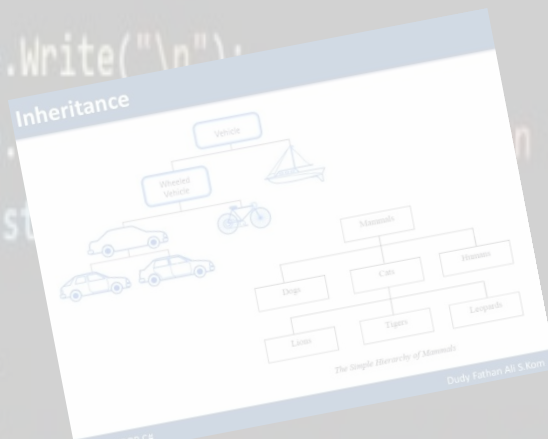
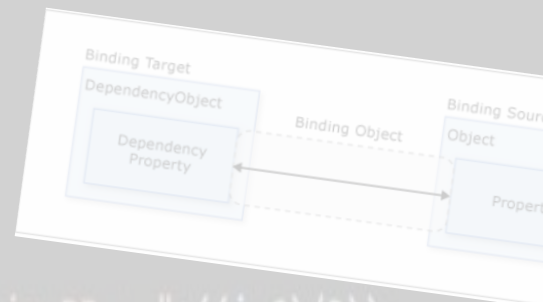


Desktop Development

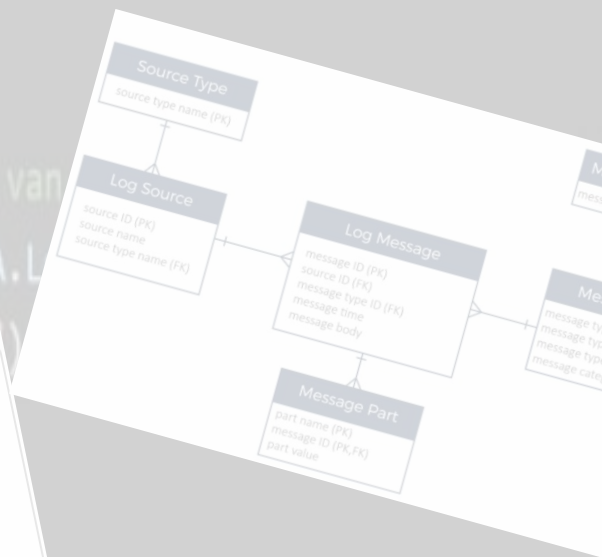
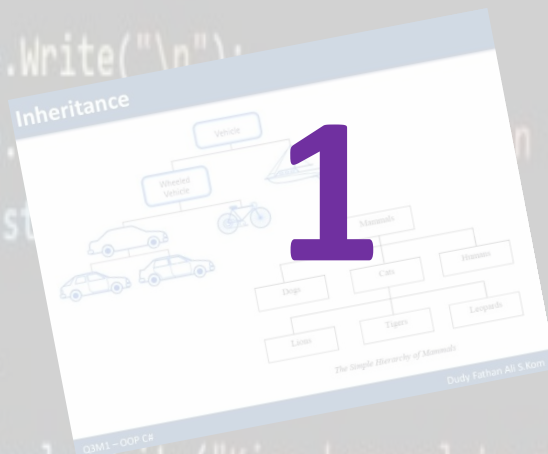


Inhoudsopgave

Desktop Development 1.....	5
Inleiding.....	7
Algemeen.....	7
Opbouw	7
Herhaling	8
Inleiding	8
Opdracht 01: Database programma: Cars.....	8
Kwaliteitseisen.....	9
Opdracht 02: Verbeter het programma	11
Samenvatting.....	11
Fastfoodrestaurant Los Pollos Hermanos	12
Inleiding	12
De product backlog	12
Definition of done	12
Opdracht 3: Inrichten ontwikkelomgeving.....	13
Opdracht 4: Databaseclass en ConnectionString	14
Opdracht 5: Userstory RestaurantDisplay.....	15
Interfaces.....	23
Binding.....	27
Opdracht 6: Userstory IngrediëntenInzicht.....	33
Normaliseren Ingrediënten: 1 op meer relaties.....	34
Opdracht 7: Userstory IngrediëntEenheden	36
Opdracht 8: UserStory Beheer Ingrediënten.....	39
Normaliseren DisplayWindow	45
Opdracht 9: Userstory Automatisch Display	47
Opdracht 10: Userstory Beheer Maaltijden	48
Use cases	49
Opdracht 11: Login	52
Conclusie	53
Optimalisatie (EXTRA STOF)	54
Inleiding	54
Generics en Delegates.....	54
SQL Select GetAll	55
SQL Select GetById	56
SQL CreateUpdateOrDelete	57

Eindopdracht Desktop Development 1	58
Opdracht.....	58
Aanpak.....	58
Alternatief.....	58
Beoordelingsmatrix	59
Bijlagen van Los Pollos Hermanos	60
Overzicht maaltijden (peildatum: 1-2-2022)	60
Overzicht dranken	60
Overzicht menu's.....	60
Overzicht Ingrediënten (peildatum: 1-2-2022)	60
Bijlagen bij opdrachten.....	61
Bijlagen PhpMyAdmin	62

Desktop Development



Inleiding

Algemeen

In het eerste jaar van de opleiding heb je geleerd om desktop applicaties te maken. Hiervoor maakte je ook gebruik van het *WPF framework* met de programmeertalen *C#* (voor de code) en *XAML* (voor de GUI). Ook heb je een databaseprogramma gemaakt. Als database maakte je gebruik van *MySQL*.

In deze module ga je hiermee verder. Je gaat werken met *WPF .Net Core*, de opvolger van het *WPF framework*. De verschillen zijn minimaal, dus een goed moment om nu over te stappen.

Als database maak je weer gebruik van *MySQL*. Daarnaast ga je ook de database *SqlServer* gebruiken. Het zijn allebei relationele databases: de gegevens staan in tabellen met daartussen relaties. Je zult merken dat er kleine verschillen zitten in het werken met deze databases. Ze worden allebei veel gebruikt in het bedrijfsleven, dus het is belangrijk dat je er mee kunt werken.

Qua techniek maak je kennis met *binding*. Met *binding* zorg je ervoor dat wanneer gegevens in jouw programma wijzigen, deze direct worden bijgewerkt op de schermen van jouw applicatie. Statements als *tbName.Text = _firstName + _lastName*, gebruik je niet meer.

Ook ga je meer *object geörienteerd* werken. Naast classen voor jouw Windows- en Databaseclass, ga je nu ook voor alle tabellen uit jouw database een class gebruiken. Je leert hoe je gegevens kunt beschermen met *encapsulation* en hoe je gegevens en gedrag van classen kunt hergebruiken (*inheritance*). Ook leer je hoe je interfaces gebruikt.

Bedrijven maken bij elke applicatie documentatie. Deze is bestemd voor opdrachtgevers en gebruikers (de functionele documentatie) maar ook voor jouzelf en je collega-programmeurs (de technische documentatie). In deze module leer je een *use case diagram* (onderdeel van de functionele documentatie), een *ERD* (Entity Relationship Diagram) en een genormaliseerd *datamodel* (onderdeel van de technische documentatie) te maken.

Opbouw

Je begint deze module met een herhalingsopdracht. Je maakt een databaseprogramma zoals je dat ook vorig jaar hebt gemaakt. Daarna worden de belangrijkste kwaliteitseisen behandeld, waaraan programma's vanaf dit jaar moeten voldoen. Je maakt een opdracht waarin je van een bestaand programma controleert, of het aan deze eisen voldoet. Kwaliteitseisen zijn erg belangrijk, omdat ze de leesbaarheid en onderhoudbaarheid van jouw programma verhogen. Je moet ze bovendien op je examens kunnen toepassen.

Daarna ga je de nieuwe stof van deze module leren met een opdracht die je uitvoert voor het restaurant *Los Pollos Hermanos*. Ben je hiermee klaar, dan ga je aan de slag met een applicatie die jijzelf bedenkt en waarin je laat zien dat jij het geleerde zelfstandig kunt toepassen.

Herhaling

Inleiding

Je begint deze periode met een herhalingsoefening om je kennis van vorig jaar op te frissen. Als eerste opdracht maak je een *databaseprogramma Cars*, waarbij je als database *MySQL* gebruikt.

Vanaf nu maak je gebruik van het *.Net Core framework*. Met *.Net Core* kun je niet alleen voor Windows applicaties maken, maar ook voor Linux en MacOS.

Nadat je jouw databaseprogramma hebt gemaakt, lees je meer over een aantal kwaliteitseisen die je vanaf nu moet toepassen. Door deze eisen toe te passen maak je jouw programma beter leesbaar en beter onderhoudbaar. Ook moet je deze eisen op je examen toepassen. Het is daarom belangrijk deze vanaf nu consequent toe te passen.

Heb je dit allemaal gedaan, dan ben je klaar om vanaf het volgende hoofdstuk aan de slag te gaan met de opdracht voor het bedrijf *Los Pollos Hermanos*.

[.NET Core: Wat zijn de voor- en nadelen? - Growteg](#)

[What is .NET? An open-source developer platform. \(microsoft.com\)](#)

Opdracht 01: Database programma: Cars

Maak een *WPF Core* CRUD-applicatie. Maak gebruik van een aparte databaseclass.

Voor het programma *mag* je gebruik maken van de startcode die je op ItsLearning aantreft. Je hebt dan onderstaande voorbeeld lay-out en ook code om een afbeelding van je computer in te lezen.

Om de database te maken, mag je gebruik maken van het script dat je bij de startcode aantreft. De database bevat 1 tabel, *cars*. Deze tabel heeft 4 kolommen: *carId*, *make*, *picture*, *yearOfIntroduction*.

carsdb cars	
carId	: int(11)
make	: varchar(255)
picture	: blob
yearOfIntroduction	: int(11)

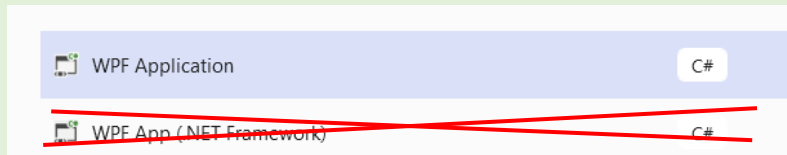
Voorbeeld lay-out:

The screenshot shows a WPF application window titled 'MainWindow'. It contains a list of cars: Alfa Romeo, Tesla, Dodge (highlighted), and Bentley. To the right of the list, there's a 'SelectedAuto' section with a 'Merk' field (Dodge), an 'Afbeelding' field (a green sports car), and a 'Jaar' field. Below this is a 'NewAuto' section with 'Merk', 'Afbeelding', and 'Jaar' fields, and a 'Create' button. There are also 'Update' and 'Delete' buttons near the 'SelectedAuto' section.

- Links: Lijst met auto's, click je op een auto, dan wordt het merk en de afbeelding in het gedeelte rechtsboven getoond
- Rechtsboven: geselecteerde auto. Je kunt hier een ander merk invullen en een andere afbeelding kiezen. Met Update sla je de gegevens op in de database. Met Delete verwijder je de auto uit de database. In beide gevallen refresh je het overzicht in het linker deel.
- Rechtsonder: hier vul je een nieuwe auto in. Met Create wordt de nieuwe auto in de database opgeslagen en refresh je het linker deel.

Let op: In het eerste jaar heb je met WPF gewerkt, maar wellicht niet met Core. Vanaf nu werken we alleen nog maar met WPF Core.

Zorg er dus voor dat je nu een WPF Core template kiest: WPF Applicatie
En niet de ouderwetse versie: WPF App (.Net Framework)



Kwaliteitseisen

Vanaf dit jaar ga je ervoor zorgen dat jouw programma's aan een aantal kwaliteitseisen voldoen. Hoe beter de kwaliteit, hoe beter het programma leesbaar wordt en hoe eenvoudiger het wordt om eventuele fouten op te lossen. De kwaliteitseisen die jij moet toepassen staan hierna opgesomd.

Gebruik zelf verklarende namen

- Geef variabelen een naam waaruit duidelijk is wat voor gegevens er in staan.
- Geef methods een naam die aangeeft wat het doel van de method is.
- Geef een class de naam die aangeeft waarvoor de class wordt gebruikt.

Voorbeelden

- `CreatePersonWindow1` : Dit is een Window class waarmee een persoon wordt gemaakt.
- `CreatePerson()` : Dit is een method die een persoon aan de database toevoegt.
- `Person person` : Dit is een variabele die de gegevens van een persoon bevat.
- `List<Person> people` : Dit is een variabele die een lijst met meer personen bevat.

¹ Het is gebruikelijk om een Window class een naam te geven die eindigt op Window. We gaan dat vanaf nu ook zo toepassen.

Houd je aan de Naming Conventions

- Namen van variabelen (locals, fields en parameters) beginnen met een kleine letter
- Namen van methods en classes beginnen met een hoofdletter
- Namen van interfaces beginnen met een hoofdletter. Bovendien zet je er nog de hoofdletter I voor. Voorbeeld `INotifyPropertyChanged`

Je gaat later in deze module gebruik maken van de *interface* `INotifyPropertyChanged`. Deze *interface* speelt een belangrijke rol bij *binding*.

Zie ook: [C# Naming Conventions Cheat Sheet by GregFinzer - Download free from Cheatography - Cheatography.com: Cheat Sheets For Every Occasion](#)

Gebruik commentaar

Zorg dat je bij *elke* class en bij *elke* method aangeeft wat het doel van de class of method is. Je gaat dus niet de hele uitwerking uitleggen, dat blijkt wel uit de code.

Bij een method geef je ook kort aan wat de betekenis is van de parameters. Als de method een return-type heeft (dus iets anders dan void), dan geef je aan wat het resultaat van de method is.

```
// Method GetCars leest alle autos uit de databasetabel cars, en zet deze in een DataTable.
// De waarde van GetCars is:
// - null: Er is een fout opgetreden bij het lezen van de gegevens.
// - !null: De DataTable, gevuld met gegevens over de autos.
public DataTable GetCars()
{
    ...
}
```

Commentaar bij code geef je alleen als de code erg complex is geworden en een programmeur deze dus niet zondermeer kan lezen. Gebruik dit echter niet als excuus om complexe code te maken! Code houd je altijd zo eenvoudig mogelijk.

Lay-out van de code

Je zorgt voor een nette lay-out:

- Geen overbodige lege regels.
- Code staat netjes onder elkaar en begint op dezelfde positie.
- Als je code laat inspringen, zorg je dat dit overal op dezelfde manier gebeurt: 1 <tab>.
- Accolades { }, zet je onder elkaar. Alle code tussen { } springt 1 <tab> in.

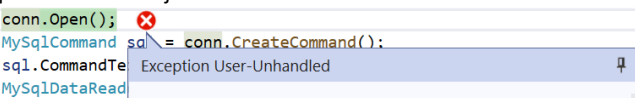
Geef voor alle mogelijke fouten altijd duidelijke instructies aan jouw gebruiker

Zorg ervoor dat je alle fouten die zich in jouw programma kunnen voordoen afhandelt. Hiermee bedoelen we:

- Zorg ervoor dat de gebruiker van jouw applicatie geen exceptions te zien krijgt, maar zorg ervoor dat hij in plaats daarvan een duidelijke boodschap krijgt wat er is gebeurd.
- Zorg bij alle fouten (exceptions, maar ook bij verkeerde invoer door de gebruiker, etc.), dat je duidelijke instructies geeft, wat de gebruiker moet doen om de fout te verhelpen.

Voorbeeld 1:

Goed Er is een fout opgetreden bij het inlezen van de persoonsgegevens. Waarschuw de service desk wanneer dit probleem zich blijft voortdoen.

Fout 

Dit geeft aan wat er is gebeurd en is een duidelijke instructie voor de gebruiker

Melding en instructie voor gebruiker ontbreekt.

Tijdens je examen mag jouw programma nooit crashen

Voorbeeld 2:

Goed Vul de straatnaam in.

Fout Straatnaam is niet ingevuld.

De gebruiker weet wat hij moet doen.

Gebruiker heeft zelf straatnaam niet ingevuld en denkt dat hem dit nu wordt bevestigd. De melding maakt niet duidelijk dat de straatnaam ingevuld had moeten worden.

Voorbeeld 3:

Goed De functie die u wilt uitvoeren is i.v.m. uitgelopen onderhoudswerkzaamheden tot 8:15u niet beschikbaar.

Fout Oeps...

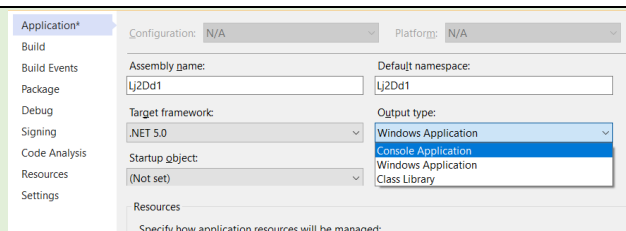
Gebruiker weet wanneer hij het opnieuw kan proberen

Hier neem je de gebruiker niet serieus.

Zorg er bij een foutsituatie voor, dat jijzelf (of de service desk) altijd de technische foutmelding te zien kan krijgen.

Bijvoorbeeld door deze naar het Console te schrijven of naar een fout-log.

Schrijf je naar het console: let er dan op dat WPF Core standaard geen console opent. Dat betekent dat wanneer je een bericht naar het console schrijft, je dit bericht niet te zien krijgt. Je kunt dit verhelpen door in je project properties aan te geven dat jij een console-applicatie hebt gemaakt. Nadeel hiervan is wel dat bij het starten van je applicatie steeds een zwart console zichtbaar is.



Geef alle gegevens het juiste gegevenstypen

Gebruik voor een variabele steeds een gegevenstype dat past bij de waarde die je in de variabele wilt opslaan. Dus voor een naam gebruik je een *string*, voor een leeftijd een *int* en voor een releasedatum een *DateTime*, voor een bedrag een *decimal*, etc..

Haal je gegevens op uit een database, zorg dan dat je ze in een variabele zet waarvan het c#-type past bij het type van de database (een auto-increment zet je dus niet meer in een string).

Als je niet weet welk gegevenstype je voor een database kolom moet gebruiken, kun je daar tijdens het debuggen, via een watch achter komen. Bijvoorbeeld:

- De kolom *quantity* is het c# type *uint* (unsigned int).
- De kolom *price* is van het c# type *decimal*.

Watch 1

Search (Ctrl+E) 🔍 Search Depth: 3

Name	Value	Type
reader["quantity"]	2	object (uint)
reader["price"]	2.23	object (decimal)

Add item to watch

Juiste code op de juiste plaats

Met de juiste code op de juiste plaats wordt een programma voorspelbaar. Je hebt al geleerd dat je bijvoorbeeld alle code die met een database werkt, in een database class staat.

Zo hoor je alle code die gebruikt wordt om met een gebruiker te "communiceren" in een Window class te staan. Dit betekent bijvoorbeeld dat je nooit een `MessageBox.Show` mag gebruiken in een Databaseclass.

Om deze reden maken we in een databaseclass geen gebruik van types als `ObservableCollection` of `DataView`. Deze types zijn bedoeld om gegevens in een Window te tonen en kunnen dus wel in een Window-class gebruikt worden.

Conclusie

Je hebt nu kennis gemaakt met een aantal kwaliteitseisen die je moet toepassen. Tijdens Desktop Development 1 en Desktop Development 2 houd je je aan de hier genoemde kwaliteitseisen.

Bij elk bedrijf worden kwaliteitseisen toegepast. Deze kunnen afwijken van de eisen die hier staan. Ga je op stage: vraag dan altijd naar de kwaliteitseisen/programmeerstandaards die binnen het bedrijf worden gebruikt.

Opdracht 02: Verbeter het programma

Bestudeer de startcode van het programma dat je op ItsLearning aantreft. Verbeter het programma zodat het aan alle kwaliteitseisen voldoet.

Samenvatting

Je hebt de stof uit het eerste leerjaar herhaald en weet hoe je een databaseprogramma moet maken. Je kent de kwaliteitseisen waar jouw programma binnen Desktop Development aan moet voldoen. Je hebt je kennis nu voldoende opgefrist om aan de applicatie voor de eigenaar van *Los Pollos Hermanos* te beginnen.

Fastfoodrestaurant Los Pollos Hermanos

Inleiding

Het fastfoodrestaurant Los Pollos Hermanos is zojuist van eigenaar verwisseld. Van de vorige eigenaar is nooit meer iets vernomen.

Los Pollos Hermanos is vooral bekend om zijn vriendelijke bediening en lekker eten. Klanten beoordelen het restaurant overwegend met 5 sterren (95%). Minder dan 0,01% geeft een waardering van minder dan 3 sterren.



