In WinForms met een Database Verbinden

Met behulp van Entity Framework Core (EFCore)

# Introductie

We maken in deze les een applicatie om producten te bekijken. Deze producten vallen onder bepaalde categorieën. Deze oefening lijkt een beetje op de opdracht bij het project ‘Barroc Intens’, desalniettemin staat deze oefening daar los van. Laat je dus qua opdracht niet verwarren. *Je kunt delen van de code hier uiteraard wel vrij gebruiken in je project.*

Afbeelding met tafel

Automatisch gegenereerde beschrijving

We gaan Windows Forms in combinatie met MySQL gebruiken. Dit doen we omdat we al kennis hebben van deze beide technieken.

*Er zijn modernere manieren om met databases te werken, maar om die nu in een project in te zetten is niet slim. Dan zouden we zowel veel nieuws moeten leren als een applicatie bouwen voor een klant. Die combinatie brengt veel extra druk met zich mee. Voor nu is MySQL en WinForms een prima combinatie voor het bouwen van een applicatie die intern bij een organisatie gebruikt gaat worden.*

**Globaal zijn dit de stappen die we in deze les gaan ondernemen:**

1. Een nieuwe Solution starten
2. De juiste NuGet package(s) binnenhalen
3. Model aanmaken
4. Met een MySQL Database verbinden
5. Met behulp van de modellen de database bouwen en vullen
6. Met behulp van een DataGridView gegevens uit de database laten zien

*Dit document is losjes gebaseerd op deze handleiding van Microsoft:* [*https://learn.microsoft.com/en-us/ef/core/get-started/winforms*](https://learn.microsoft.com/en-us/ef/core/get-started/winforms)*.*

Inhoud

[Introductie 1](#_Toc116379266)

[Solution opstarten en instellen 3](#_Toc116379267)

[Models aanmaken 6](#_Toc116379268)

[Database connectie met DbContext 9](#_Toc116379269)

[Connectie testen 12](#_Toc116379270)

[Data beheren met een DataGridView 14](#_Toc116379271)

[Installeer benodigdheden in Visual Studio 18](#_Toc116379272)

[WinUI 3 Project klaarzetten 18](#_Toc116379273)

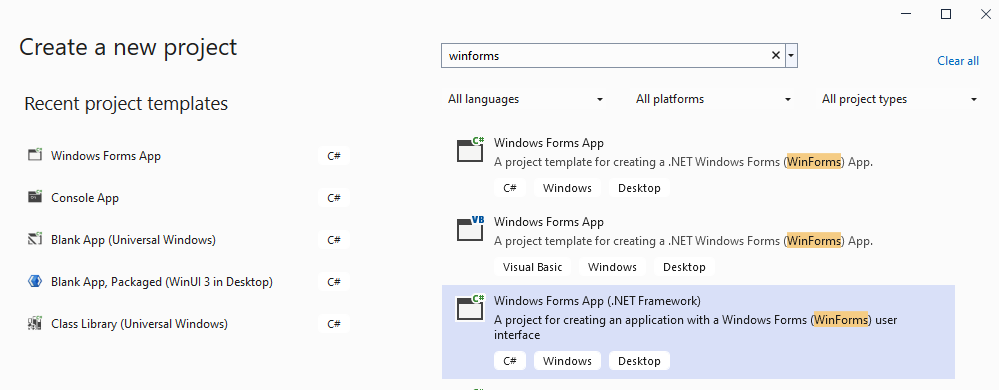
[MySQL database bouwen en gebruiken 19](#_Toc116379274)

[Mogelijke problemen/bugs/errors 21](#_Toc116379275)

