuthenticatedUser>(userBo);
        }

        [TearDown]
        0 references
        public void TearDown()
        {
            customerContext.Database.EnsureDeleted();
        }
    }
}

```

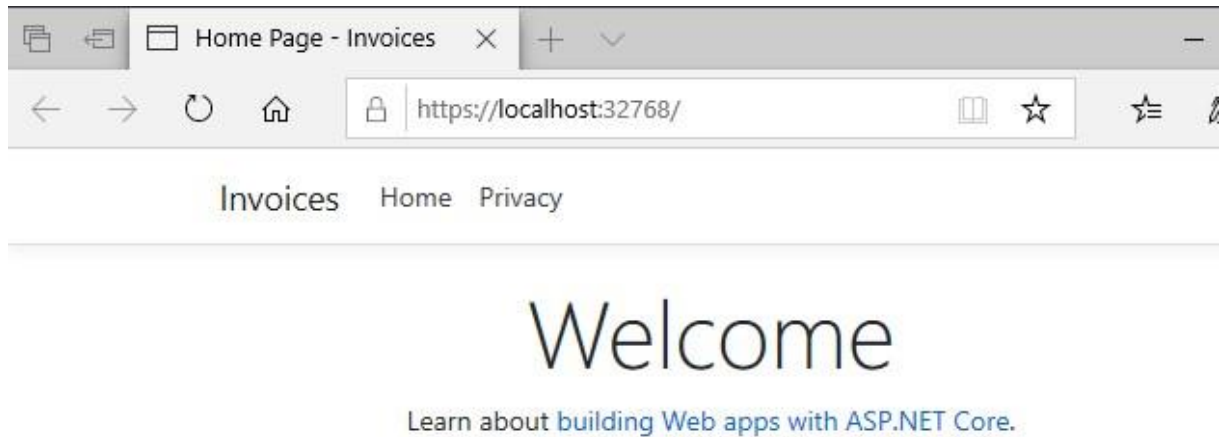
Nu kan je waar nodig de juiste authenticated user toevoegen. De userDo in CustomerExceptionRepositoryTest en de userBo in de verschillende UseCase tests.

UnitTesting

Vanaf hier wordt het te ingewikkeld om unit tests te schrijven voor de solution. Daarom gaan we deze testen doen met Postman.

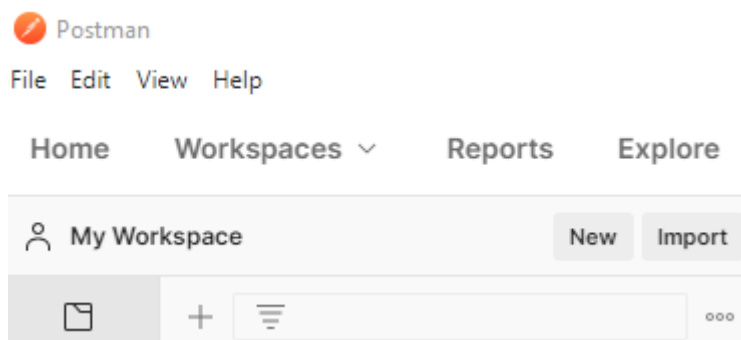
Testen van de APIs

Als we het Invoices project starten, zien we in de webbalk de HTTPS port.



Onze HTTPS port 32768 wordt dus vertaald naar de standard SSL port 443

We starten Postman op en maken om te starten een nieuwe collection aan waar we al onze testen in kunnen onderbrengen:



In je nieuwe collection klik je op ...

! Looks like your collaboration needs are growing beyond what a free team can do.

Q Filter

History

Collections

APIs

+ New Collection

Calendly
1 request

Chatbizz Portal
19 requests

ChatBizz.Api
27 requests

ChatbizzProject
39 requests

Connect Sample Collection - v1.0
46 requests

CreditSaveRequests
1 request

InfoToKeep
1 request

InvoiceDemo ☆
1 request

Share Collection

Manage Roles

Rename Ctrl+E

Edit

Create a fork

Create Pull Request

Merge changes

GET Add Request

Add Folder

Duplicate Ctrl+D

Export

Monitor Collection

Mock Collection

Publish Docs

Remove from workspace

Delete Del

En kiest Add request en geef de request de naam Register.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
[Learn more about creating collections](#)

Request name

Request description (Optional)

Descriptions support [Markdown](#)

Select a collection or folder to save to:

◀ InvoiceDemo

+ Create Folder

Cancel

Save to InvoiceDemo

Save...

Verander het request type van Get naar Post. De url kennen we nu en we voegen er de route naar de api aan toe

<https://localhost:32768>