Op het display achter de bediening zien de klanten wat ze allemaal kunnen bestellen: maaltijden, dranken en complete menu's. Meer details hierover vind je in de bijlage.

Jij krijgt van jouw opdrachtgever de taak om een applicatie voor het restaurant te maken. De eigenaar neemt de rol van *product owner* voor zich en heeft al een aantal user story's in de *product backlog* gezet. Het *development team* waartoe jij behoort heeft al een *definition of done* (DoD) opgesteld. De *product backlog* en de *DoD* tref je hiera aan.

De *product owner* is de enige persoon die *user story's* in de *product backlog* zet. De *user story* met de meeste *business value* staat bovenaan. Bij de sprint planning zal het *development team* de *user story's* op volgorde van de *product backlog* gaan maken.

De product backlog

De *product backlog* kent op dit moment 5 items:

1. Als gastheer/vrouw (bediening) wil ik de maaltijden op het Display in het restaurant kunnen tonen, zodat de klanten hieruit een keuze kunnen maken.
2. Als manager wil ik dat het display in het restaurant bij de maaltijden, de omschrijving en prijs afleiden uit de ingrediënten waaruit de maaltijden zijn samengesteld, zodat klanten altijd de actuele samenstelling en prijs zien.
3. Als manager wil ik de maaltijden kunnen beheren CRUD)
4. Als manager wil ik de ingrediënten van het restaurant kunnen beheren, zodat de mogelijkheden om een maaltijd te maken, inzichtelijk zijn.
5. Als manager wil ik gebruikers toegang tot applicatiefuncties kunnen geven, zodat ze alleen die functies kunnen uitvoeren waarvoor ze zijn geautoriseerd.

Definition of done

Het *development team* waarvan jij deel uitmaakt, beschouwt een item uit de *sprint backlog* gereed als het voldoet aan de *definition of done*:

- Alle documentatie is bijgewerkt. De documentatie omvat het *use case diagram* voor het *functioneel ontwerp* en het *datamodel* (erd diagram en beschrijving) voor het *technisch ontwerp*.
- Er is een database script om de database volgens het datamodel te maken of te wijzigen.
- Er is een database script om de database te vullen met testdata.
- Gegevens worden uitsluitend via een database class van en naar de database geschreven.
- Voor elke database tabel uit het datamodel is er 1 model class gemaakt en zijn er standaard CRUD-methods in de database class: *GetAll*, *GetById*, *Create*, *Update*, *Delete*.
- De opgeleverde code voldoet aan de kwaliteitseisen.
- Er heeft een code review plaatsgevonden door een medeteamlid (medestudent).

Definition Of Done zoals opgesteld door het Development Team

Opdracht 3: Inrichten ontwikkelomgeving

Tijdens de eerste sprint planning, heeft het ontwikkelteam vastgesteld wat zij nodig hebben om de applicatie te kunnen gaan maken. Ze gaan werken met een MySQL database; hierbij maken ze gebruik van XAMP of WAMPP. De applicatie wordt ontwikkeld met Visual Studio 2022. De applicatie wordt geschreven in c#, waarbij het Wpf Core framework wordt gebruikt. Voor de documentatie wordt gebruik gemaakt van Visio (schema's) en MsWord.

Het *development team* gaat aan het begin van een nieuwe sprint, tijdens de *sprint planning*, na wat er allemaal moet gebeuren om een *user story* te realiseren. Door middel van *poker planning* wordt een inschatting van de inspanning gemaakt. Als een *user story* niet binnen 1 sprint gemaakt kan worden, wordt deze gesplitst in 2 of meer delen: elke deel levert nog steeds *business value* op.

De *user story's* die opgepakt worden, worden op de *sprint backlog* gezet; zoveel als dat er gemaakt kunnen worden binnen 1 sprint. Ook eventuele extra activiteiten die voor de *user story's* nodig zijn worden in de sprint planning opgenomen.

Alle stakeholders weten daardoor waarmee het development team bezig is. Dat is transparant.

Opdracht:

- Zorg ervoor dat jij jouw ontwikkelomgeving op orde hebt (Visio, Visual Studio 2022, MsWord en Xamp of Wampp)
- Maak een database aan met de naam LosPollosHermanos
- Maak een WPF Core programma en geef dit de naam *Lj2Dd1En2*.
- Maak een folder voor je documentatie en zet daar de visio en MsWord templates in die je in de bijlage kunt vinden
- Maak een folder voor je database Scripts aan en zet daar het script in om jouw database te maken

Tip: Zet alle code en documentatie in een GIT repository.

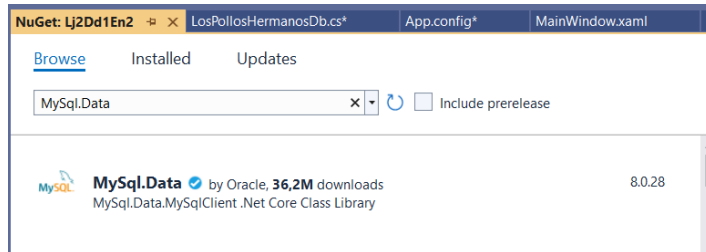
The screenshot shows a Jira board with the following content:

- Product Backlog (4 items):**
 - Als manager wil ik dat het display in het restaurant bij de maaltijden, de omschrijving en prijs afleiden uit de ingrediënten waaruit de maaltijden zijn samengesteld, zodat klanten altijd de actuele samenstelling en prijs zien. (Added by buseSummaCollege)
 - Als manager wil ik de maaltijden kunnen beheren (CRUD). (Added by buseSummaCollege)
 - Als manager wil het de ingrediënten van het restaurant kunnen beheren, zodat de mogelijkheden om een maaltijd te maken, inzichtelijk zijn. (Added by buseSummaCollege)
 - Als manager wil ik gebruikers toegang tot applicatiefuncties kunnen geven, zodat ze alleen die functies kunnen uitvoeren waarvoor ze zijn geautoriseerd. (Added by buseSummaCollege)
- Sprint Backlog (2 items):**
 - Maak Databaseclass
 - ☐ Installeer NuGet MySQL.Data van Oracle
 - ☐ Zet connectionstring in configuratiebestand (App.config)
 - ☐ Maak folder Models
 - ☐ Plaats in Models de database class, en neem hierin een private field op voor de connection string(Added by buseSummaCollege)
 - Als gastheer/vrouw wil ik de maaltijden op het Display in het restaurant kunnen tonen, zodat de klanten hieruit een keuze kunnen maken.
 - ☐ Maak documentatie (use case)
 - ☐ Maak documentatie (datamodel en ERD)
 - ☐ Pas database aan op basis van datamodel en ERD
 - ☐ Codeer Domain Model Meals
 - ☐ Codeer Databaseclass method GetMeals
 - ☐ Maak in de projectfolder Views, de class DisplayWindow om de maaltijden te tonen(Added by buseSummaCollege)
- Doing (1 item):**
 - Inrichten ontwikkelomgeving
 - ☐ Installeer / Update Visio
 - ☐ Installeer / Update Visual Studio 2022
 - ☐ Installeer / Update MsWord
 - ☐ Installeer / Update Xampp of Wamp
 - ☐ Maak WPF Core C# project, naam: Lj2Dd1
 - ☐ Maak documentatiefolder
 - ☐ Maak folder voor database scripts(Added by buseSummaCollege)
- Ready for Review (0 items):**

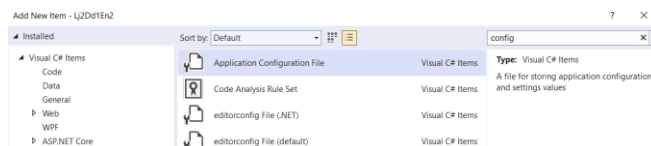
Opdracht 4: Databaseclass en ConnectionString

In deze opdracht maak je een databaseclass voor de applicatie.

- Installeer met NuGet: MySQL.Data (nodig omdat je met een MySQL database werkt):
 - Menu: Tools | NuGet Package Manager | Manage NuGet Packages for Solutions)
 - Zoek en installeer: MySQL.Data



- Voeg een App.config bestand toe:
 - Kies New Item... en zoek op *application configuration file*



- Selecteer *Application Configuration File* en voeg deze toe aan je project. De naam van het bestand laat je ongewijzigd: *App.config*
- Open de *App.config* en zet daar volgens onderstaand voorbeeld de connection string in:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add
      name="Lj2Dd1En2Conn"
      connectionString="server=localhost;database=LosPollosHermanos;uid=root;pwd="
      providerName="MySQL.Data.MySqlClient"
    />
  </connectionStrings>
</configuration>
```

1. De id die je vanuit jouw programma gebruikt voor de connectionstring
2. De connectionstring, met server, databasenaam, userid etc.
3. Dit is bibliotheek die gebruikt wordt voor jouw database. Het is dezelfde naam die jij in jouw database class gebruikt

- Gebruik het App.config bestand in jouw database class:
 - Maak een project folder en geef deze de naam Models (Models komt uit MVC)
 - Maak in deze folder een database class met de naam *LosPollosHermanosDb*.
 - Voeg de volgende code toe:

```
using MySQL.Data.MySqlClient;
using System.Configuration;

namespace Lj2Dd1En2.Models
{
    public class LosPollosHermanosDb
    {
        private string connString =
            ConfigurationManager.ConnectionStrings["Lj2Dd1En2Conn"].ConnectionString;
    }
}
```

1. Met de configurationManager lees je de connectionstring in met de id **Lj2Dd1En2Conn** uit de app.config.

In leerjaar 1 plaatste je de *connectionstring* voor de database in de code van je database class. Dit is een onwenselijk situatie omdat:

- De *connectionstring* die jij gebruikt, bedoelt is voor een ontwikkelomgeving en niet voor een productieomgeving
- De *connectionstring* vaak gevoelige informatie bevat, zoals een user id en wachtwoord. Deze informatie wil je niet met anderen delen.
- Nadat je een applicatie getest hebt en naar productie hebt overgebracht, de code ervan niet meer mag worden aangepast.

Om deze reden mag je vanaf een connection string niet meer in je code opnemen, maar moet je deze in een configuratie bestand zetten.

De productievoorbereiders kunnen dan zorgdragen voor de juiste productieinstellingen, zonder dat ze daarvoor code moeten aanpassen.

Opdracht 5: Userstory RestaurantDisplay

Als gastheer/vrouw (bediening) wil ik de maaltijden op het Display in het restaurant kunnen tonen, zodat de klanten hieruit een keuze kunnen maken.

Nadat je de ontwikkelomgeving en database class met connectionstring hebt gemaakt, ga je verder werken aan de eerste userstory van de gebruiker. Achtereenvolgens pak je de volgende onderwerpen aan:

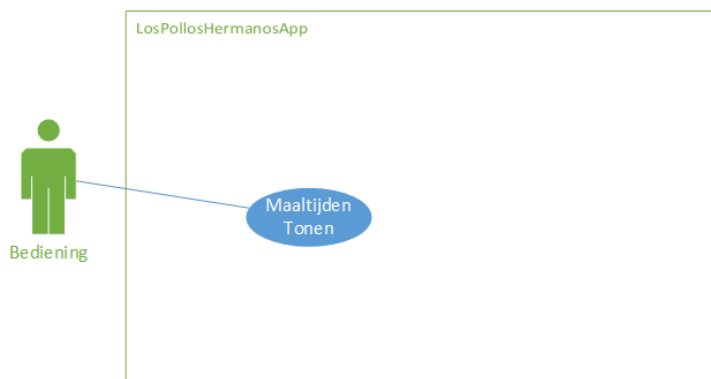
- Je werkt de documentatie bij, bestaande uit:
 - het Use Case Diagram van het functioneel ontwerp: de bediening gebruikt de use case maaltijd tonen,
 - het datamodel + ERD Diagram: de entiteit (tabel) Meals bevat de gegevens van maaltijden
- Je past de database aan volgens het datamodel: maak de tabel Meals.
- Je maakt een data clas, voor de entiteit Meals: class Meal.
- Je voegt aan de database class een method toe om de maaltijdgegevens op het display te tonen: GetMeals.

De opdrachten worden hierna verder toegelicht.

Opdracht 5.1: Werk het Use Case diagram in de documentatie bij

Op dit moment (deze user story) kent de app nog maar 1 actor: **bediening**. Deze maakt gebruik van de functie Maaltijden Tonen.

Neem het volgende **use case diagram** over met Visio:



Een Use Case Diagram is een plaatje waarin je laat zien welke functies het systeem voor welke *actors* uitvoert.

- Een *actor* wordt als een poppetje getekend. Het is de rol van een gebruiker; bijvoorbeeld bediening, manager, klant of administrator. Meestal zijn het personen, maar dat hoeft niet zo te zijn.
- Het systeem wordt als een vierkant weergegeven, met daarin de naam van het systeem.
- De use cases worden binnen het vierkant (in het systeem) met ovaal getekend.

Door doorgetrokken lijnen zie je welke actors welke use cases gebruiken.

Je weet nu hoe je een Use Case moet maken met Visio.

Opdracht 5.2: Maak het Datamodel, ERD-Diagram

Om het *datamodel* en *ERD-diagram* te maken kijk je naar de gegevens die voor de *userstory* nodig zijn.

In de *userstory* lees je dat de bediende een overzicht op een display wil kunnen tonen van alle **maaltijden** waaruit klanten een keuze kunnen maken. Het overzicht toont de volgende gegevens: de unieke **naam** van de maaltijd, een **omschrijving** waaruit blijkt wat er allemaal in de maaltijd zit en tenslotte een **prijs** die de klant moet betalen.

Voor deze gegevens maak je een tabel met de naam **Meals**. Voor elk gegeven over de maaltijd zet je een aparte kolom in deze tabel:

- **MealId**, int, auto increment: de primary key
- **Name** (varchar(50)), verplichte unieke naam
- **Description** (text), optionele omschrijving
- **Price** (decimal, met 2 decimalen), verplichte prijs

In een datamodel beschrijf je hoe je database eruit moet zien.

In een ERD (Entity Relationship Diagram) geef je het datamodel in een plaatje weer.

Het relationeel datamodel bevat meestal 1 of meer entiteiten (tabellen). Deze tabellen hebben vaak een relatie met elkaar.

In de tabellen zet je de gegevens.

Bij het maken van een datamodel zorg je ervoor dat de gegevens zo efficiënt en specifiek mogelijk worden vastgelegd: elk gegeven komt maximaal 1 keer voor en zet je steeds in een aparte kolom van een databasetabel.

Neem het volgende ERD diagram over met Visio:



Je weet nu hoe je een ERD diagram moet maken met Visio.

Maak in MsWord de volgende beschrijving van het datamodel:

Table:	Meals		
Definition:	De maaltijden die het restaurant aan haar klanten aanbiedt		
Column	Type	Required	Remarks
MealId	int	Ja	Auto increment
Name	varchar(50)	Ja	
Description	text	Nee	
Price	decimal(18,2)	Ja	
Index	Description		
IX_Name	Name moet uniek zijn		
Relations	Description		

Je weet nu hoe je een beschrijving van een datamodel moet maken.

Opdracht 5.3: Pas de database aan, op basis van het datamodel: Voeg tabel Meals toe

Nu je het datamodel en ERD hebt gemaakt, kun je de database aanpassen.

- Voeg aan de database LosPollosHermanos de tabel *Meals* toe.
- Zorg dat de kolommen het juiste type hebben, en met uitzondering van de Description, een verplichte waarde hebben (= not null zijn).
- Zorg dat de primary key goed is ingesteld, en auto incremented is.
- Zorg dat de name uniek is.
- Zorg dat de price 2 decimalen heeft.

Opmerking: Je zou kunnen bedenken dat de Name een primary key is. Immers: de Name moet uniek zijn. We kiezen liever voor een auto increment als primary key. Hierdoor heeft deze id zelf geen functionele waarde. Daarnaast kunnen we de Name een andere waarde geven (dat zou niet kunnen, als de Name een primary key zou zijn).

Opdracht 5.4: Maak Model class Meal

In het datamodel heb je een nieuwe entiteit (tabel) gemaakt: Meals. Voor elke datamodel tabel maak je een Model Class. Een Model Class beschrijft 1 rij van de tabel; de naam zet je dan ook in het enkelvoud.

Het gebruik maken van classen is een onderdeel van object georiënteerd programmeren.

- Maak een (domain model) class voor de tabel Meals; geef de class de naam Meal.

```
using System.Threading.Tasks;

namespace Lj2Dd1En2.Models
{
    // Meal bevat de eigenschappen die over een maaltijd.
    public class Meal
    {
        private int mealId;

        public int MealId
        {
            get { return mealId; }
            set { mealId = value; }
        }

        private string name = null;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        private string? description;

        public string? Description
        {
            get { return description; }
            set { description = value; }
        }

        private decimal price;

        public decimal Price
        {
            get { return price; }
            set { price = value; }
        }
    }
}
```

1. De naam van de class begint met een hoofdletter. De naam staat in *enkelvoud*: in de class beschrijf je hoe 1 maaltijd eruit ziet (dit i.t.t. tabelnamen in een database, maar daar staan dan ook meer rijen in)

2. Voor elke kolom uit de databasetabel maak je een **property**. Een property bestaat uit een private field en een public structuur met een getter en/of een setter. Doordat de fields private zijn, kan alleen de class zelf ze zien; andere classes kunnen dat niet. Dit noem je *encapsulation*.
3. Met de getter (get) zorg je ervoor dat het private field gelezen kan worden.
4. Met de setter (set) zorg je ervoor dat het private field een waarde krijgt.
5. Als je een reference variabel niet een initiële waarde geeft, volgt een *warning*. Omdat null geen geldige reference waarde is, moet je in C# expliciet aangeven dat je dat toch zo wilt hebben. Dat doe je met een !-teken.
6. Hier zie je achter het type een ?-teken staan. Daarmee geef je aan dat het een *nullable* is. In dit geval mag je wel de waarde null aan de variabele toekennen.

Opmerking:

- Omdat *Name* verplicht ingevuld moet zijn (datamodel zegt *Required = Ja*), laat je het vraag-teken achter string weg. Immers, er moet een waarde in Name staan (daar wordt bij het inlezen van de gegevens uit de database voor gezorgd).
- In het datamodel staat dat *Description* geen verplicht gegeven is. Om die reden zet je achter het type van deze variabele een vraagteken: de waarde *null* is hier een geldige waarde en dus toegestaan.

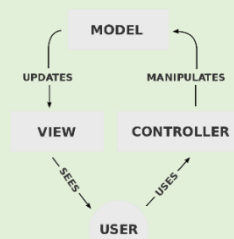
MVC staat voor Model View Control

Model: Hierin worden de classes uitgewerkt die het domain van de applicatie omvatten.

Praktisch zullen we hier voor elke databasetabel 1 class in zetten.

View: Hierin worden de classes uitgewerkt die een interface met de gebruiker vormen: de window-classes dus.

Control: Hierin zet je de classes die verzoeken van een gebruiker afwikkelen. Deze verzoeken komen binnen via een View-class. De control class zal eventueel een aantal model classes gebruiken (lezen/schrijven) om het verzoek te verwerken en om output voor de gebruiker in een view class klaar te zetten.



NB: Ingeval van WPF hebben we deze code vooralsnog in de bij de XAML horende cs staan. Bij ASP.NET zullen deze classes in een aparte folder gezet worden.

Zie ook: [Model-view-controller - Wikipedia](#)

Opdracht 5.5: Maak de method *GetMeals*

- Maak de method *GetMeals* in de databaseclass, die als doel heeft om alle rijen uit de database tabel *Meals* in te lezen. Elke rij moet in een nieuw object van het type *Meal* gezet worden. Alle objecten worden in een *ICollection* met *Meal* objecten gezet.

Bestudeer de volgende code en type deze over:

```

//1 GetMeals leest alle rijen in uit de databasetabel Meals en voegt deze toe aan een ICollection.
// Als de ICollection bij aanroep null is, volgt er een ArgumentException
// De waarde van GetMeals:
// - "ok" als er geen fouten waren.
// - een foutmelding, als er wel fouten waten (mogelijk zijn niet alle maaltijden ingelezen)
public string2 GetMeals(ICollection4<Meal> meals)
{
    if (meals == null)
    {
        Throw3 new ArgumentException("Ongeldig argument bij gebruik van GetMeals");
    }

    string methodResult = "unknown";

    using5 (MySqlConnection conn = new (connString))
    {
        Try6
        {
            conn.Open();
            MySqlCommand sql = conn.CreateCommand();
            sql.CommandText = @"
                SELECT m.mealId, m.name, m.description, m.price
                FROM meals m
            ";
            MySqlDataReader reader = sql.ExecuteReader();

            while (reader.Read())7
            {
                Meal meal = new Meal()

```

```

        {
            MealId = (int)reader["mealId"],
            Name = (string)reader["name"],
            Description = reader["description"] == DBNull.Value
                ? null
                : (string)reader["description"],
            Price = (decimal)reader["price"],
        };

        meals.Add(meal);
    }

    methodResult = "ok";
}
catch (Exception e)
{
    Console.Error.WriteLine(nameof(GetMeals));
    Console.Error.WriteLine(e.Message);
    methodResult = e.Message;
}
}
return methodResult;
}

```

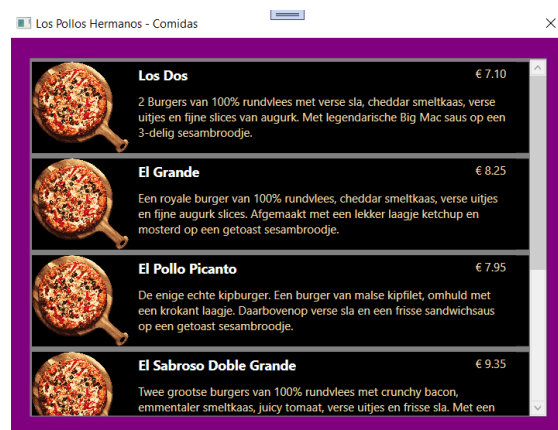
1. Commentaar is verplicht. Beschrijf het *doel* van de method (wat is er gebeurd als de method zijn werk heeft gedaan), de *parameters* en het *resultaat* van de method.
2. Het type van GetMeals is een string. Dit betekent dat er tenminste 1 return statement moet zijn. Achter elk return statement staat een string. In deze string staat de waarde van GetMeals (in dit geval : "ok", of een melding uit de exception). **In je databaseclass toon je nooit meldingen aan gebruikers; daarvoor is je Window class bedoeld.**
3. Aan de method moet een geïnstantieerde ICollection (= een met new gemaakte ICollection) als argument worden doorgegeven. Is dat niet het geval, dan volgt er een *exception* zodat jij als programmeur weet dat je nog wat moet aanpassen in je programma.
4. ICollection<Meal> staat voor een lijst met Meals. Door een ICollection te gebruiken, mag je er bij het gebruik een List<Meal> variabele aan meegeven, maar bijvoorbeeld ook een ObservableCollection<Meal>.
5. Dit using statement wordt gebruikt om er zeker van te zijn dat de MySqlConnection wordt gesloten als het statement klaar is (dus bij de sluit-accolade is aangekomen).
6. De try-catch-constructie gebruik je hier net als in het eerste jaar. In de catch vang je een eventuele exception af: je zet de melding van de exception in het result, zodat je MainWindow het aan de gebruiker kan laten zien. De tekst naar het Console is voor een technisch onderlegt persoon bedoeld (niet voor de gebruiker).
7. In de while, worden de rijen uit de database tabel via de reader 1 voor 1 ingelezen. Voor elke rij wordt er een nieuw Meal gemaakt. De gegevens uit de kolommen worden direct omgezet naar het juiste type. Hierdoor kun je ze beter gebruiken in jouw programma. Elke Meal wordt toegevoegd aan de ICollection<Meal> (lijst met maaltijden).
8. De test op DBNull.Value is hier nodig omdat Description de waarde null kan hebben. Dit moet je op deze manier afvangen; anders gaat het mis.

ICollection is geen *class* maar een *interface*. Dat kun je zien, omdat de naam met 2 hoofdletters begint, waarvan de eerste een I is. Een interface lijkt op een class, zoals MainWindow of Meal, maar is dat niet. Een interface geeft alleen aan dat de de methods en properties die erin zijn genoemd, ook in de class staan die hem implementeert.

Opdracht 5.6: Maak de View DisplayWindow

In deze opdracht maak je gebruik van de method GetMeals om het Display voor de bediening in het restaurant te maken. Hierbij zorg je ervoor dat de gegevens via *binding* aan het scherm worden doorgegeven.

1. Maak in de folder Views een nieuwe Window class: **class DisplayWindow**
2. Zorg ervoor dat DisplayWindow getoond wordt, zodra de applicatie start, door de StartupUri in app.xaml aan te passen: **StartupUri="Views/DisplayWindow.xaml"**
3. Verwijder MainWindow
4. Maak de lay-out: bestudeer de volgende code en toelichting. Type code over:



```

<Grid Margin="20">
    <ListView ItemsSource="{Binding Meals}"1 SelectedItem="{Binding SelectedMeal}"2>
        <ListView.ItemTemplate>3
            <DataTemplate>
                <Grid Background="Black" Width="500">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <Image Grid.Column="0"
                        Width="100" Stretch="UniformToFill"
                        Source="/Assets/pizzeria_14.png"/>
                    <Grid Grid.Column="1" Margin="10, 2">
                        <Grid.RowDefinitions>
                            <RowDefinition Height="Auto"/>
                            <RowDefinition />
                        </Grid.RowDefinitions>
                        <Grid Grid.Row="0" Margin="0, 2, 0, 10">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="Auto"/>
                                <ColumnDefinition />
                            </Grid.ColumnDefinitions>
                            <TextBlock Grid.Column="0" Text="{Binding Name}"
                                Foreground="White" FontSize="14" FontWeight="Bold"/>
                            <TextBlock Grid.Column="1"
                                Text="{Binding Price", StringFormat='€ 0.00'}"
                                HorizontalAlignment="Right" Margin="10,0,0,0" />
                        </Grid>
                        <TextBlock Grid.Row="1" Text="{Binding Description}"
                            TextWrapping="Wrap" />
                    </Grid>
                </Grid>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</Grid>

```

Merk op dat je geen x:name meer gebruikt. Dat is bijna nooit meer nodig als je met *binding* werkt. Over *binding* vind je in een volgend hoofdstuk meer informatie.

Toelichting bij de nummers:

1. Hier *bind* je de *ItemsSource* aan een property met de naam *Meals*. De property *Meals* staat in jouw cs code (zie hieronder) en is een lijst met elementen van het type *Meal*.
2. Hier *bind* je het *SelectedItem* aan de property *SelectedMeal*. Zodra de gebruiker op een regel/item clickt, staat dit item ook in de property *SelectedMeal*. *SelectedMeal* is dus van het type *Meal*.
3. Hier maak je een template. Wpf zorgt ervoor dat elke maaltijd uit de *ItemsSource* opgemaakt wordt volgens deze template.
4. Hier worden van een element uit de *ItemsSource* (dus van de property *Meals*), de Name, de Price en de Description getoond. Name, Price en Description zijn op hun beurt weer properties van de class *Meal*.

In dit voorbeeld worden de meals in een ListView gezet. Voor andere controls, zoals ComboBox, List, DataGrid is de werkwijze vergelijkbaar:

- Bind de itemsSource aan een property die alle items van de lijst bevat
- Bind de selectedItem aan een property
- Maak een template, waarin je alle gegevens toont op de manier zoals jij dat wilt.

5. Maak de code van DisplayWindow: bestuur de volgende code en toelichting. Type code over:

```

using System.Windows;

using Lj2Dd1En2.Models;
using System.Collections.ObjectModel;

using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace Lj2Dd1En2.Views
{
    /// <summary>
    /// Interaction logic for DisplayWindow.xaml
    /// </summary>
    public partial class DisplayWindow : Window, INotifyPropertyChanged1
    {

```

```

#region INotifyPropertyChanged
public event PropertyChangedEventHandler? PropertyChanged;
protected void OnPropertyChanged([CallerMemberName] string? name = null)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
}
#endregion

#region fields
private readonly LosPollosHermanosDb db = new LosPollosHermanosDb();
private readonly string serviceDeskBericht = "\n\nNeem contact op met de service desk";
#endregion

#region Properties
private ObservableCollection<Meal> meals = new ();

public ObservableCollection<Meal> Meals3
{
    get { return meals; }
    set { meals = value; OnPropertyChanged(); }
}

private Meal? selectedMeal;

public Meal? SelectedMeal3
{
    get { return selectedMeal; }
    set { selectedMeal = value; OnPropertyChanged(); }
}
#endregion
public DisplayWindow()
{
    InitializeComponent();
    PopulateMeals();
    DataContext = this7;
}

// Method zet alle maaltijden uit de database op het scherm in de control lvMeals
// Trad er een fout op bij het inlezen, wordt hiervan een melding getoond.
private void PopulateMeals()
{
    string dbResult5 = db.GetMeals(Meals)4;
    if (dbResult != LosPollosHermanosDb.OK6)
    {
        MessageBox.Show(dbResult + serviceDeskBericht);
    }
}
}
}

```

1. **Geel:** Deze code implementeert de interface `INotifyPropertyChanged` (over interfaces krijg je in een volgend hoofdstuk meer informatie). Voeg deze code standaard toe aan elke class die je gebruikt. `INotifyPropertyChanged` zorgt ervoor dat wanneer jij een property een andere waarde geeft, dit wordt doorgegeven aan je scherm, zodat daar de nieuwe waarde zichtbaar wordt. Dit is de code die ook door Microsoft wordt gebruikt: google maar eens op "*WPF How to implement INotifyPropertyChanged*".
2. **Groen:** De `ObservableCollection` is een List die speciaal is gemaakt voor het gebruik bij Binding. Je maakt deze alleen bij de declaratie aan (new()). Voeg je er nieuwe elementen aan toe, of verwijder je er elementen uit, dan wordt automatisch het scherm hiermee bijgewerkt. Bij een List gebeurt dat niet.
3. **Blauw:** `Meals` en `SelectedMeal` zijn gebind in de XAML aan `ItemsSource` en `SelectedItemSource`. Alles wat je hierin wijzigt, komt dankzij `INotifyPropertyChanged`, vanzelf op het scherm terecht. Wijzigt de gebruiker iets op het scherm? Dan staat het ook direct in deze 2 properties (in dit geval zal het dan vooral om `SelectedMeal` gaan, wat steeds de Meal bevat waar de gebruiker op clickt). Heeft de gebruiker niets geselecteerd, dan bevat `SelectedMeal` de waarde null.
4. Met `db.GetMeals(Meals)` gebruik je de method `GetMeals` van de database class. Als argument geef je de property `Meals` mee. Dit is een `ObservableCollection` die prima in een `ICollection` past. `Meals` is hier ongelijk aan null, want bij de declaratie heb je hem met een `new` gemaakt.
5. Het resultaat bewaar je in de string variabele `dbResult`. Immers, `GetMeals` is van het type string. Je test na gebruik wat het resultaat is. Is dat ok, dan doe je niks. Anders toon je een heldere instructie aan de gebruiker.
6. Met `LosPollosHermanosDb.OK` gebruik je een static variabele uit de class `LosPollosHermanosDb` (de databaseclass). Op deze manier zorg je ervoor dat je zeker weet dat je in dezelfde tekst voor OK gebruikt in allebei de classes (`DisplayWindow` class en de `LosPollosHermanosDb` class).

7. Je moet opgeven in welke DataContext, WPF de properties kan vinden om te binden. De properties heb je in het object zelf gemaakt, dus geef je hieraan dat de DataContext *this* is.

Merk op: In de setter van SelectedMeal kun je ook een method van je programma gebruiken. Dat werkt dan hetzelfde als een SelectionChanged event.

Je weet nu hoe je binding toepast op een lijst. Ook weet je hoe je INotifyPropertyChanged implementeert.

Interfaces

Inleiding

In opdracht 5 heb je de interface *INotifyPropertyChanged* geïmplementeerd. Hiervoor heb je simpelweg de programma code overgenomen, die al gegeven was (= de code van Microsoft).

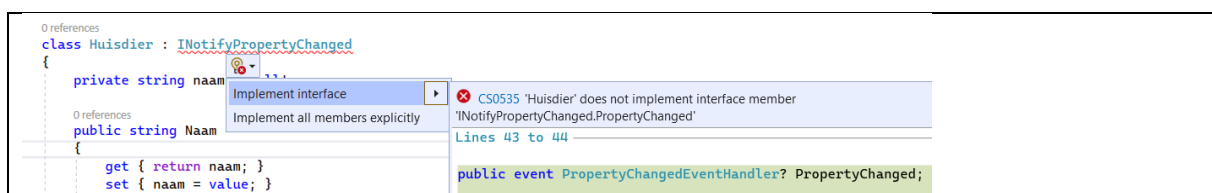
Een interface zelf doet helemaal niets; je kunt er niet eens, zoals van een class, een object van maken. In een interface geef je alleen maar op wat er in de class, die de interface implementeert, moet staan.

Interface *INotifyPropertyChanged*

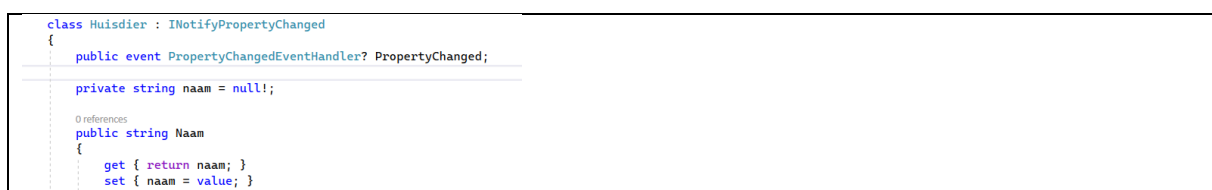
```
public interface INotifyPropertyChanged
{
    //
    // Summary:
    //     Occurs when a property value changes.
    event PropertyChangedEventHandler? PropertyChanged;
}
```

Hierboven zie je de code van de interface *INotifyPropertyChanged*. In deze interface zie je dat er een *PropertyChangedEventHandler* event staat met de naam *PropertyChanged*.

Als een class de interface *INotifyPropertyChanged* implementeert, dan betekent dit dat die class ook een *PropertyChangedEventHandler* event moet implementeren (=maken) en met dezelfde naam *PropertyChanged*.



In bovenstaande code zie je een gedeelte van de class *Huisdier*. Bij de class heb je aangegeven dat deze de interface *INotifyPropertyChanged* implementeert. Omdat je het event *PropertyChanged* nog niet hebt gemaakt (geïmplementeerd) krijg je nog een rode streep. Visual Studio kan jou helpen om dit event toe te voegen. Nadat je dit hebt gedaan, is de rode streep weg: je hebt het event immers aan de class *Huisdier* toegevoegd.



Interfaces en Classen

Een interface lijkt qua structuur wel wat op een class. Toch zijn er een paar belangrijke verschillen:

- In plaats van het keyword *class*, gebruik je het keyword *interface*.
- De naam begint met 2 hoofdletters, waarvan de eerste een hoofdletter i.
- Een interface kan een onbeperkt aantal interfaces implementeren, maar geen enkele class. Een class kan een onbeperkt aantal interfaces implementeren en maximaal een class.
- Je kunt van een class een object maken, van een interface kan dat niet. Dat is ook de reden dat je alles wat in een interface staat, moet maken in de class die de interface gebruikt.

Daarnaast zijn er ook verschillen voor wat betreft de inhoud, zoals

- In een interface kun je geen fields zetten.
- Verkorte properties mag wel, maar full properties niet (want dan zou je een field nodig hebben),
- Private wordt in een class veel gebruikt; in een interface is dat zinloos (want private betekent dat het buiten de interface niet zichtbaar is en dus ook niet gemaakt kan worden)
- In een interface heeft een method geen body; deze maak je namelijk in de class die de interface implementeert (zie echter kader hierna).

Interface IVerkoopbaar

Je gaat nu zelf een interface maken voor verkoopbare dingen, zoals huisdieren en gebouwen. Zowel een huisdier als een gebouw is verkoopbaar.

Maak een WPF programma en zet daar de volgende interface in:

<pre>interface IVerkoopbaar { 10 references public decimal EenheidsPrijs { get; set; } 3 references public decimal Bedrag(int aantal); }</pre>	<ul style="list-style-type: none">• In deze interface geef je aan dat wanneer een class deze interface implementeert, er in deze class een <i>property</i> <i>EenheidsPrijs</i> en een <i>method</i> <i>Bedrag</i> moet staan.• Merk op dat de method geen body heeft: er staat niet bij wat <i>Bedrag</i> doet; dat laat de interface over aan de class die de interface implementeert.
--	---

Je gaat nu de interface gebruiken in de classes Huisdier en Gebouw. Voeg de volgende code aan jouw programma toe:

<pre>class Gebouw : INotifyPropertyChanged, IVerkoopbaar { INotifyPropertyChanged private string adres = string.Empty; 0 references public string Adres[...] private decimal eenheidsPrijs; 2 references public decimal EenheidsPrijs[...] 1 reference public decimal Bedrag(int aantal) { return aantal * EenheidsPrijs; } }</pre>	<pre>class Huisdier : INotifyPropertyChanged, IVerkoopbaar { INotifyPropertyChanged private string naam = string.Empty; 0 references public string Naam[...] private string? ras; 0 references public string? Ras[...] private decimal eenheidsPrijs; 2 references public decimal EenheidsPrijs [...] 1 reference public decimal Bedrag(int aantal) { return aantal * EenheidsPrijs; } }</pre>
<ul style="list-style-type: none">• Hierboven zie je dat zowel Huisdier als Gebouw de interface implementeert.• In allebei de classes is de property <i>EenheidsPrijs</i> en de method <i>Bedrag</i> opgenomen.• De <i>Eenheidsprijs</i> is hier een full property (wat in de interface niet kon).• De method <i>Bedrag</i> heeft hier ook een body.• Allebei de elementen (eenheidsprijs en bedrag) zijn dus in elk van de classes gemaakt.	

Vanaf C# 8.0 is het mogelijk om in een interface een method met een body op te nemen. Deze body wordt dan gebruikt, als een class de method niet implementeert. Hiermee wordt voorkomen dat er breaking code ontstaat, indien een interface met een nieuwe method wordt uitgebreid.

Wat heb je er aan

Interfaces zijn erg handig als je met verschillende soorten gegevens werkt, waarvan je toch een zelfde soort gedrag verwacht. Kijk je naar in het vorige voorbeeld naar Huisdieren en Gebouwen, dan hebben deze gegevens niet zoveel met elkaar te maken.

Wat ze wel gemeenschappelijk hebben is alles wat je in de interface IVerkoopbaar hebt gezet (want anders had C# gezegd dat er een fout in je programma stond):

- een property eenheidsprijs
- een method bedrag

Omdat zowel Huisdieren als Gebouwen IVerkoopbaar zijn, kun je ze in een lijst met IVerkoopbaar objecten zetten. Dat gaan je met de volgende code doen:

```
List<IVerkoopbaar> verkoopbaarList = new List<IVerkoopbaar>();

1 reference
public void PopulateList()
{
    Gebouw tuinhuisje = new Gebouw() { Adres = "Achtertuin 25, achteraf", EenheidsPrijs = 2200.00m };
    Gebouw garage = new Gebouw() { Adres = "Autostraat 33, centrum", EenheidsPrijs = 12000.00m };
    Huisdier minous = new Huisdier() { Naam = "Minous", Ras = "vuilnisbak", EenheidsPrijs = 7.50m };
    Huisdier marielousie = new Huisdier() { Naam = "Marie Louise", Ras = "Hanbuijzwijn", EenheidsPrijs = 3023.00m };
    Huisdier kakel = new Huisdier() { Naam = "Juffrouw Kakel", Ras = "kip", EenheidsPrijs = 13.50m };

    verkoopbaarList.Add(tuinhuisje);
    verkoopbaarList.Add(garage);
    verkoopbaarList.Add(minous);
    verkoopbaarList.Add(marielousie);
    verkoopbaarList.Add(kakel);
}
```

- Als eerste heb je een List gemaakt waarin je allemaal gegevens mag zetten van het type IVerkoopbaar. De naam van de List is *verkoopbaarList*.
- In *PopulateList* zet je vervolgens gegevens van het type Gebouw en van het type Huisdier in deze List. Dat mag omdat zowel Gebouw als Huisdier de interface IVerkoopbaar implementeren.

Nu je de verzameling met verkoopbare spullen gemaakt hebt, kun je ook een overzicht tonen van de eenheidsprijzen en bedragen. Neem de volgende code weer over in je programma:

```
public void ShowAll()
{
    foreach (IVerkoopbaar verkoopbaar in verkoopbaarList)
    {
        Console.WriteLine($"{verkoopbaar.GetType().Name}\t{verkoopbaar.EenheidsPrijs}\t{verkoopbaar.Bedrag(2)}");
    }
}
```

- In bovenstaande code schrijf je voor elk element uit de lijst *verkoopbaarList* een regel naar het console (*vergeet niet om de properties van je project op Console te zetten*).
- Van elk element wordt de naam van het type getoond, de eenheidsprijs en het bedrag als je er 2 van zou verkopen. Dat kan, want de elementen zijn van het type IVerkoopbaar, en hierin staan deze gegevens.
- Merk op dat je hier geen ras of adres kunt tonen. Dat komt omdat noch een ras, noch een adres in de interface IVerkoopbaar staan en c# niet beter weet dan dat de elementen van het type IVerkoopbaar zijn (List<IVerkoopbaar>).
- Wil je toch een adres of ras tonen? Dan zul je de waarde van de variabele verkoopbaar moeten casten, zoals je dat ook deed bij het inlezen van gegevens uit de database. Met de operator *is*, kun je testen of de variabele verkoopbaar een object bevat van het type Gebouw of Huisdier; bijvoorbeeld:

```
if (verkoopbaar is Huisdier)
{
    Huisdier huisdier = (Huisdier)verkoopbaar;
    Console.WriteLine(huisdier.Ras);
}
```

Opdracht: Interfaces

- Werk de code die in dit hoofdstuk staat uit in een programma.
- Voeg nog een extra class toe die de interface IVerkoopbaar implementeert. Je mag zelf weten wat je in deze class zet, als het maar iets is dat verkoopbaar is (voorbeeld Games, Autos, Voetbalspelers, etc.).
- Zorg ervoor dat je naast de algemene IVerkoopbaar gegevens, tenminste 1 gegeven toont van de class die IVerkoopbaar implementeert. Dus voor gebouw toon je bijvoorbeeld het adres, voor huisdier toon je in de regel het ras. Zo ook voor de door jouw gemaakte class.

Conclusie

Interfaces worden zeer veel gebruikt. Je kunt er eigenschappen en gedrag in noemen. Je weet zeker dat wanneer een class de interface implementeert, deze class die eigenschappen en gedrag ook heeft. Dat is met name handig wanneer je van gegevens die niets met elkaar te maken hebben (bijvoorbeeld Huisdieren en Gebouwen), toch iets gemeenschappelijks wilt weten (bijvoorbeeld EenheidsPrijs, Bedrag).

Je maakt er zelf ook veel gebruik van:

- INotifyPropertyChanged: voor zowel Window classen als data classen implementeer je deze interface ten behoeve van *binding*
- ICollection: zowel List als ObservableCollection implementeren de interface ICollection (zoek maar eens op).

Met de opdracht uit dit hoofdstuk heb je laten zien dat je ook zelf een interface kunt maken die je bovendien weet toe te passen.

Hiermee is dit onderdeel afgerond. In het volgend hoofdstuk krijg je nog toelichting op het toepassen van binding. Daarna ga je weer aan de slag met de app voor LosPollosHermanos.

Binding

Inleiding

In opdracht 5 heb je kennis gemaakt met Binding. Binding is een techniek waarmee je properties uit je programma aan XAML code koppelt. Wijzigen de waarden van de properties, dan wordt dit direct zichtbaar op het scherm. Maar ook, als gegevens op het scherm wijzigen (gebruiker selecteert een regel uit een lijst of typt een naam van een klant in), dan zijn deze gegevens direct beschikbaar in de gekoppelde properties.

Binding is daarom erg handig en als je het eenmaal onder de knie hebt, bespaart het je veel tijd. Toepassen van de techniek is niet moeilijk, zolang je je maar aan het bindingrecept houdt. Dit recept vind je in dit hoofdstuk. Naast dit recept vind je hier nog informatie over een paar bijzondere situaties waarmee je rekening moet houden. We sluiten dit hoofdstuk af met een korte conclusie.

Recept

Implementeer de interface *INotifyPropertyChanged*

Om ervoor te zorgen dat gegevens in je programma automatisch bijgewerkt worden in het window (xaml), moet je de interface *INotifyPropertyChanged* implementeren. Microsoft heeft hier zelf richtlijnen voor opgesteld, dus je kunt de code die je moet gebruiken op internet vinden.

- Google: *WPF How To Implement INotifyPropertyChanged*

Op internet vind je bij de site van microsoft, de volgende code:

CS-code

```
...
using System.ComponentModel;
using System.Runtime.CompilerServices;
...

public partial class EenWindow : Window, INotifyPropertyChanged
{
    #region INotifyPropertyChanged
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
    #endregion

    ...
}
```

- #region ... #endregion zorgt ervoor dat je deze code kunt inklappen.
- Alleen de event hoeft volgens de interface *INotifyPropertyChanged* gemaakt te worden. De method *OnPropertyChanged* hoort bij de code die Microsoft voorschrijft en we hier dus overnemen.
- Vergeet je achter je class name *INotifyPropertyChanged* te zetten, dan krijg je hiervan geen foutmelding. Maar de binding zal niet werken. Bij problemen: controleer dat *INotifyPropertyChanged* achter je class name staat.

CS-code

```
public class EenClass : INotifyPropertyChanged
{
    #region INotifyPropertyChanged
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
    #endregion
    ...
}
```

- *INotifyPropertyChanged* implementeer je ook in de *data* classen die je gebruikt in je windows.
- Vergeet je achter je class name *INotifyPropertyChanged* te zetten, dan krijg je hiervan geen foutmelding. Maar de binding zal niet werken. Bij problemen: controleer dat *INotifyPropertyChanged* achter je class name staat.

Gebruik Properties en voeg OnPropertyChanged toe aan de setters

Als je met binding werkt moet je *properties* gebruiken. In de setter van de full property voer je de method *OnPropertyChanged* uit:

CS-code

```
#region properties
private string name;

public string Name
{
    get { return name; }
    set { name = value; OnPropertyChanged(); }
}
#endregion
```

XAML-code

```
<TextBox Text="{Binding Name}" />
```

Je ziet hier een property met de naam **Name** van het type **string**.

- De **property** zelf is **public**.
- De **waarde** van de property **wordt opgeslagen in** het **private field name**.
- De **naam** van de public **property** schrijf je met een **hoofdletter**.
- De **naam** van het **field** met een **kleine letter**.

Als de gebruiker zijn naam in de *TextBox* intypt, dan staat deze naam ook meteen in de property *Name*. Je hebt dus geen *x:name* meer nodig voor de *TextBox*.

Als jouw programma een andere naam in de property *Name* zet, dan is deze naam ook meteen zichtbaar in de *TextBox*. Hiervoor zorgt *OnPropertyChanged()*.

DataContext

Je moet opgeven waar WPF de properties die gebind zijn, kan vinden. Dit doe je via de *DataContext*. Elk control heeft een *DataContext*. In je .cs code geef je de *DataContext* voor je Window control op. In de XAML geef je eventueel de *DataContext* voor een andere control op:

CS-code

```
public partial class DisplayWindow : Window, INotifyPropertyChanged
{
    INotifyPropertyChanged
    ...
    #region properties
    private Meal newMeal = new ();           // Variabele newMeal krijgt meteen een waarde
    public Meal NewMeal                      // Property NewMeal gebruikt variabele newMeal
    {
        get { return newMeal; }
        set { newMeal = value; OnPropertyChanged(); }
    }
    #endregion
    ...
    public MealWindow()
    {
        ...
        InitializeComponent();
        ...

        DataContext = this;
    }
}
```

- Met **DataContext = this**; geef je aan dat WPF naar properties moet zoeken die in de cs-code van het MealWindow (**this**) staan. In dit geval is dat alleen de property **NewMeal**.

Pas op: Als je deze code vergeet, krijg je hiervan *geen* foutmelding. Maar de binding zal *niet* werken. Controleer bij problemen dus altijd of je deze code wel hebt toegevoegd.

Merk op: NewMeal maak je meteen aan met een *new* opdracht. Doe je dat niet, dan wordt de waarde null gebind, wat zoveel betekent als *bind helemaal niets*. *Vergeet dus niet om objecten direct te maken*.

XAML-code bij bovenstaande CS-code

```
<Grid DataContext="{Binding NewMeal}">
  <Grid.ColumnDefinitions ...>
  <Grid.RowDefinitions ...>

  <TextBlock Grid.Column="0" Grid.Row="1" Text="Naam:" />
  <TextBox Grid.Column="1" Grid.Row="1" Text="{Binding Name}" />

  <TextBlock Grid.Column="0" Grid.Row="2" Text="Omschrijving:" />
  <TextBox Grid.Column="1" Grid.Row="2" Text="{Binding Description}" />
</Grid>
```

- Hierboven wordt de property **NewMeal** uit de datacontext *this* (dus MealWindow.xaml.cs) gebruikt in de XAML.
- Je *bind* hier NewMeal 'verder' aan de DataContext van een Grid. Hierdoor zal WPF voor de properties die je in de Grid gebruikt, gaan zoeken in **this.NewMeal**. Hier vind WPF *Name* en *Description*
- Zou je NewMeal niet aan de datacontext van de Grid *binden*, dan had je in plaats van `{Binding Name}` de code `{Binding NewMeal.Name}` moeten gebruiken.

Recept voor Lijsten

Bind een collection aan ItemsSource en een single object aan SelectedItem

- Maak 2 properties aan in de CS-code:
 - 1 voor de lijst met elementen. Deze is van het type **ObservableCollection**.
 - 1 voor de geselecteerde rij uit de lijst. Deze is van het type dat *in* de ObservableCollection zit.
- In de XAML code, in de ListView/ComboBox/DataGrid, etc:
 - Bind de property *Meals* aan de ItemsSource en de property
 - Bind de property *SelectedMeal* aan het SelectedItem etc..

1. CS-code

```
private Meal selectedMeal;
public Meal SelectedMeal // 1 Meal, naam is een enkelvoud : SelectedMeal
{
    get { return selectedMeal; }
    set { selectedMeal = value; OnPropertyChanged(); }
}

private ObservableCollection<Meal> meals = new();
public ObservableCollection<Meal> Meals // een lijst: naam is een meervoud: Meals
{
    get { return meals; }
    set { meals = value; OnPropertyChanged(); }
}
```

2. XAML code

```
<ListView ItemsSource="{Binding Meals}"
  SelectedItem="{Binding SelectedMeal}" >
  <ListView.ItemTemplate>
    <DataTemplate>
      <StackPanel>
        <TextBlock Text="{Binding Name}" />
        <Button Content="wijzigen" Click="BtnUpd_Click" />
        <Button Content="verwijderen" Click="BtnDel_Click"/>
      </StackPanel>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

SelectionChanged

- SelectionChanged event nodig? Dat hoef je dat niet meer te maken. Je kunt simpelweg de code toevoegen aan de setter van de property die je gebind hebt aan het SelectedItem.

3. In plaats van SelectionChanged-event:

```
public Meal SelectedMeal // Nogmaals SelectedMeal, maar nu met uitgebreide
{ // setter
```

```

get { return selectedMeal; }
set
{
    selectedMeal = value;
    PopulateMealIngredients(); // Wijzigt het SelectedMeal, werk dan het
                               // overzicht met maaltijdingredienten bij
    OnPropertyChanged();
}
}

```

- Clickt de gebruiker op een element uit de ListView, dan zal de setter van SelectedMeal worden uitgevoerd, omdat je SelectedMeal aan het SelectedItem van de ListView hebt gebind.
- In plaats van een SelectionChanged event, kun je dus net zo goed in de setter code opnemen die uitgevoerd moet worden. In dit geval is dat PopulateIngredients().

Knoppen in een regel

Via de DataContext van de Button in de regel waar de Button staat (zie hiervoor bij 2), heb je toegang tot het object van de regel. Voorbeeld:

4. Button_Click

```

private void BtnUpd_Click(object sender, RoutedEventArgs e)
{
    Button btnUpd = sender as Button;
    Meal meal = btnUpd.DataContext as Meal; // Dit is het Meal in de regel met de Update Button

    if (new MealUpdateWindow(meal.MealId).ShowDialog() == true)
    {
        PopulateMeals();
    }
}

```

- Als je in een regel van de ListView een Button opneemt, en de gebruiker clickt op deze button, dan heeft kun je via de DataContext van deze Button het element uit de ListView ophalen waarpo was geclickt. In het voorbeeld met Meals, is het type van dat element een Meal.

Bijzondere situaties

Automatisch een leeftijd tonen (afleidbare gegevens)

Een leeftijd is een afleidbaar gegeven. Hiermee wordt bedoeld dat je een leeftijd geen waarde geeft maar dat je hiervoor de geboortedatum gebruikt. Als je van iemand de geboortedatum weet en je weet welke dag het is, dan kun je zijn leeftijd afleiden uit de dag en de geboortedatum:

- Geboortedatum : 12 januari 2004
- Vandaag : 22 maart 2022
- Leeftijd (vandaag) : 18 jaar (berekend)

```

public class Person : INotifyPropertyChanged
{
    + #region INotifyPropertyChanged

    private int personId;
    + public int PersonId ...

    private DateTime dateOfBirth;

    public DateTime DateOfBirth
    {
        get { return dateOfBirth; }
        set { dateOfBirth = value; OnPropertyChanged(); OnPropertyChanged(nameof(Age)); }
    }

    public int Age
    {
        get
        {
            int age = 0;

```

```

        age = DateTime.Now.Year - dateOfBirth.Year;
        if (DateTime.Now.DayOfYear < dateOfBirth.DayOfYear)
        {
            age = age - 1;
        }
        return age;
    }
}

```

In bovenstaande class zie je dat wanneer je de geboortedatum wijzigt, er een `OnNotifyPropertyChanged()` gecodeerd is. De geboortedatum wordt zichtbaar op het scherm. Maar je ziet hier ook een `OnPropertyChanged(nameof(Age));`. Dit betekent dat de waarde van Age ook wordt bijgewerkt op het scherm.

Een read-only property in een TextBox (one-way-mode)

Een read-only property is een property waarvan alleen de getter zichtbaar is. In voorgaand voorbeeld is Age een read-only property. Als je een read-only property aan een control bind, dan doe je dat standaard aan een TextBlock. Een TextBlock is bedoeld om een waarde (leeftijd in dit geval) te tonen.

Kies je om wat voor reden toch voor een TextBox, dan leidt dit ingeval van een readonly property tot een fout:

```

<StackPanel DataContext="{Binding Person}">
    <TextBlock Text="Datum:"/>
    <DatePicker SelectedDate="{Binding DateOfBirth}"/>
    <TextBlock Text="Leeftijd"/>
    <TextBox Text="{Binding Age}"/>
</StackPanel>

```

Exception Unhandled

System.InvalidOperationException: 'A TwoWay or OneWayToSource binding cannot work on the read-only property 'Age' of type 'MyFriendsApp.Person'.'

This exception was originally thrown at this call stack:
[External Code]

[Copy Details](#) | [Start Live Share session...](#)

Exception Settings

☐ Break when this exception type is thrown

[Open Exception Settings](#)

In de melding staat precies aangegeven wat de situatie is: *Er is sprake van een TwoWay of van een OneWayToSource binding*. Ofwel: De gebruiker mag een Age intypen (want het is een TextBox) en deze Age wordt dan naar de property Age van de Person gestuurd. En dat mag in dit geval niet, omdat er geen setter in de Age property zit.

Hoe los je dit op?

- Door een TextBlock control te gebruiken; maar dat was hier nu net niet de bedoeling.
- Door een Mode eigenschap toe te voegen, waarin je aangeeft dat er OneWay binding moet worden toegepast.

```

Title="MainWindow" Height="450" Width="800">
<StackPanel DataContext="{Binding Person}">
    <TextBlock Text="Datum:"/>
    <DatePicker SelectedDate="{Binding DateOfBirth}"/>
    <TextBlock Text="Leeftijd"/>
    <TextBox Text="{Binding Age, Mode=OneWay}"/>
</StackPanel>
</Window>

```

UpdateSourceTrigger

WPF zorgt ervoor dat de gegevens die een gebruiker in een Control intypt in de properties zet. Voor controls die bedoeld zijn de gebruiker gegevens te laten invoeren, gebeurt dat meestal op het moment dat de gebruiker naar een andere control toe gaat (ofwel op het moment dat het LostFocus event optreedt).

Soms wil je dat dit direct gebeurt. In dat geval kun je de standaard van WPF overrulen met de eigenschap `UpdateResourceTrigger`. Voorbeeld:

```
<StackPanel DataContext="{Binding Person}">
  <TextBlock Text="Voornaam:"/>
  <TextBox Text="{Binding Voornaam, UpdateSourceTrigger=PropertyChanged}" />
  <TextBlock Text="Achternaam:"/>
  <TextBox Text="{Binding Achternaam}" />

  <TextBlock Text="{Binding VolledigeNaam}" />
```

Hier zie je XAML waarmee je de gebruiker een voornaam en een achternaam laat intypen.

Er zijn 3 properties: Voornaam, Achternaam en VolledigeNaam. (VolledigeNaam is een read-only property dat afgeleid wordt uit de Voornaam en Achternaam: `($"{Voornaam} {Achternaam}"`);

Bij de voornaam heb je opgegeven dat de `UpdateSourceTrigger` *PropertyChanged* is. Dit betekent dat elke letter die de gebruiker intypt, direct in de property wordt bijgewerkt. Je ziet dan elke letter ook direct in de VolledigeNaam verschijnen.

Bij de achternaam heb je niets opgegeven: `UpdateSourceTrigger` is dan voor een `TextBox` *LostFocus*. Je ziet dan de VoorNaam pas wijzigen, nadat de gebruiker het `TextBox` voor de Voornaam verlaat (*LostFocus*).

Conclusie

Je hebt nu een goede kennis van Binding. Allereerst ken je het basisrecept. Daarnaast is nog eens extra ingegaan op het binden wanneer je met lijsten werkt; dat komt erg veel voor. Tenslotte is een aantal bijzonder situaties behandeld waarmee je in je opdrachten te maken kunt krijgen.

Hierna gaan we verder de met userstories van LosPollos. Kijk vooral nog eens terug in dit hoofdstuk, als iets niet meteen blijkt te werken.

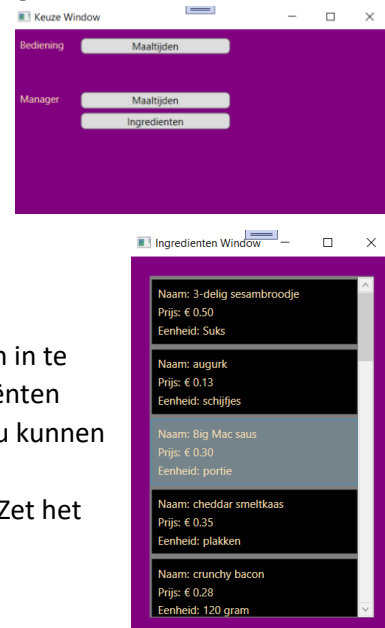
Opdracht 6: Userstory IngrediëntenInzicht

Als Manager wil ik een overzicht hebben van de ingrediënten die worden gebruikt voor het bereiden van een maaltijd. Het overzicht bevat:

- de naam van het ingrediënt,
- de prijs van het ingrediënt,
- de eenheid van het ingrediënt

Werk de userstory uit in de documentatie, database en programmatuur:

- Breid het Use Case diagram uit met use case *Ingrediënten tonen*. Deze use case wordt exclusief door de manager gebruikt.
 - Pas het datamodel (beschrijving) en ERD diagram aan met de ingrediënten. Voor de gegevens over een ingrediënt: zie het overzicht met ingrediënten in de bijlage.
 - Werk de database bij op basis van de door jou aangepaste documentatie
 - Maak in de folder Views, een KeuzeWindow waarop het personeel een keuze kan maken uit het Display Window en overzicht met ingrediënten. Zorg ervoor dat dit KeuzeWindow wordt getoond, zodra de manager of de bediening de applicatie start.
- Voeg aan de databaseclass een method toe om alle Ingrediënten in te lezen (GetIngredients). Zorg ervoor dat deze method de ingrediënten sorteert op hun naam (raadpleeg zo nodig internet hoe je dit zou kunnen doen).
 - Maak een Window met alle ingrediënten (IngredientsWindow). Zet het in de juiste folder.



Tip: Je kunt in alle Windows, behalve het KeuzeWindow, de volgende code toevoegen. Als je de gebruiker het window dan afsluit, keer je steeds terug naar het keuzewindow.

```
protected override void OnClosing(CancelEventArgs e)
{
    base.OnClosing(e);
    new KeuzeWindow().Show();
}
```

OnClosing is een method die in elk Window zit en die wordt uitgevoerd als het Window sluit. Dit gedrag kun je aanpassen door de method te "**overriden**".

In dit geval pas je het gedrag aan, door ervoor te zorgen dat niet alleen het Window wordt gesloten, maar dat ook het KeuzeWindow wordt getoond.

Protected is net als **public** en **private** een *access modifier*: je regelt ermee welke code OnClosing kan zien. Bij protected is dat alle code in de Window-class zelf en alle code die de Window class *extenden* (MainWindow : Window).

Normaliseren Ingrediënten: 1 op meer relaties

Kijk naar de lijst met ingrediënten met als peildatum 1-2-2022 (zie bijlage). Wat meteen opvalt is dat de eenheden waarin de ingrediënten geprijsd zijn, vol met spelfouten zitten en dat de eenheden ook nog eens verschillend genoteerd worden.

Je ziet dat wanneer je gegevens vaker dan 1 keer opslaat, dit vroeger of later tot problemen leidt:

- Inconsistentie, omdat gegevens op verschillende plaatsten er anders uitzien (stuks, Stuks, stks, etc).
- Extra werk, omdat de gebruiker bij wijzigen gegevens op meer plaatsen moet wijzigen.
- Kostbaar, omdat er onnodig veel geheugenruimte wordt gebruikt.

Deze problemen voorkom je door de gegevens maar 1 keer op te slaan. Dat doe je door ze in een aparte tabel te zetten en er met een **id** naar te verwijzen:

1. Je maakt dus een tabel voor de eenheden en zet hier alle eenheden in. Elke eenheid krijgt een id.
2. Je vervangt in de tabel met ingrediënten de omschrijving van de eenheid door de id van die eenheid in de tabel met eenheden.

Door de eenheden uit de tabel met ingrediënten te halen, heb je ervoor gezorgd dat elke eenheid precies 1 keer opgeschreven hoeft te worden. Je bent hier aan het normaliseren.

De tabellen zien er uiteindelijk als volgt uit:

Ingrediënten			
IngredientId	Naam	Prijs	EenheidId
1	Rundvlees burger	€ 3,23	1
2	Sla	€ 0,12	2
3	cheddar smeltkaas	€ 0.35	3
4	verse uitjes	€ 0.15	4
5	augurk	€ 0.13	5
6	Big Mac saus	€ 0.30	4
7	3-delig sesambroodje	€ 0.50	1
8	ketchup	€ 0.20	4
9	getoast sesambroodje	€ 0.45	1
10	geroosterd sesambroodje	€ 0.45	1
11	Mosterd	€ 0.25	6
12	ragout	€ 4.20	7
13	kipfilet	€ 4.15	1
14	sandwichsaus	€ 0.18	4
15	crunchy bacon	€ 0.28	8
16	emmentaler smeltkaas	€ 0.45	5
17	grillsaus	€ 0.35	4
18	tomaat	€ 0.65	5

Eenheden	
EenheidId	Naam
1	Stuks
2	Blaadjes
3	Plakken
4	Portie
5	Schijfjes
6	Kuipje
7	300 gram
8	120 gram

De eenheid stuks is nu nog maar 1 keer opgeslagen (id = 1). Vanuit de tabel ingrediënten verwijst je naar deze tabel. In de tabel Ingrediënten zie je bij de Rundvlees burger dat deze eenheid id 1 heeft. In de tabel Eenheden zie je dat hier de omschrijving Stuks bij hoort.

De activiteit om uit te zoeken op welke manier je gegevens zo efficiënt mogelijk kunt opslaan, wordt **normaliseren** genoemd. De mate waarin gegevens efficiënt worden opgeslagen, wordt uitgedrukt in normaalvormen. In de praktijk normaliseren programmeurs de gegevens naar de 3^{de} normaalvorm. Hiermee is bereikt dat:

- Gegevens in 1 of meer tabellen staan.
- Alle tabelrijen een sleutel (een unieke id) hebben.
- Voor elk gegevens er een aparte kolom is (voornaam, achternaam i.p.v. volledige naam)
- Er geen gegevens worden opgeslagen die je kunt afleiden/berekenen uit andere gegevens van de database (leeftijd kun je berekenen, volledige naam kun je afleiden, etc).
- Er zijn geen gegevens die altijd dezelfde waarde hebben (onnodig om op te slaan)
- Er staan geen lijsten in een kolom. Opsommingen (repeating groups) staan in een aparte tabel.
- Alle gegevens in een tabelrij alleen maar afhankelijk zijn van de sleutel (en dus niet afhankelijk van andere gegevens in de tabelrij)

Omdat de EenheidId uit de tabel met Ingrediënten een verwijzing bevat naar de tabel met Eenheden, zeggen we dat er een **relatie** tussen deze tabellen is. Deze relatie moet je in je datamodel beschrijven en in je ERD weergeven. Dat laatste doe je met 1 van de volgende lijnen die je tussen 2 tabellen tekent:

	<p>Deze lijn heeft 1 harkje. Het harkje zit bij de Student: dit betekent dat er in elke klas meer studenten kunnen zitten.</p> <p>Aan de kant van de klas zit geen harkje; dit betekent dat elke student in 1 klas zit. <i>Deze relaties komen veel voor.</i></p>
	<p>Dit is een lijn zonder harkjes. Dit betekent dat er sprake is van een 1-op-1 relatie. Bij elke Klant heb je precies 1 BrpPersoon (BasisRegistratiePersonen) en bij elke BrpPersoon heb je precies 1 klant. <i>Deze relaties komen niet zo veel voor.</i></p>
	<p>Deze lijn heeft 2 harkjes (meer). Een voetbalteam heeft 0, 1 of meer spelers, dus een harkje bij spelers. Maar een speler kan ook bij 0, 1 of meer teams spelen (geen club, of bij bijvoorbeeld PSV en het Nederlands Elftal). <i>Deze relaties komen zeer veel voor.</i></p>
<p>In bovenstaande voorbeelden staat een aantal relaties opgesomd. Vaak is een relatie optioneel. Bijvoorbeeld: Een BrpPersoon <i>kan</i> een klant zijn. In dat geval wordt er een 'bolletje' (0) aan het eind van de relatie getekend, dus:</p> <div data-bbox="635 622 960 689" data-label="Diagram"> </div> <p>Evenzo zie je vaak een dergelijk bolletje bij een harkje staan. Een klas kan uit 0, 1 of meer studenten bestaan.</p>	

Om terug te komen op het datamodel dat je moet bijwerken, zie je het volgende:

1. Bij elk *ingrediënt* heb je precies 1 *eenheid*.
2. Bij elke *eenheid* heb je 0, 1 of meer *ingrediënten*.

In de volgende opdracht ga je aan de slag om de documentatie en de database aan te passen aan dit genormaliseerde datamodel. Ook leer je in deze opdracht om de gegevens uit de database met een inner join in te lezen en te gebruiken in je programma.

Opdracht 7: Userstory IngrediëntEenheden

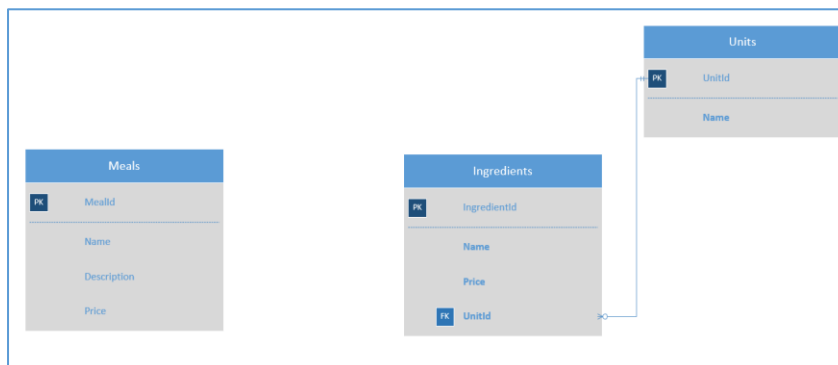
Doel van deze opdracht is te oefenen met het normaliseren van gegevens, het implementeren van relaties in een database en het lezen van gegevens uit gerelateerde tabellen.

- Pas het datamodel als volgt aan (let goed op dat je de relaties ook opschrijft; doe dit zodanig dat een programmeur begrijpt hoe hij de database moet maken en/of moet aanpassen.

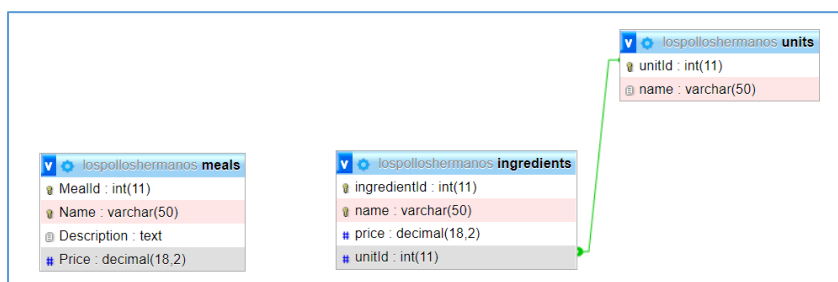
Table:	Ingredients		
Definition:	De maaltijden die het restaurant aan haar klanten aanbiedt		
Column	Type	Required	Remarks
ingredientId	int	Ja	Auto increment
name	varchar(50)	Ja	
price	decimal(18,2)	Ja	
unitId	int	Ja	FK
Index	Description		
ingredientId	Primary key		
name	Name moet uniek zijn		
Relations	Description		
Units	Bij elke ingredient hoort precies 1 units (unitId)		

Table:	Units		
Definition:	De eenheden waarin de ingrediënten geprijsd zijn		
Column	Type	Required	Remarks
unitId	int	Ja	Auto increment
name	varchar(50)	Ja	
Index	Description		
unitId	Primary key		
name	Name moet uniek zijn		
Relations	Description		
Ingredients	1 unit kan bij 0, 1 of meer ingredients gebruikt worden (unitId)		

- Pas het ERD aan



- Pas de database aan



- Maak de SQL query waarmee je de ingrediënten, gesorteert op naam, met hun eenheden, uit de database inleest. Hiervoor maak je gebruik van een inner join:

```
SELECT i.ingredientId, i.name, i.price, u.name as unitName2
FROM ingredients i1
INNER JOIN3 units u1 ON u.unitId = i.unitId
ORDER BY i.name
```

1. De letter **i** achter *ingredients* en de letter **u** achter *units* worden **aliassen** genoemd. Deze alias (de letters mag je zelf bedenken) kun je gebruiken in plaats van de volledige tabelnaam:
 - *i.name* is de naam van een ingredient
 - *u.name* is de naam van de eenheid
2. Achter *u.name* staat **as unitName**. Hiermee zorg je ervoor dat je niet dezelfde kolomnaam hebt voor de *name* in de *ingredients* tabel en de *name* uit de *units* tabel.
3. Met **INNER JOIN** combineer je gegevens uit 2 of meer tabellen. Met **ON** zorg je ervoor dat je de rijen uit de tabel *Ingredients* combineert met alleen de rijen uit de tabel *units* waarvoor geldt dat de unitId hetzelfde is.
Laat je **ON** weg, dan worden alle rijen met elkaar gecombineerd (probeer maar eens).

Opmerking. Er zijn meer vormen voor joins, kijk op internet (bijvoorbeeld W3Schools) wat de volgende joins doen:

- **INNER JOIN**
- **LEFT JOIN**
- **RIGHT JOIN**
- **FULL OUTER JOIN** (wordt niet ondersteund door MySql)

- Maak voor de rijen uit de databasetabel Units een class Unit (enkelvoud) aan
- Pas de class *Ingredient* aan, op basis van de gewijzigde Ingredient tabel. Neem ook een object op waarin je gegevens van de unit zet.

```
// Class bevat de gegevens van een ingredient met eenheid
public class Ingredient
{
    ...
    public decimal Price
    {
        get { return price; }
        set { price = value; }
    }

    private int unitId;
    public int UnitId2
    {
        get { return unitId; }
        set { unitId = value; }
    }

    public Unit Unit { get; set; }3
}
```

1. Je verwijdert de property voor de unit (name)
 2. Je voegt toe een property voor de id van de unit.
 3. Je voegt toe een property voor de rij uit de tabel Units
- De eigenschap unitId is nu eigenlijk 2 x in het Ingredient opgenomen: Ingredient.unitId en Ingredient.Unit.unitId. Deze aanpak wordt vaker gebruikt (bijvoorbeeld bij EF) en passen we daarom ook hier toe.
- NB: INotifyPropertyChanged is hier niet gemaakt; voeg dit zelf toe.

- Pas de databaseclass aan:
 - Wijzig de SQL in GetIngredients
 - Zorg dat de gegevens goed worden ingelezen

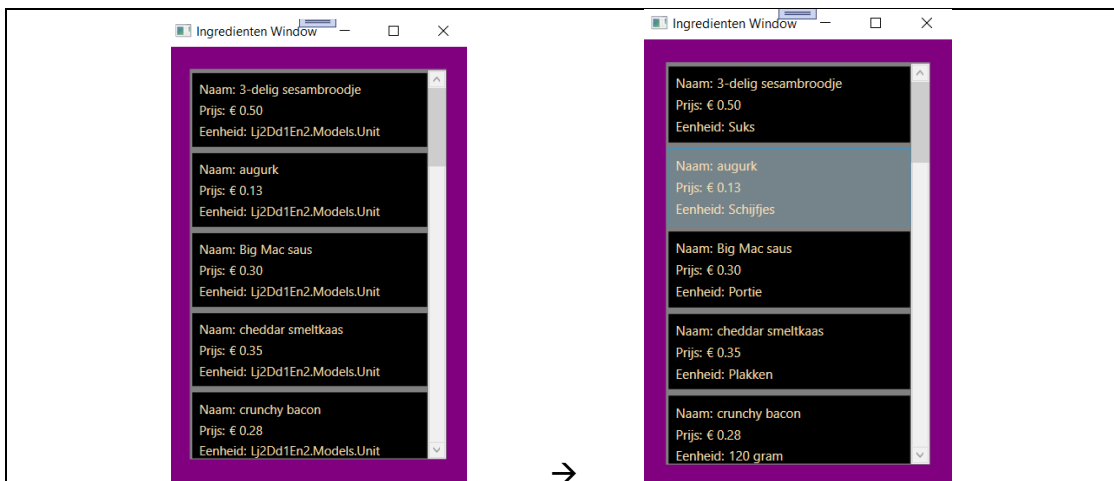
```

...
sql.CommandText = @"
    SELECT i.ingredientId, i.name, i.price, i.unitId, u.name as unitName
    FROM ingredients i
    INNER JOIN units u ON u.unitId = i.unitId
    ORDER BY i.name
";
...
Ingredient ingredient = new ()
{
    IngredientId = (int)reader["ingredientId"],
    Name = (string)reader["name"],
    Price = (decimal)reader["price"],
    UnitId = (int)reader["unitId"],
    Unit = new Unit()
    {
        UnitId = (int)reader["unitId"],
        Name = (string)reader["unitName"],
    }
};
ingredients.Add(ingredient);
...

```

1. Het SQL statement zorgt ervoor dat je de ingrediënten inleest uit de Ingredients tabel en met een *Inner join* gegevens de bijbehorende eenheden bijleest uit de tabel Units
2. In plaats van de unit, wordt nu de unitId gelezen
3. Ook wordt er een object voor alle Unitgegevens aan Ingredient gemaakt (new) en gevuld.

- Pas de XAML van je IngredientWindow aan, zodat de eenheden worden getoond (i.p.v. de naam van de Unit-class).



```

<ListView.ItemTemplate>
    <DataTemplate>
        <StackPanel Width="200" Margin="5">
            <DockPanel>
                <TextBlock Text="Naam:" Margin="2"/>
                <TextBlock Text="{Binding Name}" Margin="2"/>
            </DockPanel>
            <DockPanel>
                <TextBlock Text="Prijs:" Margin="2"/>
                <TextBlock Text="{Binding Price, StringFormat='€ 0.00'}" Margin="2"/>
            </DockPanel>
            <DockPanel>
                <TextBlock Text="Eenheid:" Margin="2"/>
                <TextBlock Text="{Binding Unit.Name}" Margin="2"/>
            </DockPanel>
        </StackPanel>
    </DataTemplate>
</ListView.ItemTemplate>

```

1. In de ItemTemplate codeer je op welke manier je een Ingrediënt uit de lijst met ingrediënten wilt tonen. Met een punt-notatie, kun je naar de gegevens van objecten verwijzen. Dus vanuit het ingredient noem je de naam van het unit-object (Unit) en daarbinnen de naam van de unit (Name). Je krijgt dan *Unit.Name*.

Opdracht 8: UserStory Beheer Ingrediënten

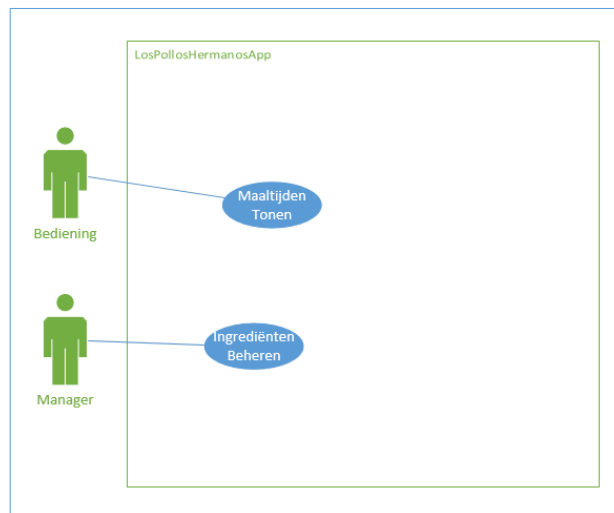
Inleiding

Je hebt nu geoefend met objecten, properties, binding en database classen. Je wilt natuurlijk ook de gegevens vastleggen in de database. Dit ga je in de volgende opdracht doen.

Als manager wil ik het assortiment ingrediënten kunnen uitbreiden. Ook wil ik bestaande ingrediënten kunnen aanpassen (de naam en de prijs).

Opdracht 8.1. UseCase

Pas het Use Case diagram aan, zodat helder is dat de manager niet alleen ingrediënten opvraagt, maar ze beheert.



Zorg ervoor dat het use case diagram een praat-plaatje blijft. Het is niet de bedoeling dat je er een compleet programma-flow-diagram van maakt.

Als een functie een volledige CRUD omvat, kun je dat het best in 1 use case weergeven.

Je maakt aparte use cases voor opvoeren, wijzigen, etc. als je geen volledige crud hebt, of als er iets bijzonders met een van de beheerfuncties is.

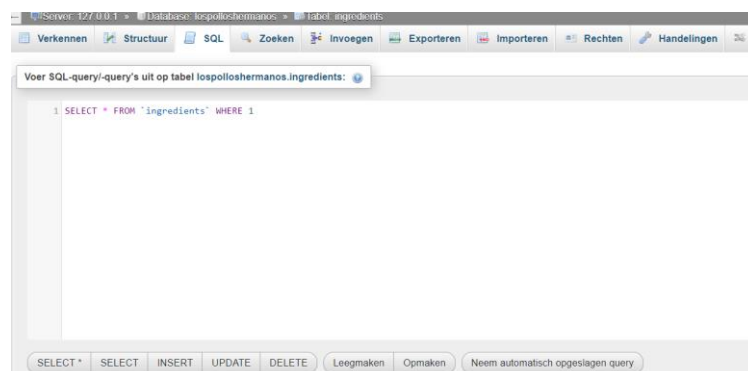
NB: Create, Read, Update, Delete en zijn allemaal beheerfuncties.

Opdracht 8.2. Databaseclass

In een databaseclass neem je alle beheerfuncties voor jouw tabellen op: de gegevens van elke tabel moet je kunnen toevoegen, verwijderen en wijzigen. Daarnaast moet je de gegevens natuurlijk ook kunnen raadplegen (lezen), waarbij je vaak ook gegevens uit andere tabellen bijleest met een JOIN.

Voor het toevoegen, wijzigen en verwijderen heeft SQL speciale statements. Je kunt hierover uitleg vinden op <https://www.w3schools.com/sql/>, of op <https://www.mysqltutorial.org/>.

Maar PhpMyAdmin kan je ook al helpen. Ga hiervoor naar PhpMyAdmin en kies de tab SQL. Onderaan zie je vervolgens knoppen voor SELECT *, SELECT, INSERT, UPDATE en DELETE.



We zullen nu methods in de databaseclass gaan maken voor het beheer van de gegevens van de databasetabel *Ingredients* en voor de databasetabel *Units*.

SQL	Method	Voorbeeld
SELECT	GetAll GetById	<code>SELECT i.ingredientId, i.name, i.price, i.unitId FROM ingredients i WHERE i.ingredientId = @ingredientId</code>
INSERT	Create	<code>INSERT INTO ingredients (ingredientId, name, price, unitId) VALUES (NULL,@name,@price,@unitId)</code>
UPDATE	Update	<code>UPDATE ingredients SET name = @name, price = @price, unitId = @unitId WHERE ingredientId = @ingredientId</code>
DELETE	Delete	<code>DELETE FROM ingredients WHERE ingredientId = @ingredientId</code>

```
// CreateIngredient voegt het ingredient object uit de parameter toe aan de database.
// Het ingredient object moet aan alle database eisen voldoen. De waarde van CreateIngredient:
// - "ok" als er geen fouten waren.
// - een foutmelding (de melding geeft aan wat er fout was)
public string CreateIngredient(Ingredient ingredient)
{
    if (ingredient == null || string.IsNullOrEmpty(ingredient.Name) ||
        ingredient.Price < 0 || ingredient.UnitId == 0)
    {
        throw new ArgumentException("Ongeldig argument bij gebruik van CreateIngredient");
    }

    string methodResult = UNKNOWN;

    using (MySQLConnection conn = new(connString))
    {
        try
        {
            conn.Open();
            MySqlCommand sql = conn.CreateCommand();
            sql.CommandText = @"
                INSERT INTO ingredients
                (ingredientId, name, price, unitId)
                VALUES (NULL, @name, @price, @unitId);
            ";
            sql.Parameters.AddWithValue("@name", ingredient.Name);
            sql.Parameters.AddWithValue("@price", ingredient.Price);
            sql.Parameters.AddWithValue("@unitId", ingredient.UnitId);

            if (sql.ExecuteNonQuery() == 1)
            {
                methodResult = OK;
            }
            else
            {
                methodResult = $"Ingrediënt {ingredient.Name} kon niet toegevoegd worden.";
            }
        }
        catch (Exception e)
        {
            Console.Error.WriteLine(nameof(CreateIngredient));
            Console.Error.WriteLine(e.Message);
            methodResult = e.Message;
        }
    }
    return methodResult;
}
```

1. Voordat je gegevens toevoegt, controleer je of ze aan de eisen voldoen. Dit hoort jouw Window class al te doen. Is dat niet gebeurd, dan geef je hier een *exception*.
2. Hier voeg je het SQL *INSERT-statement* toe. Dit statement zet je in een string (tussen 2 dubbele quotes: "). Hier is voor de eerste quote een @-teken gezet, waardoor je de string in meer regels kunt typen, wat de leesbaarheid van het SQL statement kan verhogen.
3. Als je variabele gegevens in een SQL statement wilt toevoegen, doe je dat uit veiligheidsoverwegingen altijd via Parameters. Zoals je ook al bij andere vakken hebt geleerd is mag je dit niet met een concatenatie doen.
4. ExecuteNonQuery geeft als resultaat het aantal rijen dat door het statement is "geraakt". Als het goed is, heb je nu 1 rij toegevoegd, dus moet het antwoord 1 zijn. Zo niet, is het fout


```

// UpdateIngredient wijzigt het ingredient met id ingredientId (parameter) met de gegevens uit
// de parameter ingredient. De gegevens van ingredient moeten aan alle database eisen voldoen.
// De waarde van UpdateIngredient:
// - "ok" als er geen fouten waren.
// - een foutmelding (de melding geeft aan wat er fout was)
public string UpdateIngredient(int ingredientId, Ingredient ingredient)
{
    if (ingredient == null || string.IsNullOrEmpty(ingredient.Name)
        || ingredient.Price < 0 || ingredient.UnitId == 0)
    {
        throw new ArgumentException("Ongeldig argument bij gebruik van UpdateIngredient");
    }

    string methodResult = UNKNOWN;

    using (MySQLConnection conn = new(connString))
    {
        try
        {
            conn.Open();
            MySqlCommand sql = conn.CreateCommand();
            sql.CommandText = @"
                UPDATE ingredients
                SET name = @name,
                    price = @price,
                    unitId = @unitId
                WHERE ingredientId = @ingredientId;
            ";
            sql.Parameters.AddWithValue("@ingredientId", ingredientId);
            sql.Parameters.AddWithValue("@name", ingredient.Name);
            sql.Parameters.AddWithValue("@price", ingredient.Price);
            sql.Parameters.AddWithValue("@unitId", ingredient.UnitId);

            if (sql.ExecuteNonQuery() == 1)
            {
                methodResult = OK;
            }
            else
            {
                methodResult = $"Ingrediënt {ingredient.Name} kon niet gewijzigd worden.";
            }
        }
        catch (Exception e)
        {
            Console.Error.WriteLine(nameof(UpdateIngredient));
            Console.Error.WriteLine(e.Message);
            methodResult = e.Message;
        }
    }

    return methodResult;
}

```

1. Hier voeg je het SQL UPDATE-statement toe

```

// DeleteIngredient verwijdert het ingredient met de id ingredientId uit de database. De waarde
// van DeleteIngredient :
// - "ok" als er geen fouten waren.
// - een foutmelding (de melding geeft aan wat er fout was)
public string DeleteIngredient(int ingredientId)
{
    string methodResult = UNKNOWN;

    using (MySQLConnection conn = new(connString))
    {
        try
        {
            conn.Open();
            MySqlCommand sql = conn.CreateCommand();
            sql.CommandText = @"
                DELETE
                FROM ingredients
                WHERE ingredientId = @ingredientId
            ";
            sql.Parameters.AddWithValue("@ingredientId", ingredientId);
            if (sql.ExecuteNonQuery() == 1)
            {
                methodResult = OK;
            }
            else
            {
                methodResult = $"Ingrediënt met id {ingredientId} kon niet verwijderd worden.";
            }
        }
        catch (Exception e)
        {

```

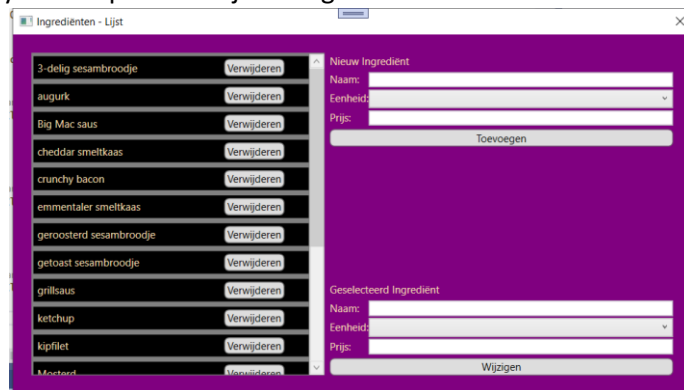
<pre> Console.Error.WriteLine(nameof(DeleteIngredient)); Console.Error.WriteLine(e.Message); methodResult = e.Message; } } return methodResult; } </pre>	
<p>1. Hier voeg je het SQL <i>DELETE-statement</i> toe</p> <pre> // GetIngredient leest 1 rij in uit de databasetabel Ingredients. Wordt er een rij gevonden, dan // worden de gegevens hiervan in de output parameter ingredient gezet. // Parameters: // - ingredientId : Id van het in te lezen ingredient // - ingredient(o) : null = niet gevonden // anders nieuw ingredient object met de database gegevens // De waarde van GetIngredient: // - "ok" als er geen fouten waren. // - een foutmelding, als er wel fouten ware public string GetIngredient(int ingredientId, out Ingredient? ingredient) { ingredient = null; string methodResult = UNKNOWN; using (MySQLConnection conn = new(connString)) { try { conn.Open(); MySqlCommand sql = conn.CreateCommand(); sql.CommandText = @" SELECT i.ingredientId, i.name, i.price, i.unitId, u.name as unitName FROM ingredients i INNER JOIN units u ON u.unitId = i.unitId WHERE i.ingredientId = @ingredientId; "; sql.Parameters.AddWithValue("@ingredientId", ingredientId); MySqlDataReader reader = sql.ExecuteReader(); while (reader.Read()) { ingredient = new() { IngredientId = (int)reader["ingredientId"], Name = (string)reader["name"], Price = (decimal)reader["price"], UnitId = (int)reader["unitId"], Unit = new Unit() { UnitId = (int)reader["unitId"], Name = (string)reader["unitName"], } }; } methodResult = ingredient == null ? NOTFOUND : OK; } catch (Exception e) { Console.Error.WriteLine(nameof(GetIngredient)); Console.Error.WriteLine(e.Message); methodResult = e.Message; } } return methodResult; } </pre>	<ol style="list-style-type: none"> 1. Omdat de parameter ingredient een <i>output</i>-parameter is (de method geeft deze parameter een waarde, zodat jouw Window class hem kan gebruiken), moet je er <i>out</i> voorzetten. 2. Omdat ingrediënt een <i>out</i> parameter is, moet je hem ook een initiële waarde geven, desnoods <i>null</i>. 3. Hier staat het SQL SELECT-statement om 1 ingrediënt in te lezen

Opdracht 8.4.beheer Units

- Maak op een vergelijkbare manier alle methods voor het beheer van de Units.

Opdracht 8.5. Window Ingredienten

- Maak de volgende lay-out en pas daarbij binding toe



- Zorg ervoor dat je in alle classes de `INotifyPropertyChanged` implementeert:
 - Het gaat om `Ingredient`, `Meal`, `Unit`
 - Voeg de juiste *usings* toe
 - Voeg achter de class naam : `INotifyPropertyChanged` toe
 - Voeg in de class de standaard code voor de implementatie van `INotifyPropertyChanged` toe
 - Zet in elke setter `OnNotifyPropertyChanged()`;
- De linkerlijst: deze is al gebind aan de `ObservableCollection<Ingredient>` property (`Ingredients`). Ook is het `SelectedItem` al gebind aan de `Ingredient` property `SelectedIngredient` (zo niet, doe dat alsnog)
- Rechtsboven: bind aan een property `NewIngredient`. Zorg dat je deze property met een new statement maak, anders wordt hij niet gebind
- Rechtsonder: bind aan de property `SelectedIngredient`, zodat wanneer de gebruiker op een ingredient van de lijst clickt, de gegevens automatisch rechtsonder worden getoond
- Bind allebei de comboboxen aan een `ObservableCollection<Unit>` property (naam *Units*)
- Bind de `SelectedItem` van de combobox rechtsboven aan `NewIngredientUnit` property
 - Zorg ervoor dat wanneer er een andere unit wordt geselecteerd (setter), dat dan de id van deze unit in de id van `NewIngredient.UnitId` wordt gezet
- Bind de `SelectedItem` van de combobox rechtsonder aan `ExistingIngredientUnit` property
 - Zorg ervoor dat wanneer er een andere unit wordt geselecteerd (setter), dat dan de id van deze unit in de id van `SelectedIngredient.UnitId` wordt gezet

Voeg functionaliteit toe:

- *SelectedIngredient*
 - Als er op een ingrediënt in de lijst wordt geklickt, worden de gegevens van het ingrediënt via Binding rechtsonder getoond, zodat ze gewijzigd kunnen worden
 - In de setter van *SelectedIngredient* voeg je code toe, waarmee de property *ExistingIngredientUnit* gelijk maakt aan de Unit uit Properties waarvan de *unitId* hetzelfde is als de *unitId* van het geselecteerde ingrediënt.

```
set
{
    selectedIngredient = value;
    // geen ingrediënt geselecteerd of Units property nog niet gevuld?
    if (value == null || Units == null)
    {
        // er is nog geen geselecteerde unit
        ExistingIngredientUnit = null;
    }
    else
    {
        // zoek de unit op waarvan de Unit Id gelijk is aan de Unit id van het
        // geselecteerde ingrediënt
        ExistingIngredientUnit = Units.FirstOrDefault(x => x.UnitId == value.UnitId);
    }
    OnPropertyChanged();
}
```

1. Met *FirstOrDefault* kun je in een lijst naar het eerste element zoeken dat aan een bepaalde conditie voldoet.
Hier wordt in de lijst met *Units* gezocht naar een *unit* waarvan de *unitId* gelijk is aan de *unitId* van het geselecteerde ingrediënt (*value.UnitId*).
De *x* is hier een placeholder. Je mag er ook een andere letter zetten, maar *x* is gebruikelijk.

- Knop Verwijder:
 - In de event handler zet je de sender om naar een Button
 - Van deze Button haal je vervolgens de DataContext op. Dit is het Ingredient van de regel waarin de Button staat
 - Verwijder het Ingredient
 - Werk het overzicht bij (*PopulateIngredients*)
- Knop Toevoegen
 - Controleer dat de invoer juist is
 - Voeg het ingredient toe aan de database
 - Werk het overzicht bij
 - Maak een nieuwe object voor *NewIngredient*
- Knop Wijzigen
 - Controleer dat de invoer juist is
 - Wijzig het ingredient in de database, de id van het Ingredient staat in *SelectedIngredient.IngredientId*
 - Werk het overzicht bij

Normaliseren DisplayWindow

Inleiding

Eerder heb je de tabel met ingrediënten uit de bijlage genormaliseerd. Hierbij zag je dat eenzelfde eenheid, bijvoorbeeld stuks, vaker opgenomen was in de tabel met ingrediënten. Door de eenheden in een aparte tabel te zetten, hoefde je ze nog maar 1 keer vast te leggen. Vanuit de tabel met ingrediënten verwees je naar de tabel met ingrediënten.

Je gaat nu in dit hoofdstuk leren hoe je de lijst met maaltijden moet normaliseren.

DisplayWindow

Als je kijkt naar de maaltijden (database tabel Meals) die in het DisplayWindow worden getoond, valt het volgende op:

- Bij Meals staan de ingrediënten ervan opgesomd. Pas je de naam van een ingrediënt aan dan moet de manager er voor zorgen dat hij ook de omschrijving bij de Meals aanpast. Evenzo: wijzigt de manager de eenheid van een ingredient (*kuipje* i.p.v. *300 gram*), dan zal hij eraan moeten denken dat hij ook de omschrijving bij de maaltijd aanpast.
- Bij de Meals staat ook een prijs ingevuld. Als de manager de prijs van een ingrediënt wijzigt, dan moet hij er ook voor zorgen dat de prijs van de maaltijden waarin het ingrediënt wordt gebruikt wordt aangepast.

Ook hier zie je dat gegevens op meer plaatsen zijn vastgelegd. Bovendien worden er ook afleidbare gegevens vastgelegd, zoals de prijs van een maaltijd. Dit betekent opnieuw dat er een grote kans is dat er fouten in de gegevens gaan ontstaan.

Ook hier valt veel winst te behalen als je de gegevens van de maaltijden gaat normaliseren. Je kijkt hiervoor naar de volgende 2 maaltijden: *Los Dos* en *El Grande*.

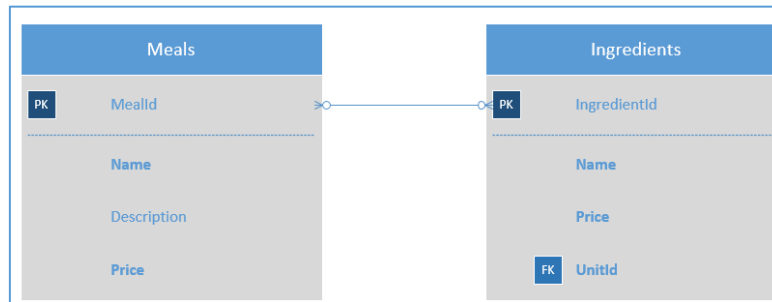
Van elk van de 2 maaltijden is de naam in de tabel gezet. Bij elke maaltijd staan vervolgens alle Ingrediënten die erin worden gebruikt. Bij elk van de gebruikte ingrediënten staat de prijs, het aantal eenheden van het ingrediënt en het bedrag (aantal x prijs). Ook staat er een Totaalprijs van alle ingrediëntbedragen bij elkaar opgeteld. Dit is de prijs van de maaltijd.

Maaltijd	Ingrediënt	Prijs	Aantal	Bedrag
Los Dos	Rundvlees burger	€ 3,23	2	€ 6,46
	Sla	€ 0,12	4	€ 0,48
	Verse uitjes	€ 0,15	2	€ 0,30
	Augurk	€ 0,13	5	€ 0,65
	Big Mac saus	€ 0,30	2	€ 0,60
	3-delig sesambroodje	€ 0,50	1	€ 0,50
Totaalprijs:				€ 8,99

Maaltijd	Ingrediënt	Prijs	Aantal	Bedrag
El Grande	Rundvlees burger	€ 3,23	1	€ 3,23
	cheddar smeltkaas	€ 0,35	2	€ 0,70
	Verse uitjes	€ 0,15	4	€ 0,60
	Augurk	€ 0,13	7	€ 0,91
	Ketchup	€ 0,20	3	€ 0,60
	Mosterd	€ 0,25	3	€ 0,75
	Getoast sesambroodje	€ 0,45	1	€ 0,45
Totaalprijs:				€ 7,24

Normaliseren is 'efficiënt' opslaan. De bedragen en het totaalbedrag sla je niet op: deze bedragen kun je immers uitrekenen. **In het algemeen geldt dat je nooit afleidbare gegevens in de database opslaat.**

Wat je hier ook ziet, is dat in elke maaltijd meer dan 1 ingrediënt kan zitten. Ook wordt een ingrediënt in meer maaltijden gebruikt (bijvoorbeeld de rundvlees burger). Je hebt dan dus een lijn tussen de tabellen Meals en Ingredients met aan allebei de kanten een 'harkje':



De relatie tussen Meals en Ingredients is een *meer-op-meer*-relatie (elke maaltijd heeft 0, 1 of meer ingrediënten en elk ingrediënt kan in 0, 1 of meer maaltijden gebruikt worden).

De vraag is nu, welke kolommen we in de tabellen kunnen opnemen voor deze relatie. In de tabel Ingredients staat al een kolom UnitId waarin je de relatie met de tabel Units opneemt: Elk *Ingredient* heeft precies 1 *Unit*.

Als je in de tabel Ingredients ook een kolom opneemt met een MealId, dan kun je hier ook maar 1 waarde in kwijt. Je moet dan kiezen uit de mealId van Los Dos, of die van El Grande:

Meals			Ingredients		
MealId	Name	IngredientId	IngredientId	Name	MealId
1	Los Dos	1, 2, 3	1	Rundvlees Burger	1, 2
2	El Grande	1, 2	2	Sla	1, 2
			3	Verse uitjes	1

Allebei de Id's kun je niet kwijt: in een kolom kun je maar 1 waarde zetten. Je maakt daarom een *KoppelTabel*. Deze tabel geef je een eigen unieke id en daarnaast zet je er de id's in van de maaltijd en het ingrediënt. Dit ziet er als volgt uit:

Meals		Ingredients	
MealId	Name	IngredientId	Name
1	Los Dos	1	Rundvlees Burger
2	El Grande	2	Sla
		3	Verse uitjes

MealIngredients		
MealIngredientId	MealId	IngredientId
1	1	1
2	1	2
3	1	3
4	2	1
5	2	3

Als je wilt weten uit welke ingrediënten *Los Dos* bestaat, zoek je de *MealId* op. Deze heeft de waarde 1. Deze waarde zoek je vervolgens op in de kolom *MealId* van de tabel *MealIngredients*: je vindt 3 rijen: een rij met *IngredientId* 1 (volgens de tabel *Ingredients* Rundvlees Burger), een rij met *IngredientId* 2 (Sla) en een rij met *IngredientId* 3 (Verse uitjes).

Andersom werkt ook: Verse Uitjes (*IngredientId* = 3) worden gebruikt in *MealId* 1 (*Los Dos*) en *MealId* 2 (*El Grande*)

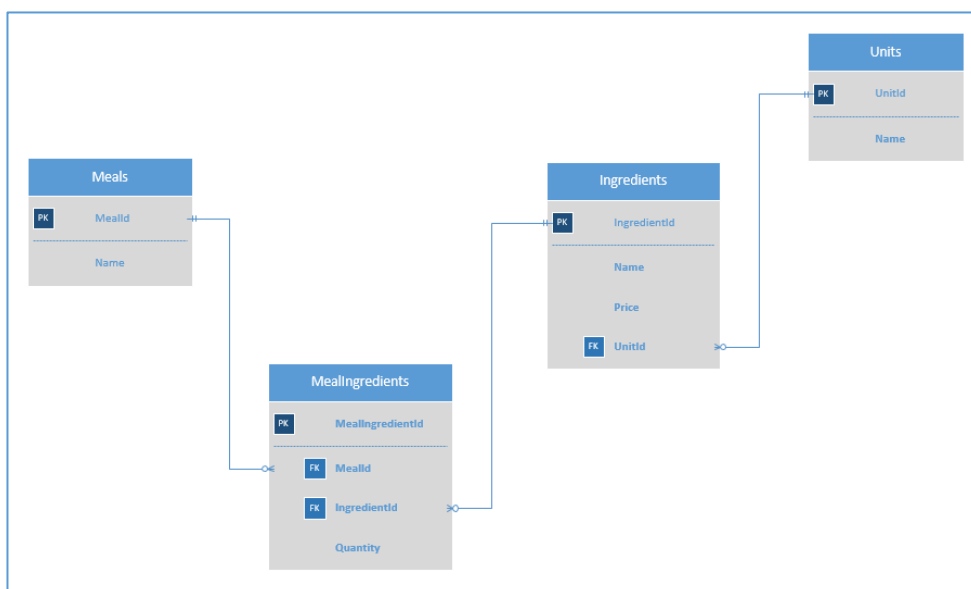
Je moet nu alleen nog een kolom toevoegen voor het aantal eenheden dat het ingrediënt in een maaltijd wordt gebruikt. Bereken de waarde van deze kolom in de tabel *MealIngredients* moet toevoegen.

Opdracht 9: Userstory Automatisch Display

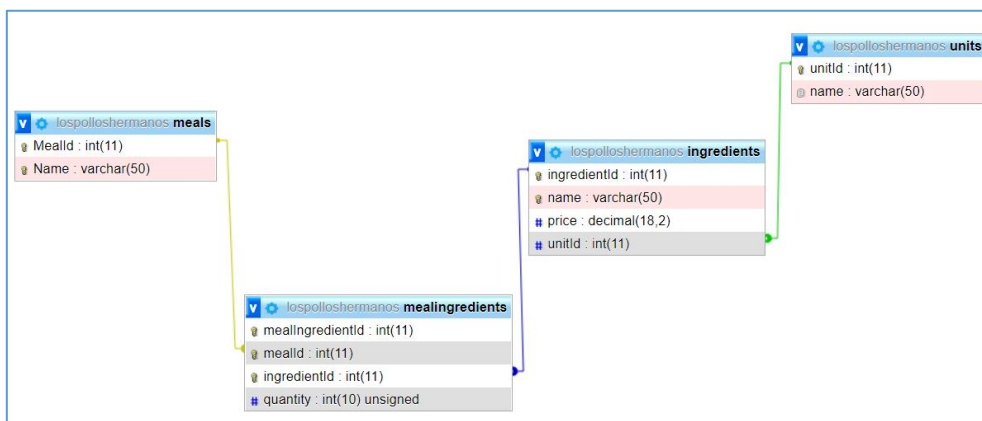
Als manager wil ik dat de omschrijving van de maaltijden en de prijzen van de maaltijden op het Display van in het restaurant, automatisch worden bijgewerkt op basis van de actuele samenstelling van maaltijden uit ingrediënten en prijzen.

Opdracht9.1.Datamodel

- Normaliseer de gegevens van de maaltijden op het display van het restaurant:
 - Zie hiervoor. Je hebt nu het volgende ERD



- Pas de database aan



- Pas de method GetMeals van de databaseclass aan, zodat de omschrijving wordt samengesteld en de prijs wordt berekend uit de gebruikte ingrediënten.

```
SELECT m.MealId, m.Name, mi.quantity, i.name as 'IngredientName', i.price, u.name as 'UnitName'
FROM meals m
LEFT JOIN mealingredients mi ON m.MealId = mi.mealId
LEFT JOIN ingredients i ON mi.ingredientId = i.ingredientId
LEFT JOIN units u ON i.unitId = u.unitId
```

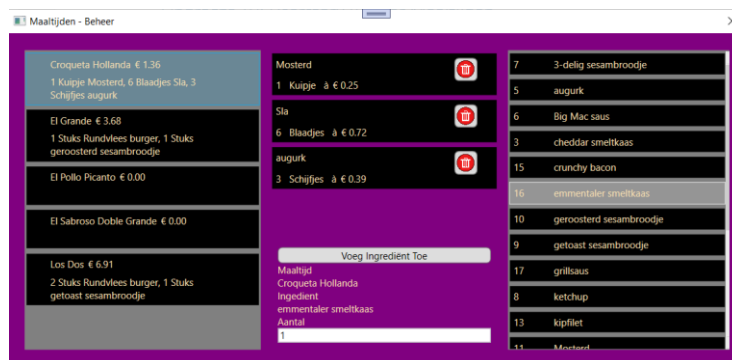
- Je selecteert alle *Meals* en leest daar de *MealIngredients* bij.
- Merk op dat je hier en left join gebruikt. Dat betekent dat je ook de Meals selecteert waarvoor geen mealingredient in de tabel *MealIngredients* bestaat.

Opdracht 10: Userstory Beheer Maaltijden

Als manager wil ik zelf maaltijden kunnen beheren: toevoegen, wijzigen, verwijderen. Hierbij wil ik de mogelijkheid hebben om de maaltijden met een aantal ingrediënten te kunnen samenstellen.

Realiseer de user story:

- Pas het use case diagram aan
- Pas het KeuzeWindow aan, zodat de manager ervoor kan kiezen om maaltijden te beheren
- Maak het BeheerWindow voor de maaltijden, waarbij je binding toepast
 - Layout:



Use cases

Inleiding

Een van deze technieken is het Use Case diagram. Deze techniek wordt gebruikt wanneer je wilt beschrijven *wie wat* met de applicatie kan doen. Het Use Case diagram is dus bedoeld om de *functionaliteiten* van een applicatie te beschrijven en tref je daarom ook vaak aan in het functioneel ontwerp. Het is bedoeld voor de gebruiker, zodat deze weet wat het systeem voor hem zal gaan doen, maar ook voor de programmeur, zodat deze weet wat hij precies moet maken.

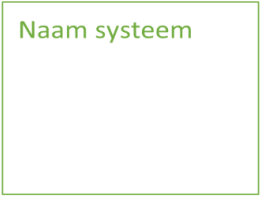

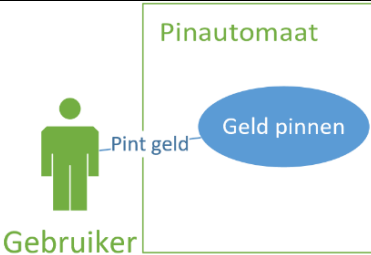
De Use Case diagram is een opgemaakt in UML (Unified Modeling Language). In dit hoofdstuk leer je hoe je een Use Case diagram volgens deze regels maakt.

Achtereenvolgens wordt ingegaan op:

1. Het use case diagram (het schema dat je ziet),
2. De beschrijving bij het diagram, wat altijd bij het schema hoort.

Use Case Diagram

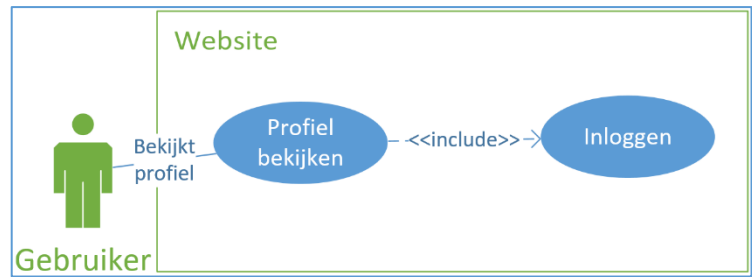
In een use case diagram word aangegeven wie wat kan doen. Dit wil dus niets meer zeggen dan: hoe reageert het systeem op bepaalde verzoeken afkomstig van buiten het systeem (mens-machinegrens). Bij een use case diagram hebben we dus te maken met twee aspecten, namelijk: **Wie** en **Wat**.

Systeem: Een use case diagram geeft aan hoe een systeem reageert op bepaalde verzoeken afkomstig van buiten het systeem. Hierdoor is het in een diagram dus erg belangrijk om aan te geven wat bij het systeem hoort en wat niet. Dit kan aangegeven worden door een rechthoek met de naam van het systeem erin.	
Wie: Wie iets doet bij een systeem heet in de terminologie van use case diagram een Actor . Het gaat hierbij dus niet om een persoon maar om een rol die iemand speelt: bijvoorbeeld administrator of verkoper. Omdat de Actor een invloed is van buiten het systeem, hoort dit symbool ook buiten het systeem te staan in het diagram. De actor wordt aangegeven door het volgende symbool:	
Wat: Een use case is wat een actor met het systeem kan doen. Bijvoorbeeld: geld pinnen. Een use case kan op twee manieren aangeroepen worden: via een actor en via een andere use case. Dit resulteert dus ook in twee verschillende soorten verbindingen: een doorgetrokken lijn en een stippellijn (zie hierna). De doorgetrokken lijn wordt gebruikt als de use case aangeroepen wordt door een Actor. De use case wordt aangegeven door een ovaal. De use case geeft hier dus aan dat met het systeem geld gepind kan worden.	

Een **stippellijn** is een verbintenis tussen twee Use Cases. Hier wordt opnieuw onderscheid gemaakt in twee soorten, namelijk: Include en Extend.

Een **Include** is een uitbreiding die dient om redundantie te voorkomen. Dit is dus **geen optionele** uitbreiding.

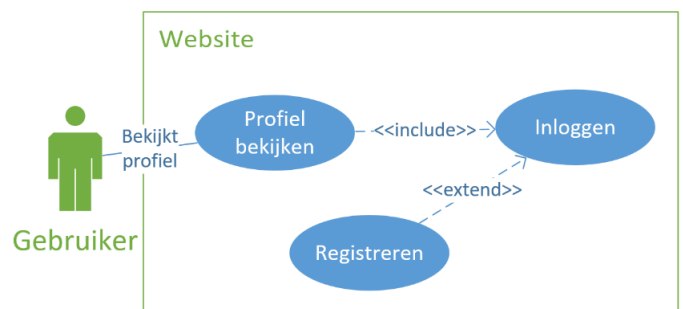
Bijvoorbeeld voor het bekijken van het profiel van een gebruiker op een website, moet de gebruiker eerst **verplicht** inloggen.



Een **Extend** is wel een optionele uitbreiding. Deze wordt ook met een stippellijn aangegeven, maar ziet er net iets anders uit dan de include.

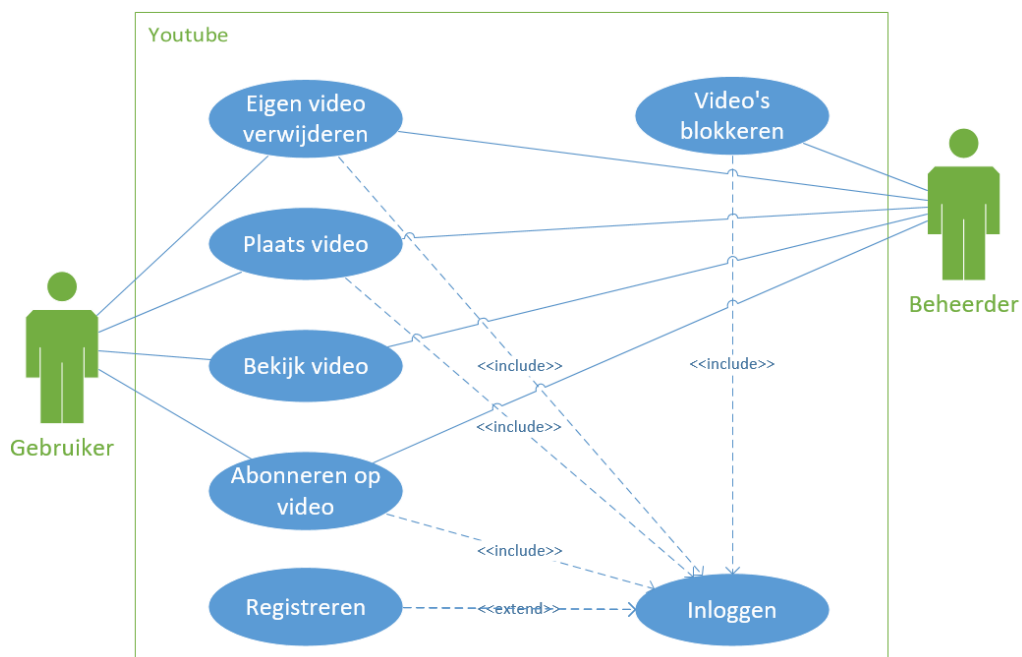
Let op de richting van de pijl bij de include en extend! Deze is namelijk andersom. Deze pijl is in de leesrichting.

De use case “Profiel bekijken” **bevat de** use cases “Inloggen” en “Registreren”. De use case “Registreren” **is een optionele uitbreiding op** de use case “Inloggen”.



Voorbeeld met uitleg

Een voorbeeld voor de welbekende site Youtube:



Toelichting: Bij YouTube heb je meerdere *actors* die het systeem kunnen gebruiken, bijvoorbeeld: een gebruiker en een beheerder. Een beheerder mag in dit geval meer acties op YouTube uitvoeren dan een gewone gebruiker, bijvoorbeeld het kunnen blokkeren van video's.

Verder is het voor bepaalde Use Cases verplicht om ingelogd te zijn. Hierdoor zijn de lijnen voor het inloggen een stippellijn en van het type <<include>>. Dit aangezien je redundantie wil voorkomen en inloggen een verplichte uitbreiding is.

De Use Case Registreren wordt alleen uitgevoerd indien de gebruiker bij het inloggen nog geen account heeft. Dit is dus een optionele uitbereiding en is dus met een stippellijn van het type <<extends> verbonden aan de Use Case inloggen.

De heer Pietersen

Pietersen werkt als beheerder bij YouTube. Privé vindt hij het leuk om films van YouTube te bekijken. De heer Pietersen heeft dus 2 rollen: die van beheerder en die van gebruiker.

Als beheerder kan hij films bekijken en blokkeren.

Als gebruiker kan hij net als alle andere gebruikers, ook films bekijken, maar niet blokkeren.

Use Case beschrijving

Wat er precies in een specifieke use case gebeurt, wordt beschreven in de *use case description*. Net als het diagram, zijn er ook regels voor de beschrijving: de beschrijving ziet er als volgt uit:

Naam	<< naam van de use case >>
Actoren	<< opsomming van actoren die op verschillende manieren de use case gebruiken>>
Aannamen	<< aannamen die je doet voor de gegeven use case >>
Beschrijving	<< stappenplan van welke actor wat doet in de use case in chronologische volgorde >>
Uitzondering	<< mogelijke uitzonderingen waar deze use case beschrijving niet voor geldt >>
Resultaat	<< het resultaat van deze use case >> <i>Dit kan een happy path zijn, maar hoeft niet. Met een happy path wordt bedoeld de gewenste uitkomst.</i> <i>Bijvoorbeeld: de use case "inloggen" kan als resultaat hebben dat een persoon is ingelogd (happy path) of dat het account geblokkeerd is na 3x foutief inloggen (dit resulteert in twee use case beschrijvingen).</i>

Hieronder zie je een voorbeeld van een **use case description** "Bekijk video" die hoort bij het **use case diagram** YouTube:

Naam	Bekijk video
Actoren	Gebruiker (beheerder hoeft hier dus niet bij aangezien de beheerder op geen andere manier deze use case gebruikt)
Aannamen	<ul style="list-style-type: none"> - De actor heeft een internetverbinding. - De actor heeft een browser of de Youtube app ter beschikking.
Beschrijving	<ol style="list-style-type: none"> 1. De gebruiker gaat naar www.youtube.nl 2. Het systeem toont de pagina. 3. De gebruiker typt een titel in het zoekveld in

	4. De gebruiker drukt op de Enter toets of klikt op het zoek icoon. 5. Het systeem zoekt alle video's gerelateerd aan de zoekterm. 6. De gebruiker klikt de gewenste video aan. 7. Het systeem speelt de video af, als deze niet is geblokkeerd.
Uitzondering	Als de gewenste video geblokkeerd is, wordt hiervan een melding waarna wordt vervolgd met stap 2.
Resultaat	Gebruiker bekijkt de gewenste video.

Conclusie

In dit hoofdstuk zijn we ingegaan op de techniek van het use case diagram. Het is een UML-techniek die gebruikt wordt in het functioneel ontwerp. Een use case bestaat uit een diagram en een beschrijving. Bij het uitwerken van het use case diagram kun je functionaliteiten efficiënt benoemen door middel van optionele extend- en verplichten include relaties. Hiermee voorkom je redundantie en zorg je er ook nog eens voor dat functionaliteiten (use cases) in verschillende situaties hetzelfde zijn.

Use cases zijn bedoeld voor zowel de gebruiker om af te stemmen wat het systeem voor ze gaat doen als voor de ontwikkelaars/programmeurs/testers, die de use cases gebruiken om de functionaliteit te maken en te testen.

Meer informatie

- <https://www.ivarjacobson.com/publications/white-papers/use-case-20-ebook-dutch-version>

Opdracht 11: Login

De eigenaar van Los Pollos Hermanos wil dat de applicatie binnen zijn het restaurant ook beschikbaar wordt voor zijn gasten:

- Alle gasten krijgen toegang tot het Display; ze kunnen daar in scrollen.
- Alle medewerkers van het bedrijf krijgen dezelfde rechten als de gasten en kunnen bovendien een lijst met ingrediënten die het bedrijf gebruikt, inzien.
- De eigenaar heeft dan weer dezelfde rechten als de medewerkers maar kan ook ingrediënten en maaltijden beheren.

Gevraagd:

- Maak een use case diagram met beschrijving en stem dat met de eigenaar van Los Pollos Hermanos af.

NB: Je hoeft het inloggen niet te maken.

Conclusie

Je hebt nu alle kennis in huis om:

- Use case diagrams te gebruiken en te maken
- Datamodellen en ERD-diagrammen te gebruiken en te maken
- Databases handmatig in te richten op basis van een datamodel
- Binding toe te passen
- Interfaces te gebruiken (en te maken)
- Classen te maken en gebruiken

Wellicht denk je bij het uitwerken van de CRUD-functies in de database: *volgens mij ben ik dingen dubbel aan het doen*. Dat is gedeeltelijk waar: de CRUD-functies lijken heel erg veel op elkaar, behalve dan het SQL-statement en het omzetten van de ingelezen gegevens in een strong typed object. Hierna ga je kijken op welke manier je dit kunt optimaliseren.

Optimalisatie (EXTRA STOF)

Inleiding

In de databaseclass staat erg veel code die veel op elkaar lijkt. Dat lijkt niet erg, want je kunt gemakkelijk een kopietje maken en dat vervolgens aanpassen. Problemen ontstaan, zodra je achteraf de code moet aanpassen omdat inzichten gewijzigd zijn. Je moet dan dezelfde aanpassingen vaker intypen, wat niet alleen tijdrovend is, maar ook een grote kans op fouten heeft.

Kan het allemaal niet wat efficiënter? In Leerjaar 1 heb je daar wellicht al wel eens naar gekeken. Je gaat nu ook de code optimaliseren, door voor de verschillende SQL-statements een generieke method te maken.

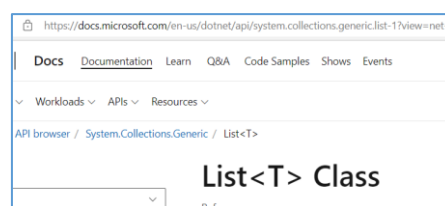
Method	SQL-Statement	Specifiek deel	Output
GetAll	SELECT	<ul style="list-style-type: none">• SQL-Statement• Omzetten database row naar een object van een specifieke class	<ul style="list-style-type: none">• ICollection met objecten van de specifieke class• Resultaat
GetById	SELECT	<ul style="list-style-type: none">• SQL-Statement• Omzetten database row naar een object van een specifieke class	<ul style="list-style-type: none">• Object van de specifieke class• Resultaat
CRUD	CREATE UPDATE DELETE	<ul style="list-style-type: none">• SQL-Statement	<ul style="list-style-type: none">• Resultaat

Deze methods mag je in het vervolg in al jouw programma's gebruiken. Zet de code dus ergens neer waar je hem goed kunt terugvinden.

Generics en Delegates

In deze module is het resultaat van een GetAll method niet langer een DataTable maar een ICollection met objecten van een bepaalde Class. Dit betekent dat je de gegevens uit de *DataReader* omzet naar objecten van die bepaalde class.

Omdat je niet weet om welke class het gaat maak je gebruik van een zogenaamde *Generic*. Dit is een *placeholder* die bij gebruik wordt omgezet in het *echte* type is. Je kent deze constructie met generics al bij bijvoorbeeld een List. Ook een List kent een generic (zoek maar op in de documentatie):



Wil je een List gebruiken, dan moet je daarbij ook het type opgeven. Bijvoorbeeld:

```
List<string> namen = new();
```

Nu je voor het clas type een oplossing hebt, moet je nog een oplossing hebben om de gegevens van een DataReader om te zetten naar de gegevens van de class. Hiervoor heb je een callback method

nodig, die als argument de `MySqlDataReader` heeft (daar staan de gegevens in, die omgezet moeten worden) en als resultaat een object teruggeeft van het specifieke type class.

Om deze callback method als argument in een method te kunnen gebruiken, maak je hiervoor een delegate:

```
// Call back method om een Reader-row in een nieuw object van de class T te zetten:  
private delegate T EntityToClass<T>(MySqlDataReader reader);
```

T: placeholder voor de generic. Het echte type geef je bij gebruik op (zie hierna)

delegate: hier beschrijf je hoe de callback method eruit ziet, zodat je deze als argument aan een method kunt meegeven.

SQL Select GetAll

Neem de volgende code over en lees de toelichting nauwkeurig na.

```
private string ReadObjects<T>¹(ICollection<T>² objectList, string sqlCommand, EntityToClass<T>³ entityToClass)  
{  
    return ReadObjects²(objectList, sqlCommand, null, entityToClass);  
}
```

1. Hier geeft je aan dat je een *generic* gebruikt. Het echte type geef je bij gebruik aan (zie hierna).
2. Er is een `ReadObject` gemaakt voor het geval je SQL-parameters wilt doorgeven (4 parameters) en voor het geval je dat niet wilt (3 parameters). Hier wordt de `ReadObjects` method gebruikt met 4 argumenten. Als 3^{de} argument vul je `null` in. Hierdoor wordt de method die hieronder uitgewerkt is, uitgevoerd.

Als je in 1 class verschillende methods dezelfde naam geeft, heet dat *overloaden*. C# voert de code uit van de method waarvan de parameters overeenkomen met de argumenten. *Overloaden* wordt veel gebruik bij OO programmeren.

```
private string ReadObjects¹<T>(ICollection<T> objectList, string sqlCommand,  
    MySqlParameter[]? sqlParameters, EntityToClass<T> entityToClass³)  
{  
    if (objectList == null)  
    {  
        throw new ArgumentException(ILLEGALARGUMENT);  
    }  
    string methodResult = "";  
  
    using (MySqlConnection conn = new(connString))  
    {  
        try  
        {  
            conn.Open();  
            MySqlCommand sql = conn.CreateCommand();  
            sql.CommandText = sqlCommand;  
            if (sqlParameters != null)²  
            {  
                sql.Parameters.AddRange(sqlParameters);  
            }  
            MySqlDataReader reader = sql.ExecuteReader();  
  
            while (reader.Read())  
            {  
                objectList.Add(entityToClass³(reader));  
            }  
            methodResult = OK;  
        }  
        catch (Exception e)  
        {  
            Console.Error.WriteLine(nameof(ReadObjects));  
            Console.Error.WriteLine(sqlCommand);  
            Console.Error.WriteLine(e.Message);  
            methodResult = e.Message;  
        }  
    }  
    return methodResult != "" ? methodResult : OK;  
}
```

1. Dit is de method met 4 parameters. De extra parameter is een array met MySqlParameters. Hier zet je alle parameters in voor de SQL statement in (dus de @mealId, etc)
2. De parameters worden hier aan het Sql commando toegevoegd.
3. De entityToClass is de callback method die ervoor zorgt dat de gegevens uit de Reader omgezet worden naar een object van het type <T>. Het type van de is de delegate die je hiervoor hebt gemaakt.

Voorbeeld gebruik

```
public string GetUnits(ICollection<Unit> units)
{
    string sqlCommand = @"
        SELECT u.unitId, u.name
        FROM units u
        ORDER BY u.name
    ";

    return ReadObjects(units, sqlCommand,
        reader => new Unit()
        {
            UnitId = (int)reader["unitId"],
            Name = (string)reader["name"],
        });
}
```

Hier zie je een voorbeeld hoe je de algemene methode gebruikt. Dat is best even wennen, dus veel oefenen!

1. Je maakt het SQL commando
2. In de return geeft je de volgende argumenten mee:
 - a. De ICollection voor de units
 - b. Het SQL commando
 - c. De call back method. Deze wordt hier inline gecodeerd. Het is gebruikelijk om dat zo te doen. De method schrijf je als een *lambda expressie* (arrow-notatie):
 - i. reader is een placeholder voor het argumnt dat aan de method wordt meegegeven. In de delegate zie je dat het type van deze placeholder *MySqlDataReader* is.
 - ii. => is de arrow die er moet staan
 - iii. Achter de arrow zet je wat de method moet doen. In dit geval een nieuw Unit object maken en de properties vullen met de gegevens uit de reader.

SQL Select GetById

```
private string ReadObject<T>(string sqlCommand, out T t, EntityToClass<T> entityToClass,
    MySqlParameter[]? sqlParameters = null)
{
    string methodResult = "";
    t = default(T); // Instantieer t op null; null mag niet

    using (MySqlConnection conn = new(connString))
    {
        try
        {
            conn.Open();
            MySqlCommand sql = conn.CreateCommand();
            sql.CommandText = sqlCommand;
            if (sqlParameters != null)
            {
                sql.Parameters.AddRange(sqlParameters);
            }
            MySqlDataReader reader = sql.ExecuteReader();

            if (reader.Read())
            {
                t = entityToClass(reader);
            }
            methodResult = OK;
        }
        catch (Exception e)
        {
            Console.Error.WriteLine(nameof(ReadObjects));
            Console.Error.WriteLine(sqlCommand);
            Console.Error.WriteLine(e.Message);
            methodResult = e.Message;
        }
    }
    return methodResult != "" ? methodResult : OK;
}
```



```
}
```

Deze code lijkt erg veel op de code voor GetAll. Omdat je hier altijd een parameter moet meegeven, is een overload niet nodig. De callback method wordt hier gebruikt om de output parameter *t* een waarde te geven.

1. Met *default* geef je aan dat de default waarde aan de variabele *t* moet worden toegekend; voor objecten is dat null.

SQL CreateUpdateOrDelete

```
private string CreateUpdateOrDeleteObject(string sqlCommand, MySqlParameter[]? sqlParameters = null)
{
    string methodResult = "";

    using (MySqlConnection conn = new(connString))
    {
        try
        {
            conn.Open();
            MySqlCommand sql = conn.CreateCommand();
            sql.CommandText = sqlCommand;
            sql.Parameters.AddRange(sqlParameters);

            if (sql.ExecuteNonQuery() == 1)
            {
                methodResult = OK;
            }
            else
            {
                methodResult = UPDATERROR;
            }
        }
        catch (Exception e)
        {
            Console.Error.WriteLine(nameof(ReadObjects));
            Console.Error.WriteLine(sqlCommand);
            Console.Error.WriteLine(e.Message);
            methodResult = e.Message;
        }
    }
    return methodResult != "" ? methodResult : OK;
}
```

Deze code zal weinig verrassingen kennen. Merk op dat er hier voor is gekozen om de SQL-parameters als laatste argument mee te geven. In dit geval mag je aan deze parameter een default waarde toekennen; hier *null*. Dit betekent dat je de method met én zonder argument voor parameters mag gebruiken. Hierdoor heb je de *overload* evenmin nodig.

Eindopdracht Desktop Development 1

Opdracht

Mocht je binnenkort op vakantie gaan dan heb je vast gekozen voor één van jouw favoriete vakantie landen. Maar niet iedereen heeft dezelfde favoriete landen. Maak daarom een WPF-applicatie waarbij het mogelijk is om per persoon aan te geven welke landen favoriet zijn.

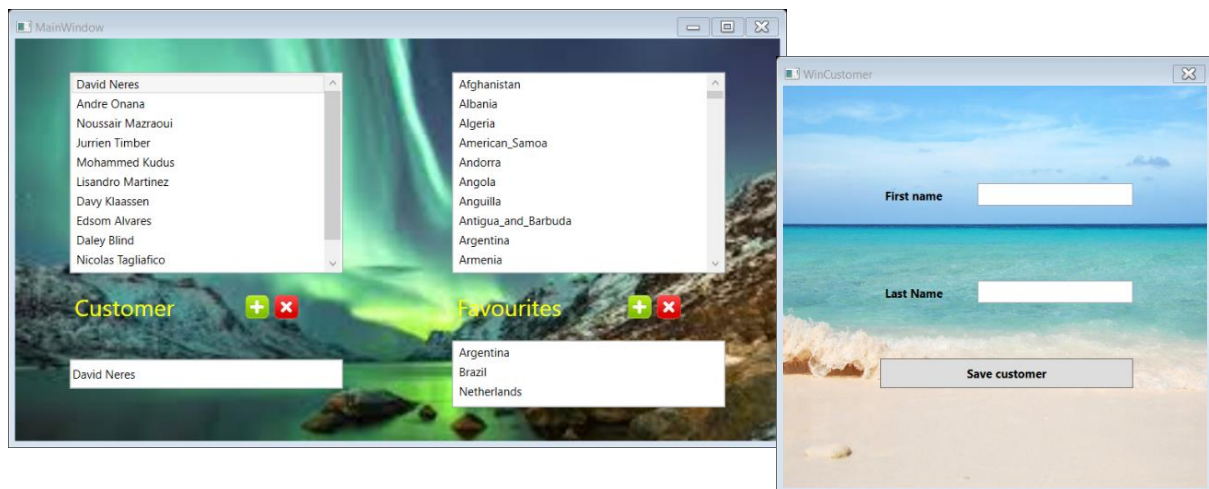
Aanpak

Bestudeer de beoordelingsmatrix. Maak voor jezelf een overzicht wat je allemaal moet doen en zet dit op volgorde in een todo-lijst (zie hieronder). Zet bij elk onderwerp een indicatie hoelang je ermee bezig zult zijn. Ben je langer dan 2 uur met een onderwerp bezig: splits het onderwerp dan op in kleinere delen, en zet bij deze kleinere delen een tijdsindicatie. Zet tenslotte bij elk onderwerp een datum en tijdstip waarop je met het onderwerp begint en wanneer je ermee klaar denkt te zijn. Geef bij elk onderwerp de status aan.

ToDo – Eindopdracht DD1					
Nr	Onderwerp	Tijd	Start	Eind	Status
1	Use case maken	0:30	8-4-2022 9:30u	8-4-2022 10:00u	Gereed
2	Bezig
...
...	Te doen

Eisen: Zie beoordelingsmatrix

Voorbeeld schermen:



Alternatief

- Heb jij een beter idee? Lever dan een schets in, wat jij wilt gaan maken (globaal ontwerp van de schermen en de gegevens die de applicatie gebruikt). Zorg ervoor dat jouw applicatie met een database werkt. Het datamodel moet tenminste 2 tabellen hebben met een 1-op-meer relatie.
- Nadat jouw schets is goedgekeurd, ga je met je eigen applicatie aan de slag.

Beoordelingsmatrix

Onderwerp	1	2	3	4
Naamgeving / Naming conventions / Programma Layout	<input type="checkbox"/> Namen van variabelen, methods en classes zijn onvoldoende goed toegepast. Naming conventions zijn onvoldoende toegepast. Programma layout is slordig.	<input type="checkbox"/> Namen van variabelen, methods en classes zijn regelmatig goed toegepast. Naming conventions zijn regelmatig goed toegepast. Programma layout is matig.	<input type="checkbox"/> Namen van variabelen, methods en classes zijn vaak goed toegepast. Naming conventions zijn vaak goed toegepast. Programma layout is matig.	<input type="checkbox"/> Namen van variabelen, methods en classes zijn goed toegepast. Naming conventions zijn goed toegepast. Programma layout is goed.
Commentaar	<input type="checkbox"/> Commentaar is nauwelijks van toepassing of niet relevant	<input type="checkbox"/> Commentaar ontbreekt vaak of is onduidelijk / niet relevant.	<input type="checkbox"/> Bijna alle methods en classes zijn voorzien van goed commentaar.	<input type="checkbox"/> Alle methods en classes zijn voorzien van goed commentaar.
Foutafhandeling Juiste/ code juiste plaats / Gegevenstypen*	<input type="checkbox"/> Er is nauwelijks aandacht aan foutafhandeling besteed. Er is geen gebruik gemaakt van een database class. Voor gegevens worden niet de juiste gegevenstypen gekozen.	<input type="checkbox"/> Foutafhandeling is in redelijke mate aanwezig; meldingen geven onvoldoende instructie. Er is gebruik gemaakt van een database class. Voor gegevens worden meestal de juiste gegevenstypen gekozen.	<input type="checkbox"/> Foutafhandeling is goed aanwezig; Programma crasht in sommige gevallen. Er is gebruik gemaakt van een database class. Voor gegevens worden de juiste gegevenstypen gekozen.	<input type="checkbox"/> Foutafhandeling is goed aanwezig; Programma crasht niet. Er is gebruik gemaakt van een database class, zonder gebruikersinterface elementen. Voor gegevens worden de juiste gegevenstypen gekozen.
Use case Diagram*	<input type="checkbox"/> Er is geen Use Case diagram gemaakt.	<input type="checkbox"/> Er is een Use Case diagram gemaakt, maar belangrijke use cases of actoren ontbreken.	<input type="checkbox"/> Er is een Use Case diagram dat de actoren en use cases van de applicatie grotendeels correct bevat.	<input type="checkbox"/> Er is een Use Case diagram dat de actoren en use cases van de applicatie correct bevat.
Datamodel / ERD Diagram /Database Script*	<input type="checkbox"/> Er is geen Datamodel gemaakt of ERD diagram gemaakt.	<input type="checkbox"/> Er is een Datamodel en of ERD diagram. Er is echter geen script aanwezig.	<input type="checkbox"/> Er is een Datamodel en of ERD diagram dat grotendeels correct is.	<input type="checkbox"/> Er is correct Datamodel en ERD diagram. Ook is er een Script aanwezig.
Binding*	<input type="checkbox"/> Er wordt geen binding toegepast.	<input type="checkbox"/> Er wordt binding toegepast.	<input type="checkbox"/> Bijna in alle gevallen wordt binding correct toegepast.	<input type="checkbox"/> In alle gevallen wordt binding correct toegepast.
Config.Bestand*	<input type="checkbox"/> Database connectie wordt niet via Config.Bestand ingesteld.	<input type="checkbox"/> Database connectie wordt via Config.Bestand ingesteld.		
Data classes*	<input type="checkbox"/> Database class leest gegevens niet in dataclasses in.	<input type="checkbox"/> Database class leest gegevens in dataclasses in.	<input type="checkbox"/> Database class leest gegevens in dataclasses in en ook de overige CRUD functies werken met meestal dataclasses.	<input type="checkbox"/> Database class leest gegevens in dataclasses in en ook de overige CRUD functies werken met meestal dataclasses. Er wordt gebruik gemaakt van de geoptimaliseerde databaseclass-code
Kan Code uitleggen*	<input type="checkbox"/> Student weet onvoldoende uit te leggen wat de code van zijn programma doet.	<input type="checkbox"/> Student kan zijn programma goed uitleggen.		

Bijlagen van Los Pollos Hermanos

In deze bijlage tref je alle gegevens aan die door de door jou te maken applicatie nodig zijn.

Overzicht maaltijden (peildatum: 1-2-2022)

Los Dos	2 Burgers van 100% rundvlees met verse sla, cheddar smeltkaas, verse uitjes en fijne slices van augurk. Met legendarische Big Mac saus op een 3-delig sesambroodje.	€ 7,10
El Grande	Een royale burger van 100% rundvlees, cheddar smeltkaas, verse uitjes en fijne augurk slices. Afgemaakt met een lekker laagje ketchup en mosterd op een getoast sesambroodje.	€ 8,25
El Pollo Picanto	De enige echte kipburger. Een burger van malse kipfilet, omhuld met een krokant laagje. Daarbovenop verse sla en een frisse sandwichsaus op een getoast sesambroodje.	€ 7,95
El Sabroso Doble Grande	Twee grootse burgers van 100% rundvlees met crunchy bacon, emmentaler smeltkaas, juicy tomaat, verse uitjes en frisse sla. Met een speciale grillsaus op een geroosterd sesambroodje.	€ 9,35
Kroket Hollanda	De echte oer-Hollandse burger. Knapperig vanbuiten en zacht van binnen. Met een vulling van ragout met stukjes puur rundvlees. Afgemaakt met een lekkere laag mosterd.	€ 6,25

Overzicht dranken

Cola	€ 2,25
Sinas	€ 2,25
7-up	€ 2,25

Overzicht menu's

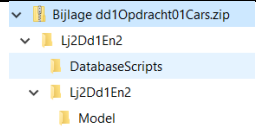
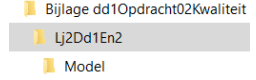
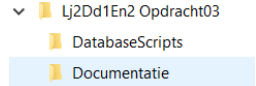
Lekker trek	Een overheerlijke Los Dos met een grote friet met naar keuze een 2 porties mosterd of ketchup. Natuurlijk neem je er ook een drankje naar keuze bij.	€ 12,50
Smakelijk eten	El Pollo Picanto met een portie versegebakken friet. Voeg daar een heerlijke grillsaus aan toe met de overheerlijke ragout. Om het af te maken kies je er een heerlijk drankje bij.	€ 11,80
Gezondheid	Maak de Kroket Hollanda af met een extra portie salade met heerlijke tomaatjes. Ook hier geldt dat de drank geheel naar jouw keuze is.	€ 9,30

Overzicht Ingrediënten (peildatum: 1-2-2022)

Rundvlees burger	€ 3,23	Stuks		Mosterd	€ 0.25	Kuipje
Sla	€ 0,12	Blaadjes		Ragout	€ 4.20	300 gram
cheddar smeltkaas	€ 0.35	plakken		kipfilet	€ 4.15	Stuks
verse uitjes	€ 0.15	portie		sandwichsaus	€ 0.18	Portie
augurk	€ 0.13	schijfjes		crunchy bacon	€ 0.28	120 gram
Big Mac saus	€ 0.30	portie		emmentaler smeltkaas	€ 0.45	schijf
3-delig sesambroodje	€ 0.50	Suks		grillsaus	€ 0.35	Portie
ketchup	€ 0.20	Portie		tomaat	€ 0.65	Schijfjes
getoast sesambroodje	€ 0.45	Stuks				
geroosterd sesambroodje	€ 0.45	Stuks				

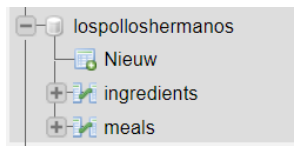
Bijlagen bij opdrachten

- Zorg ervoor dat je Visual Studio 2022 hebt geïnstalleerd. Oudere *installer*-en *IDE-versies* deïnstalleer je. Je maakt gebruik van C# 6.0 of hoger.
- Let bij de database scripts op voor welke database deze zijn. Dit kan zowel MySQL/MariaDb als SqlServer zijn.

Opdracht	Bestand	Toelichting	Afbeelding
Opdracht 01	Bijlage dd1Opdracht01Cars.zip	Bevat startcode en een script voor de MySQL-database carsdb.sql (folder DatabaseScripts)	
Opdracht02	Bijlage dd1Opdracht02Kwaliteit.zip	Bevat de code die de student aan de kwaliteitseisen moet aanpassen. Er wordt gebruikt gemaakt van hetzelfde database script als bij opdracht 01.	
Opdracht03	Bijlage dd1Opdracht03ScriptsEnTemplates.zip	Bevat Visio templates voor use cases, ERD en Class diagrams; Word template voor datamodel, Basisscript voor de database.	

Bijlagen PhpMyAdmin

Nieuwe tabel



Server: 127.0.0.1 » Database: lospolloshermanos

Structuur SQL Zoeken Query opbouwen Exporteren Importeren Handelingen Rechten Routines Gebeurtenissen Triggers

Tabelnaam: units Toevoegen 1 Kolom(men) Starten

Naam	Type	Lengte/Waarden	Standaardwaarde	Collatie	Attributen	Leeg	Index	A_i	Opmerkingen
unitid	INT		Geen			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	
name	VARCHAR	50	Geen			<input type="checkbox"/>	---	<input type="checkbox"/>	
	INT		Geen			<input type="checkbox"/>	---	<input type="checkbox"/>	

Uniek Index

Selecteer gewenste kolom en kies *Unieke waarde*

Server: 127.0.0.1 » Database: lospolloshermanos » Tabel: units

Verkennen Structuur SQL Zoeken Invoegen Exporteren Importeren Rechten Handelingen Triggers

Tabelstructuur Relatieoverzicht

#	Naam	Type	Collatie	Attributen	Leeg	Standaardwaarde	Opmerkingen	Extra	Actie
<input type="checkbox"/>	1 unitid	int(11)			Nee	Geen		AUTO_INCREMENT	Veranderen Verwijderen Meer
<input checked="" type="checkbox"/>	2 name	varchar(50)	utf8mb4_general_ci		Nee	Geen			Veranderen Verwijderen Meer

↑ Selecteer alles Met geselecteerde: Verkennen Veranderen Verwijderen Primaire sleutel Unieke waarde Index Ruimtelijk

Volledige tekst

Relaties

Server: 127.0.0.1 » Database: lospolloshermanos » Tabel: ingredients

Verkennen Structuur SQL Zoeken Invoegen Exporteren Importeren Rechten Handelingen Triggers

Tabelstructuur Relatieoverzicht

Beperkingen voor vreemde sleutels

Acties	Beperkingseigenschappen	Kolom	Vreemde sleutel (INNODB)
			Database Tabel Kolom
	FK_UNIT		unitid lospollosherman units unitid
	ON DELETE RESTRICT ON UPDATE RESTRICT		+ Kolom toevoegen

+ Voeg beperking toe

SQL-voorbeeld Opslaan

Met RESTRICT geeft je aan dat je geen Units mag verwijderen die nog in de tabel Ingredients staan.