# Solution opstarten en instellen

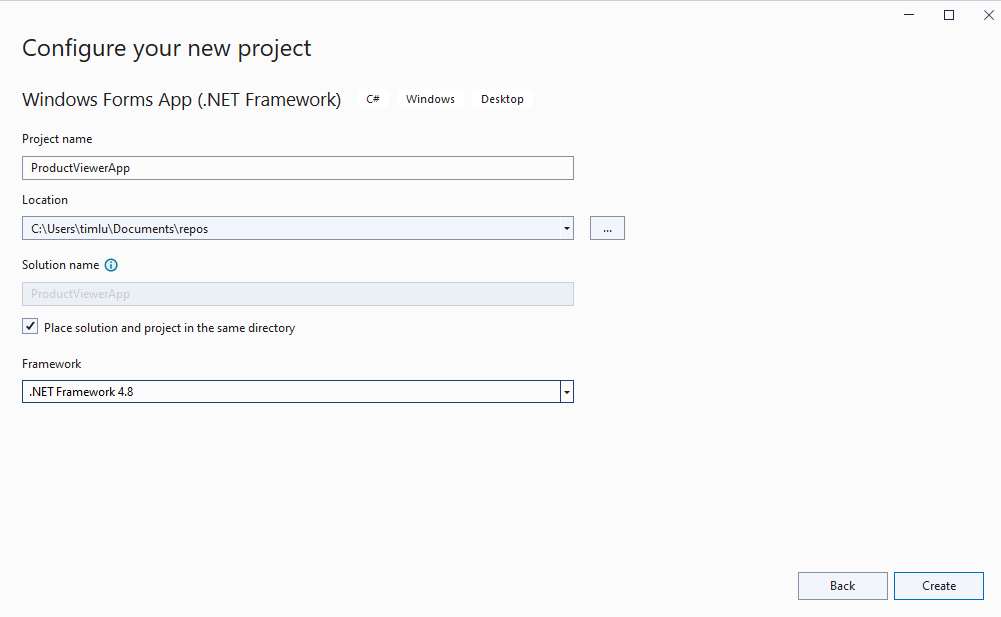
1. Start een WinForms (.NET Framework) project

*.NET Core ondersteund nog niet de DataGridView control die we gaan gebruiken.*



1. Selecteer een .NET Framework versie hoger dan 4.6.1+

*Wij gaan een NuGet Package gebruiken die enkel die versies van het .NET Framework ondersteunt.*

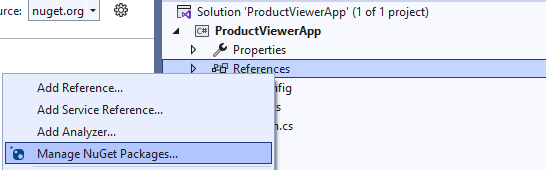


1. We gaan een NuGet package toevoegen aan dit project, die ons gaat helpen om met een MySQL database te verbinden: <https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql>
2. Neem de documentatie van die package door. Je ziet dat het .NET Framework ondersteund wordt enkel bij de Release 3.2.7:

*Afbeelding met tekst, schermafbeelding, monitor, zwart

Automatisch gegenereerde beschrijving*

1. Open de NuGet Package Manager voor het zojuist aangemaakte project:

**

1. Zoek de package in het Browse tabblad op en **selecteer de 3.2.7 versie** die werkt met onze versie van het .NET Framework:

*Afbeelding met tekst

Automatisch gegenereerde beschrijving*

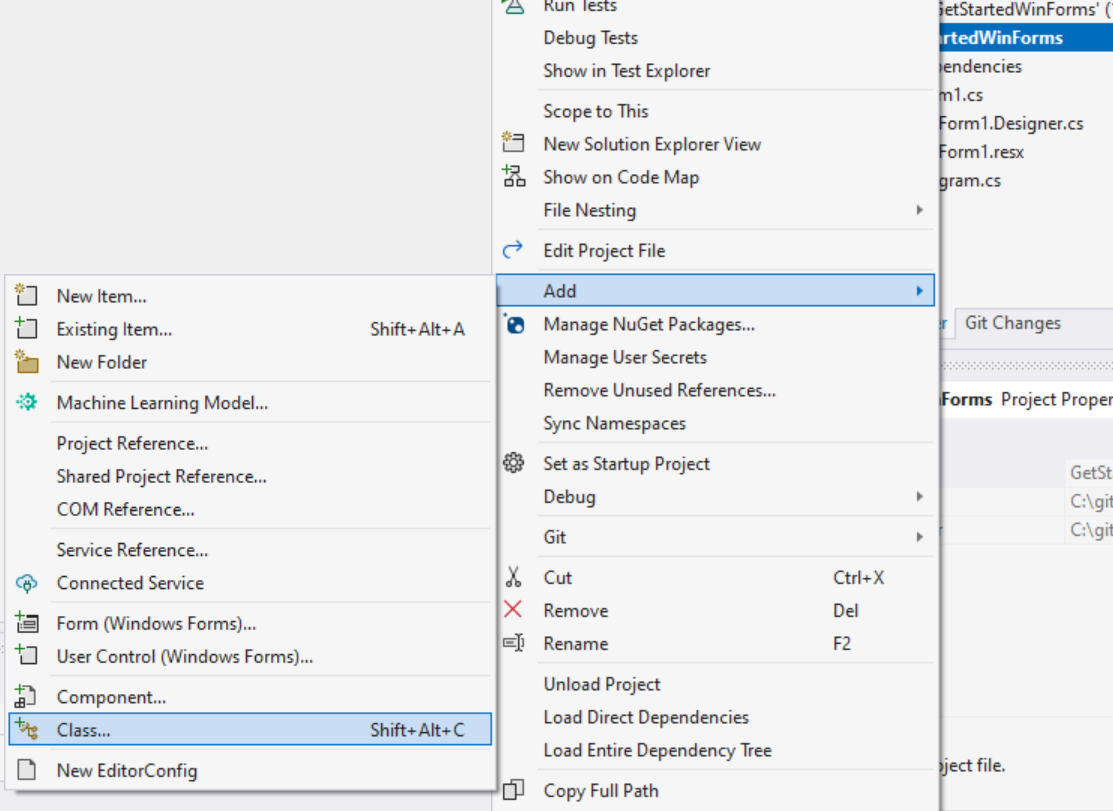
*Je ziet recht onderin de afbeelding hierboven dat deze versie “.NET Standard 2.0” ondersteund. Wanneer je dat ziet werkt de package ook vanaf het .NET Framework 4.6.1 of hoger. Bron:* [*https://learn.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-2-0#select-net-standard-version*](https://learn.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-2-0#select-net-standard-version)

1. Klik op Install/Installeer om dit NuGet package aan ons project toe te voegen.
2. Als laatste stap hernoemen we Form1.cs naar MainForm.cs

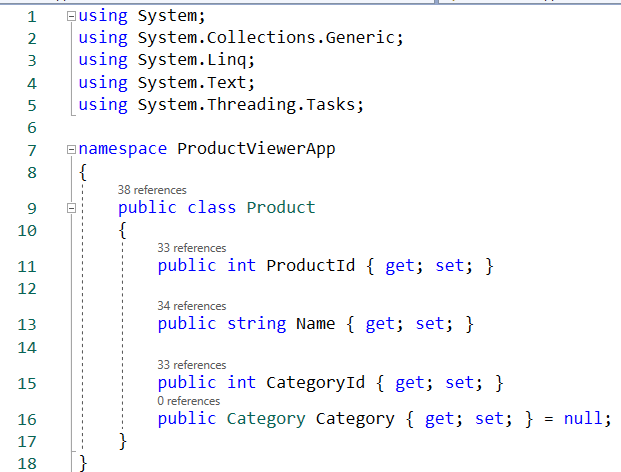
# Modellen aanmaken

We gaan onze database ‘Code First’ opbouwen. Dat wil zeggen dat we eerste classes gaan maken, die we vervolgens tot database tabellen laten omzetten door Entity Framework. *Het alternatief is om op basis van een database ontwerp, onze classes te genereren.*

1. Voeg een nieuwe class toe aan ons project voor onze ‘producten’:



1. Noem de klasse “Product” en plaats het in de file “Product.cs”
2. Pas de klasse aan zodat deze er zo uitziet:



*(Let op dat er een rood error-lijntje staat onder Category, totdat je de volgende stappen doorlopen hebt)*

1. Maak nog een klasse aan voor onze categorieën:
2. Noem de klasse ‘Category’ in de file ‘Category.cs’ en vul die met deze code:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*(*ObservableCollectionListSource komt uit de namespace ‘Microsoft.EntityFrameworkCore.ChangeTracking’, zorg dat die using dus bovenaan staat.)