+

```
[Route("api/[controller]")]  
[ApiController]  
1 reference  
public class InvoiceController : Controller
```

[Route..] /api/[Class InvoiceController – Controller(Convention)=Invoice]
<https://localhost:32768/api/invoice>

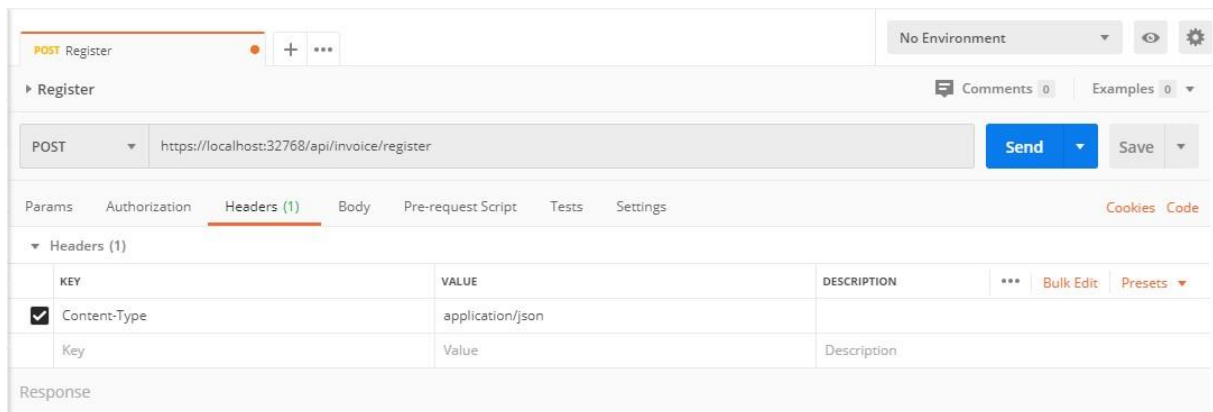
+

`[HttpPost("Register")]` / register

<https://localhost:32768/api/invoice/register>

= POST url

Onder Headers geven we enkel mee dat de content van het type json is:

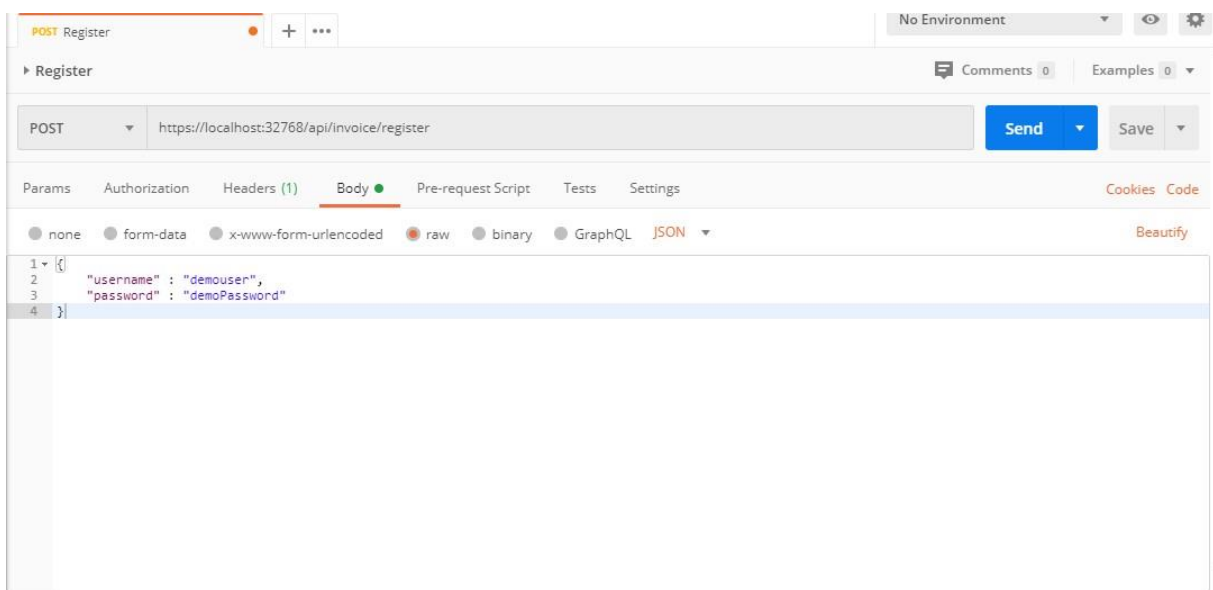


Kies de tab Body en kies type 'raw'. De data die we moeten meegeven aan de call is:

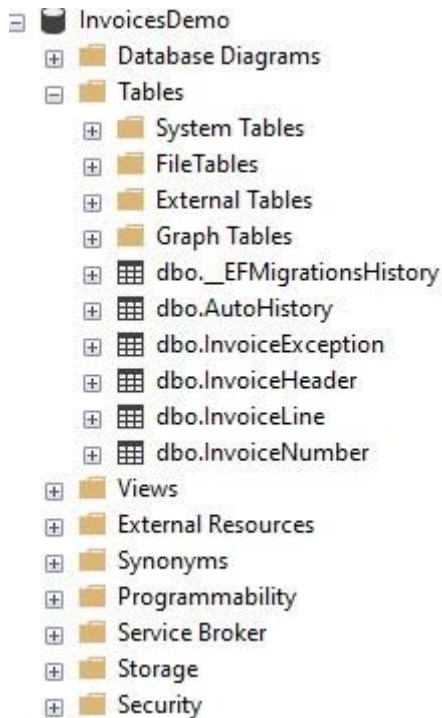
```
[HttpPost("Register")]
0 references
public async Task<IActionResult> Register(RegisterUserInput registerUserInput)

public class RegisterUserInput
{
    7 references
    public string userName { get; set; }
    4 references
    public string password { get; set; }
}
```

Dus in json:



Voor de eerste call ziet onze database er zo uit:

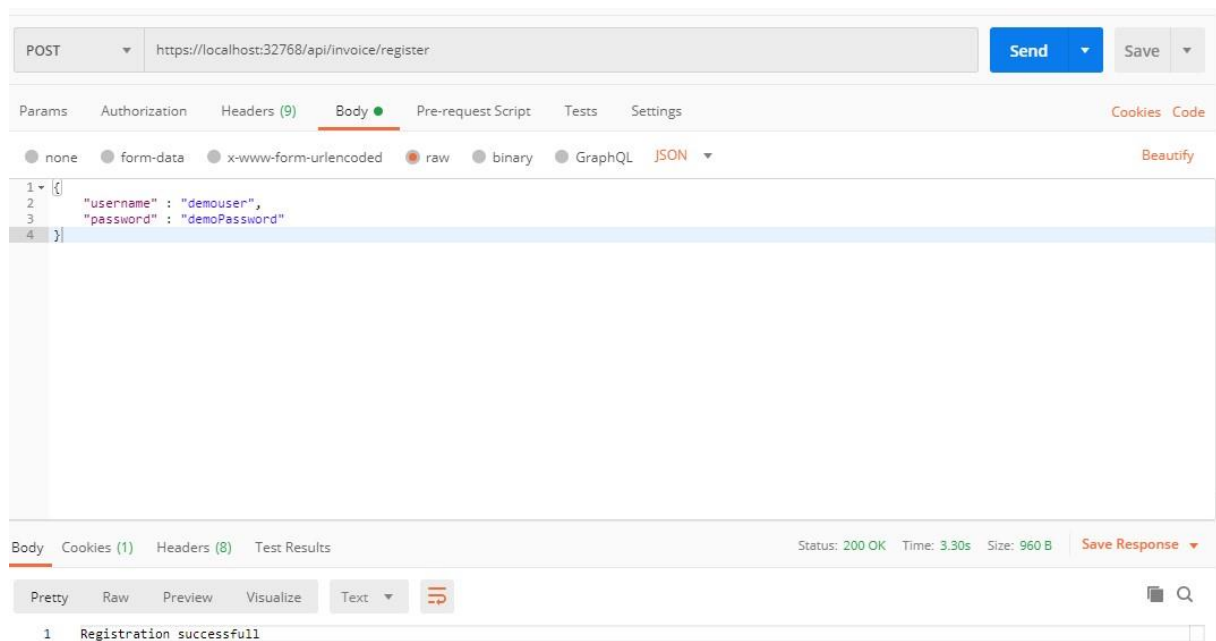


Als we nu op send klikken zullen we een fout krijgen.

```
System.AggregateException: One or more errors occurred. (Invalid object name 'AspNetUsers'.)
```

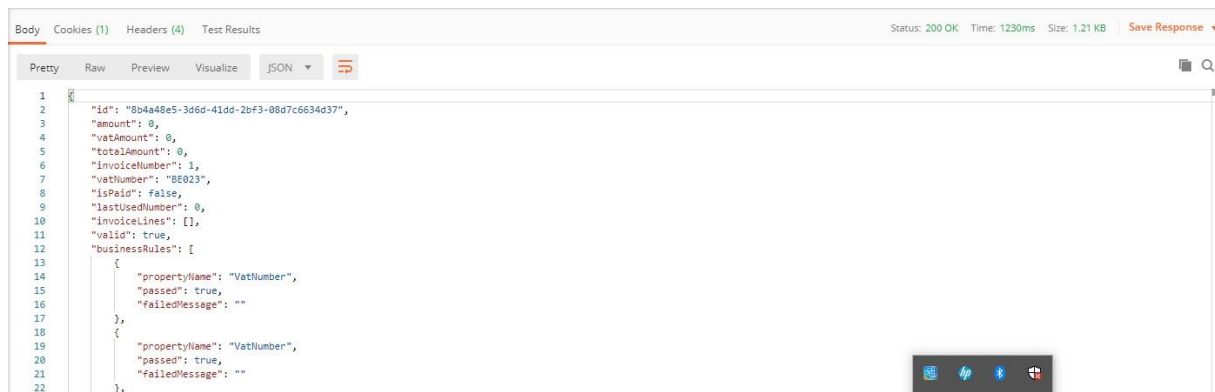
En dit klopt, we hebben onze database nog niet geüpdatet na het aanpassen van de context van DbContext naar IdentityDbContext. Het eenvoudigste is gewoon via ssms de database wegwerpen, en in de solution de map Migrations verwijderen met alle inhoud. Via de console 'add-migration init' en 'update-database'. En we zijn klaar.

Voeren we nu de send uit:

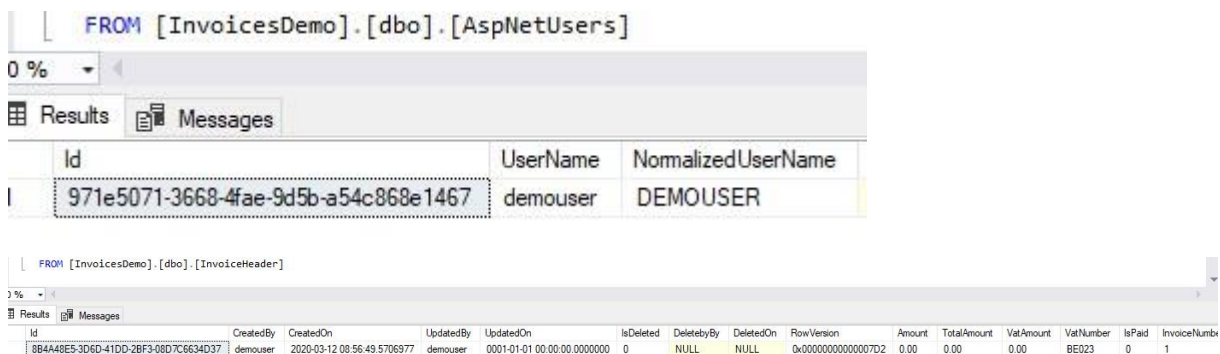


De volgende stap is login en logout testen. Het testen van de login Api is bijna identiek aan de register api

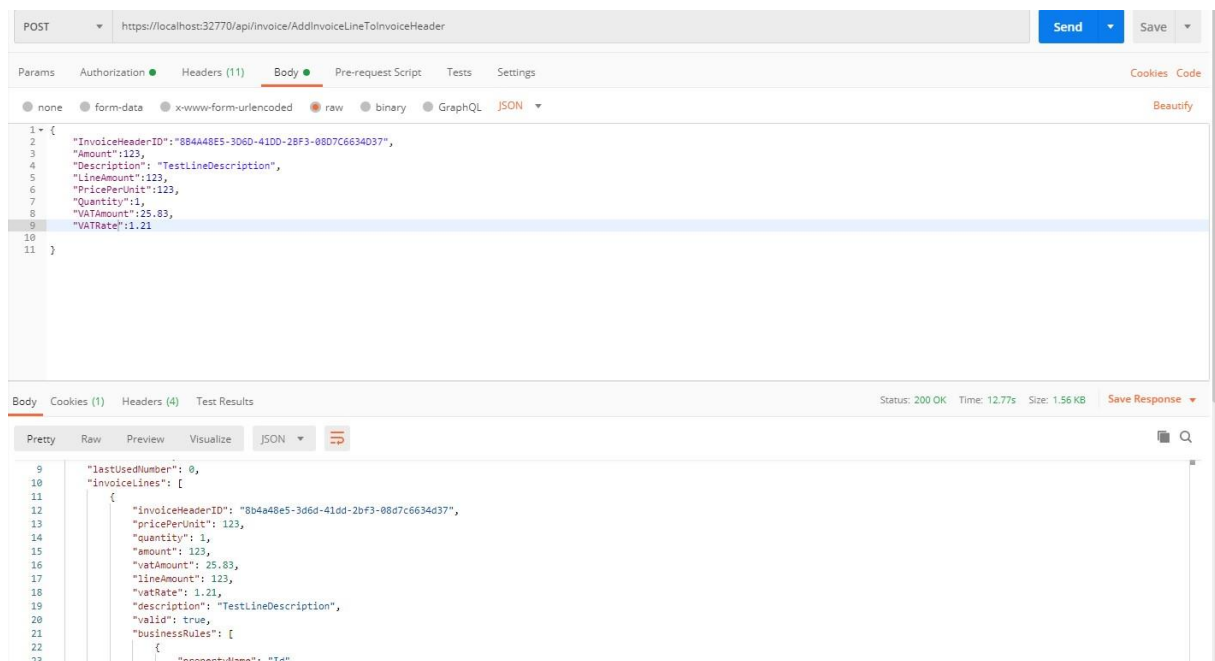
We krijgen onze header terug in json formaat:



En in de database:



Nu nog een invoicelijn toevoegen, uit de database kopiëren we daarvoor de Id van de InvoiceHeader en roepen de api op:



We krijgen onze invoice terug met de toegevoegde lijn.

De methode GetInvoiceByName geeft het zelfde resultaat:

POST `https://localhost:32770/api/invoice/GetInvoiceByName` Send Save

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "InvoiceHeaderId": "884A48E5-3D6D-41DD-2BF3-08D7C6634D37"
3 }
```

Body Cookies (1) Headers (4) Test Results Status: 200 OK Time: 151ms Size: 1.1 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "8b4a48e5-3d6d-41dd-2bf3-08d7c6634d37",
3   "amount": 123.00,
4   "vatAmount": 123.00,
5   "totalAmount": 246.00,
6   "invoiceNumber": 1,
7   "vatNumber": "BE023",
8   "isPaid": false,
9   "lastUsedNumber": 0,
10  "invoiceLines": [
11    {
12      "invoiceHeaderId": "8b4a48e5-3d6d-41dd-2bf3-08d7c6634d37",
13      "pricePerUnit": 123.00,
14      "quantity": 1.00,
15      "amount": 123.00,
```

Rest ons nog Logout te testen en te zien of we bijvoorbeeld nog een record kunnen opvragen:

logout POST `https://localhost:32770/api/invoice/logout` Send Save

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1
```

Body Cookies Headers (9) Test Results Status: 200 OK Time: 46ms Size: 530 B Save Response

Pretty Raw Preview Visualize Text

```
1
```

Logout Ok resultaat bij terug oproepen zelfde InvoiceHeader:

GetInvoiceByName POST `https://localhost:32770/api/invoice/GetInvoiceByName` Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "InvoiceHeaderId": "884A48E5-3D6D-41DD-2BF3-08D7C6634D37"
3 }
```

Body Cookies Headers (3) Test Results Status: 404 Not Found Time: 37ms Size: 99 B Save Response

Pretty Raw Preview Visualize Text

```
1
```

Resultaat...Not Found.