De Product klasse is redelijk duidelijk. Een aantal eigenschappen van het product worden opgeslagen én via de ‘Category’ eigenschap is er een verwijzing naar een Category-object.

De Category klasse heeft een bijzondere eigenschap ‘Products’. Een categorie kan meerdere producten bevatten. Daarom willen we daar een lijst, maar we gebruiken hier een krachtigere lijst. Een ‘ObservableCollectionListSource’ is een lijst die in de gaten kan worden gehouden door andere code in onze applicatie. Een ObservableCollectionListSource’ lijst roept events aan wanneer er iets verandert. Dat gaat goed uitkomen om wijzigingen direct in onze schermen te laten zien. De DataGridView die we straks gaan gebruiken luistert ook naar die events om zo de laatste data te tonen.

De basis van onze applicatie is nu gebouwd: we hebben dankzij deze klassen duidelijk welke objecten er kunnen zijn en welke eigenschappen ze allemaal hebben.

# Database connectie met DbContext

We gaan werken met een MySQL database. Vervolgens verbinden we vanuit onze C# code met die database. Onze C# code gaat de database structuur ook opbouwen en vullen met test data.

1. Open XAMPP (of Laragon o.i.d.)
2. Start MySQL
3. Maak met PHPMyAdmin (of HeidiSQL o.i.d.) een database aan voor onze applicatie. Noem de database “barroc\_intens\_app”.

Nu onze database klaar staat gaan we vanuit onze applicatie met C# verbinden. Dat doen we doormiddel van een ‘Database Context’. Dat is een structuur bedacht door Microsoft en het gebruikt functionaliteiten uit de Entity Framework Core NuGet package.

1. Open ons project in Visual Studio en maak een nieuwe klasse ‘AppDbContext’
2. Pas de klasse aan zodat deze er zo uit ziet:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Laten we rustig naar deze code kijken, er zijn een aantal bijzonderheden:

* De klasse AppDbContext is erft eigenschappen en methodes over van ‘DbContext’ (die klasse komt uit: ‘Microsoft.EntityFrameworkCore’):



* + In OOP-talen kunnen klassen van elkaar ‘overerven’. Zo kunnen we functionaliteiten van andere ontwikkelaars in onze eigen klassen opnemen.
  + We gebruiken de dubbele punt om overerving aan te geven.
  + Microsoft weet dat applicaties met een database willen kunnen verbinden, daarom hebben ze een DbContext gemaakt die door andere applicaties gebruikt kan worden.
  + Onze AppDbContext verbind met een MySQL database met bepaalde gegevens. Een andere applicatie noemt het misschien OtherDbContext en gebruikt een ander soort database. Beide benutten ze de eigenschappen en methodes die DbContext laat overerven.
* Er zijn twee eigenschappen van het type ‘DbSet’: Dat zijn weer een bijzonder soort lijsten, dit keer zijn het lijsten gekoppeld aan database tabellen. Tussen de < > geven we het type van ieder item in de lijst aan:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* + De eigenschap Products gaat gekoppeld worden aan een ‘products’ tabel
  + De eigenschap Categories aan een ‘categories’ tabel
* De eerste methode die we zien ziet er zo uit:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* + Met ‘override’ overschrijven we de ‘OnConfiguring’ methode in DbContext. Standaard doet DbContext niet zo veel. Wij overschrijven de methode en stellen via de opties in (optionsBuilder) dat er met een MySQL database verbonden moet worden.
  + We geven aan de optionsBuilder.UseMySQL(x) een connection string mee. Daarin zit alle informatie die nodig is om te weten:
    - server = waar is de database? Vaak een IP-adres óf localhost als we lokaal werken.
    - database = wat is de naam van de database? Deze databasenaam hebben we aan het begin van dit hoofdstuk gekozen
    - user = wat is de gebruikersnaam van een gebruiker die toegang heeft tot de database? We gebruiken root omdat dat in XAMPP een gebruiker is die overal toegang toe heeft. (In productieomgevingen zouden we een gebruiker met beperkte rechten aanmaken)
    - password = wat is het wachtwoord voor die gebruiker? We laten dit leeg omdat in XAMPP er geen wachtwoord is voor de root gebruiker.
    - Al deze connectiegegevens worden gescheiden door puntkomma tekens.
* In de OnModelCreating methode gaan we onze database ‘seeden’. *Ofwel: vullen met (test-)data*. Kopieer de onderstaande code in de OnModelCreating:

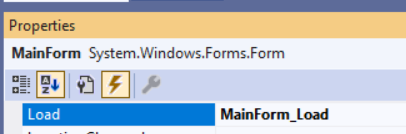
|  |
| --- |
| modelBuilder.Entity<Category>().HasData(  new Category { CategoryId = 1, Name = "Cheese" },  new Category { CategoryId = 2, Name = "Meat" },  new Category { CategoryId = 3, Name = "Fish" },  new Category { CategoryId = 4, Name = "Bread" });  modelBuilder.Entity<Product>().HasData(  new Product { ProductId = 1, CategoryId = 1, Name = "Cheddar" },  new Product { ProductId = 2, CategoryId = 1, Name = "Brie" },  new Product { ProductId = 3, CategoryId = 1, Name = "Stilton" },  new Product { ProductId = 4, CategoryId = 1, Name = "Cheshire" },  new Product { ProductId = 5, CategoryId = 1, Name = "Swiss" },  new Product { ProductId = 6, CategoryId = 1, Name = "Gruyere" },  new Product { ProductId = 7, CategoryId = 1, Name = "Colby" },  new Product { ProductId = 8, CategoryId = 1, Name = "Mozzela" },  new Product { ProductId = 9, CategoryId = 1, Name = "Ricotta" },  new Product { ProductId = 10, CategoryId = 1, Name = "Parmesan" },  new Product { ProductId = 11, CategoryId = 2, Name = "Ham" },  new Product { ProductId = 12, CategoryId = 2, Name = "Beef" },  new Product { ProductId = 13, CategoryId = 2, Name = "Chicken" },  new Product { ProductId = 14, CategoryId = 2, Name = "Turkey" },  new Product { ProductId = 15, CategoryId = 2, Name = "Prosciutto" },  new Product { ProductId = 16, CategoryId = 2, Name = "Bacon" },  new Product { ProductId = 17, CategoryId = 2, Name = "Mutton" },  new Product { ProductId = 18, CategoryId = 2, Name = "Pastrami" },  new Product { ProductId = 19, CategoryId = 2, Name = "Hazlet" },  new Product { ProductId = 20, CategoryId = 2, Name = "Salami" },  new Product { ProductId = 21, CategoryId = 3, Name = "Salmon" },  new Product { ProductId = 22, CategoryId = 3, Name = "Tuna" },  new Product { ProductId = 23, CategoryId = 3, Name = "Mackerel" },  new Product { ProductId = 24, CategoryId = 4, Name = "Rye" },  new Product { ProductId = 25, CategoryId = 4, Name = "Wheat" },  new Product { ProductId = 26, CategoryId = 4, Name = "Brioche" },  new Product { ProductId = 27, CategoryId = 4, Name = "Naan" },  new Product { ProductId = 28, CategoryId = 4, Name = "Focaccia" },  new Product { ProductId = 29, CategoryId = 4, Name = "Malted" },  new Product { ProductId = 30, CategoryId = 4, Name = "Sourdough" },  new Product { ProductId = 31, CategoryId = 4, Name = "Corn" },  new Product { ProductId = 32, CategoryId = 4, Name = "White" },  new Product { ProductId = 33, CategoryId = 4, Name = "Soda" }); |

Nu onze Database Context hebben moeten we die bij het opstarten van onze applicatie enkel nog laten uitvoeren:

1. Voeg boven in onze MainForm.cs een veld toe waar we onze Database Context gaan opslaan:



1. Maak een event handler die luistert naar het opstarten van onze eerste form:



1. Vul de event handler met deze code:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* + Bij het laden maken we een nieuwe AppDbContext aan en slaan die op in ons veld in de Form. Vanaf dit punt kunnen we hier in de MainForm bij de database context.
  + Met ‘this.dbContext.Database.EnsureCreated()’ zorgen we dat de database structuur gebouwd wordt. Als je met PHPMyAdmin (of HeidiSQL) in de database kijkt, dan zijn (nadat deze regel is uitgevoerd) de tabellen en kolommen aangemaakt.
  + Entity Framework gebruikt de classes en eigenschappen om te bepalen welke tabel- en kolomnamen er moeten komen.
  + EnsureCreated maakt de database als die er niet is én voert de OnModelCreating methode uit in onze database context. Zo kan na het opbouwen van de database ook (test-)data in de tabellen gezet worden.

# Connectie testen

Om te testen of we toegang hebben tot de database gegevens kunnen we een stukje test code schrijven in de MainForm\_Load. Neem de hieronder gemarkeerd code over, zet een breakpoint onderaan de methode, start de applicatie en inspecteer de variabelen:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Als het goed is zie je dat er 5 categories zijn én dat de eerste category 10 producten bevat.*

**Hier wat uitleg over de code hierboven:**

* 
  + Haalt alle categorieën op en zet ze in een List (waar je doorheen kunt ‘loopen’)
* Afbeelding met tekst

  Automatisch gegenereerde beschrijving
  + Dit zegt eigenlijk:
    - Haal van de eerste (0) categorie regel (Entry) in de database
    - de Products eigenschap op (dat is een verzameling/Collection)
    - Laad al die gerelateerde regels (Load)
  + We halen hier alle producten op die horen bij deze category.
  + Bij een een-op-meer relatie gebruiken we Collection. Bij een een-op-een relatie: Reference (zie hoofdstuk Extra onderaan dit document)
* 
  + Nu de products in de category geladen zijn kunnen we die naar een lijst omzetten.

Als je gegevens uit andere tabellen niet expliciet laad op deze manier, dan lijkt de lijst leeg.

Entity Framework voert op de achtergrond database query’s uit om de gegevens op te halen.

# Fist run of the applicationData beheren met een DataGridView

Een DataGridView (zie screenshot rechts) is een element waarin gegevens in tabelvorm gepresenteerd worden. Daarnaast kunnen gegevens direct in de tabel aangepast worden.

Voor Project Barroc Intens zijn er een paar schermen waar dit mogelijk interessant is. Bij de andere schermen kun je een combinatie van Label, TextBox, CheckBox, etc. elementen gebruiken.

Volg vanaf nu de handleiding vanaf het hoofdstuk ‘Adding controls to the form’: <https://learn.microsoft.com/en-us/ef/core/get-started/winforms#adding-controls-to-the-form>

# Extra

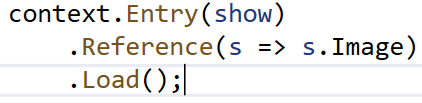
In deze voorbeelden is een Show een Netflix Show. Een Episode hoort bij één Show. Een Show heeft meerdere Episodes. Er is ook een Image, een Image hoort bij één Show of één Episode.

* Gerelateerde data (uit een andere tabel) moet geladen worden met

Explicit, Eager of Lazy loading:

<https://docs.microsoft.com/en-us/ef/core/querying/related-data/explicit>

Voorbeeld voor een-op-een relatie:



*Image is hier een relatie naar een ander model/tabel*

Of om meteen bij het ophalen van de Shows ook de gerelateerde data te joinen:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Hier laden we meteen bij het opvragen van alle shows óók de bijbehorende Image en Episodes informatie.*

**Veel voorkomende errors**

* InvalidOperationException bij loop door data:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

**Oorzaak:** context.Shows doet iedere foreach herhaling een nieuwe query en houdt binnen de loop de database verbinding bezet. Hierdoor kan de context niet opnieuw benaderd worden.

**Oplossing:** De hele query uitvoeren doormiddel van ToList, voordat de loop gestart wordt:

