3 Lagen Applicatie in WPF

Opbouw stappenplan

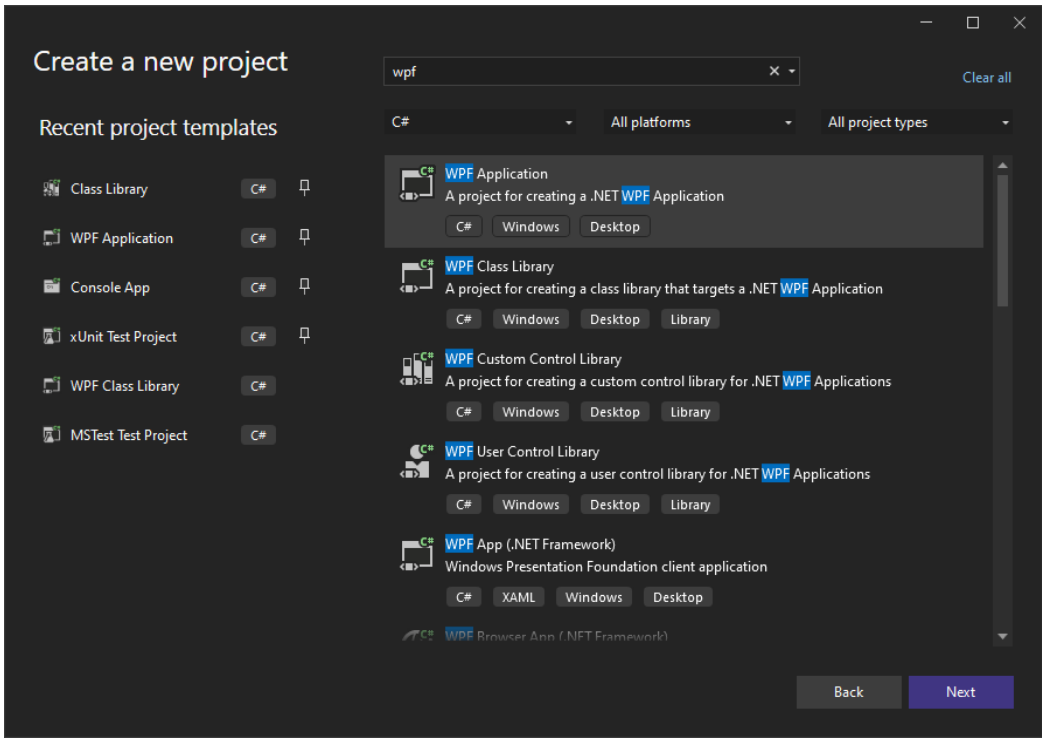
Deel 1 - Solution opzetten

Startup project

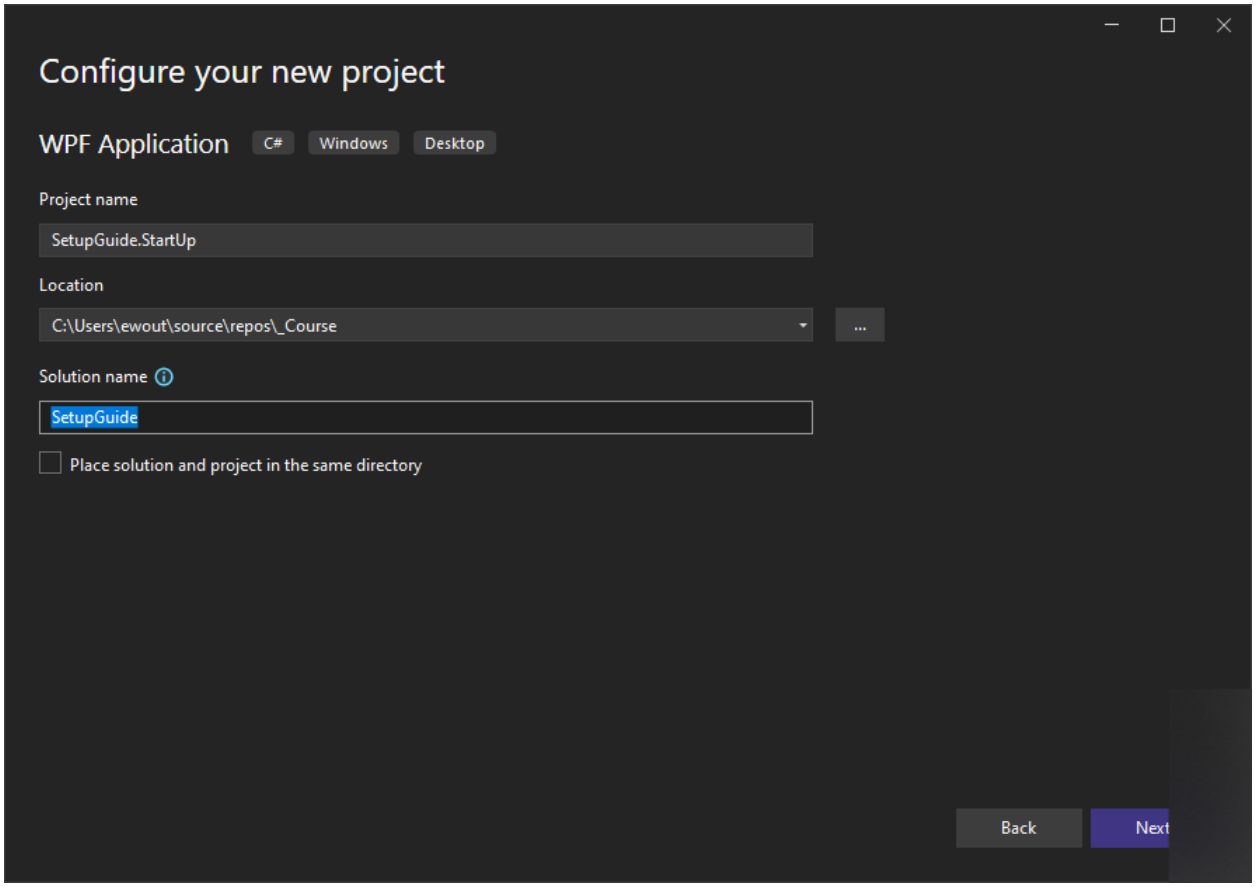
We beginnen het opzetten van de solution door een naam te bedenken, deze zullen we gebruiken doorheen de volledige solution. Zorg dus dat je een goede naam kiest die je achteraf niet meer moet veranderen. In volgende stappen wordt hiernaar verwezen als {app_naam}. *In de screenshots zal je zien dat hiervoor SetupGuide gekozen is als projectnaam.*

We beginnen met het aanmaken van een startup project, dit is geen volwaardige laag binnen het drie lagen model. De startup klasse in dit project breekt sommige afspraken die gelden voor de rest van de solution daarom zetten we deze apart.

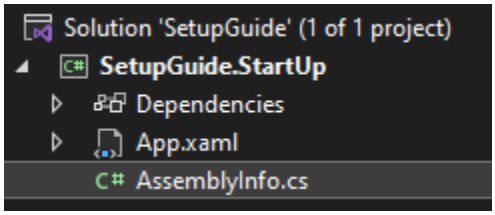
Maak dit project aan met het template “WPF Application”.



Geef je solution de naam van je applicatie en het project de naam {app_naam}.StartUp. **Pas op bij het ingeven van je *Project name*, dit veld is verbonden aan het *Solution name* veld.** Geef dus eerst de volledige naam van je project in voordat je de naam van de solution ingeeft.



We hebben nu een solution die één project bevat. Verwijder hier de MainWindow, ons StartUp project bevat geen windows. Die zitten in de presentation layer.

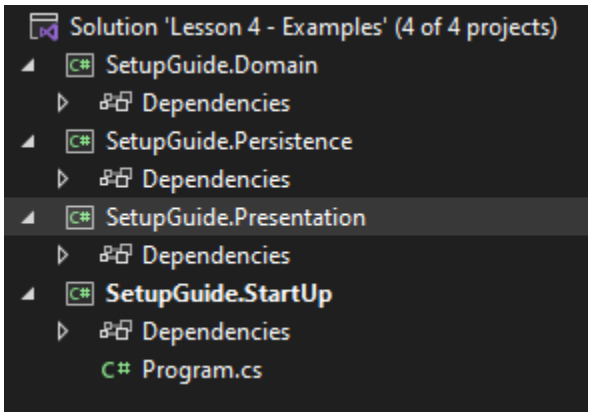


Drie lagen projecten

Nu voegen we drie lagen toe aan de solution, rechtsklik op je solution > add > new project. Voeg hier de domain en persistence projecten toe die elk gebruik maken van de “Class Library” template. Voor de presentation layer maken we gebruik van de “WPF Class Library” template.

De standaard naamgeving van de drie lagen is als volgt, je mag hier variaties van gebruiken. Zo lang het onderscheid tussen de lagen maar duidelijk is.

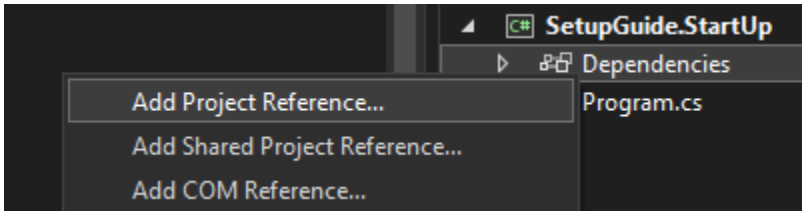
1. {app_naam}.Presentation
2. {app_naam}.Domain
3. {app_naam}.Persistence



Dependencies

Dependencies op voorhand vastleggen

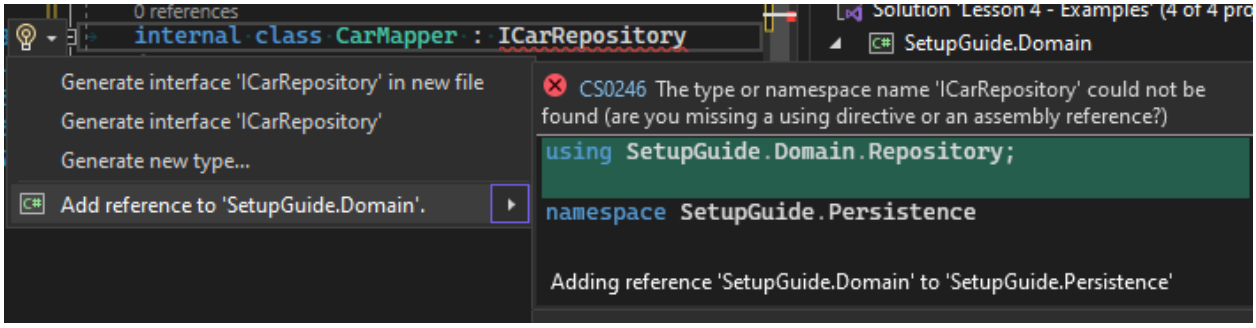
Voeg in het Startup project dependencies toe naar de andere drie projecten. Dit doen we door in het startup project rechts te klikken op Dependencies en dan Add Project Reference…



Voeg op dezelfde manier in het Persistence project een dependency toe naar het Domain project. Ook in het Presentation project voegen we een dependency toe aan het Domain project. Normaal zou je de references ook moeten zien verschijnen als je de dependencies van een project aanklikt en kijkt onder projects. Eenmaal deze vijf dependencies gelegd gaan we door met de opbouw van ons project.

Dependencies achteraf instellen

Bovenstaande manier om dependencies vast te leggen is de gemakkelijkste, zeker in het begin. Maar er is ook nog een andere manier: je kan de verbindingen ook leggen tijdens het schrijven van je applicatie. Let dan wel op dat je de references in de juiste richting legt.



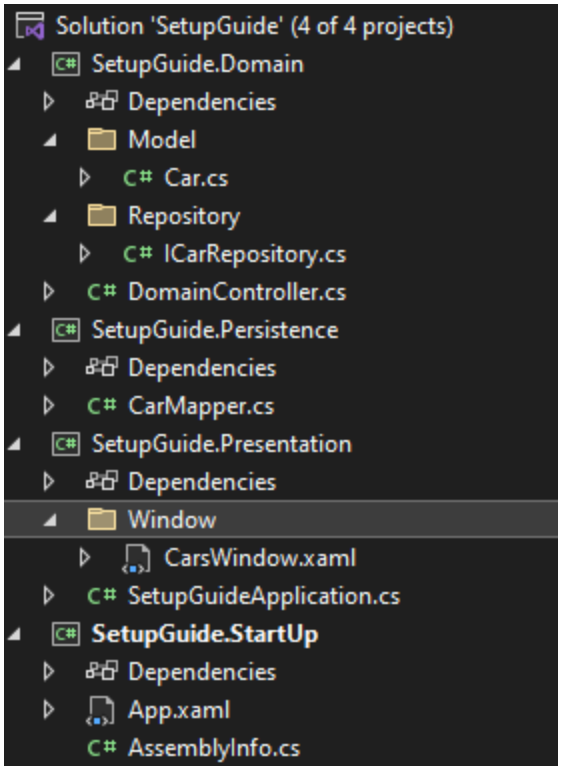
Deel 2 - Structuur schrijven

Klassen aanmaken

Maak in de verschillende projecten volgende klassen aan (of hernoem de bestaande):

- Presentation
    - {app_name}Application
    - Een of meerdere windows (eventueel in een subfolder)
- Domain
    - DomainController
    - Een of meerdere data klassen/models (eventueel in een subfolder)
    - Repository interface per data klasse (eventueel in een subfolder)
- Persistence
    - Mapper klasse per data klasse/model
- Startup
    - Behoud de App.xaml file

In dit voorbeeld doen we alsof onze SetupGuide applicatie met data van auto's werkt, in de praktijk verandert dit natuurlijk afhankelijk van de oefening of applicatie die je maakt.



Public en internal access modifiers

Vanaf nu moet je beginnen met het gebruiken van het keyword internal als access modifier. Deze waren tot nu toe identiek, onze projecten bestonden bijna uitsluitend uit één project in één solution. Als je deze access modifier gebruikt in plaats van public zal je die klasse, property of methode niet kunnen gebruiken buiten het project waar het in zit.
Ga daarom zeker niet alles standaard public zetten, het is nog steeds het beste om de access scope zo klein mogelijk te houden. Daarmee wordt bedoeld dat alles wat internal kan staan best internal blijft, enkel wat buiten het project (of de laag van je applicatie) toegankelijk moet zijn mag je public zetten.

Namespaces

Zorg er voor dat de namespaces van alle klasses ook kloppen. Een namespace moet bestaan uit de naam van je solution + de naam van je project + eventuele subfolders. Voor dit voorbeeld zal dat er als volgt uit zien:

```
namespace SetupGuide.Domain.Model
```

Persistence

Zorg ervoor dat elke mapper klasse zijn respectievelijke interface uit de Domain layer implementeert. Let er op dat methodes om de data aan te spreken hier **en** in de geimplementeerde IRepository interface gedefinieerd worden.

Domain

De constructor van de DomainController verwacht een instantie van elke IRepository interface mee te krijgen via zijn constructor. Deze instanties worden bijgehouden in private (readonly) fields.

Presentation

De constructor van de {app_name}Application verwacht een instantie van de DomainController mee te krijgen via zijn constructor. Deze instantie wordt bijgehouden in een private (readonly) field.

Vergeet niet om in de Application klasse een window aan te maken en te tonen, anders zal je applicatie wel opstarten maar niets visualiseren.

Startup

De App.xaml klasse bouwt laag voor laag onze applicatie op in een EventHandler van het StartUp event.

```
<Application x:Class="SetupGuide.StartUp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:SetupGuide.StartUp"
             Startup="Application_Startup">
    <Application.Resources>

    </Application.Resources>
</Application>
```

In de Persistence layer wordt elke IRepository interface geïnstantieerd met een Mapper object die de respectievelijke interface implementeert.
De Domain layer wordt aangemaakt door de DomainController te instantieren en in zijn constructor elke IRepository interface mee te geven die we eerder aan maakten.

De Presentation layer wordt aangemaakt door de {app_name}Application klasse aan te maken en de instantie van de DomainController mee te geven in zijn constructor.

```
using SetupGuide.Domain;
using SetupGuide.Domain.Repository;
using SetupGuide.Persistence;
using SetupGuide.Presentation;
using System.Windows;

namespace SetupGuide.StartUp
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    3 references
    public partial class App : Application
    {
        1 reference
        private void Application_Startup(object sender, StartupEventArgs e)
        {
            // Create persistence layer
            ICarRepository carRepository = new CarMapper();

            // Create domain layer & inject persistence layer repositories
            DomainController domainController = new(carRepository);

            // Create presentation layer & inject domain controller
            SetupGuideApplication application = new(domainController);
        }
    }
}
```

Deel 3 - Regels bij het drie lagen model

1. De Persistence & Presentation layer mogen nooit rechtstreeks met elkaar communiceren. Dit moet altijd via de Domain layer gebeuren.
2. Gebruik zo weinig mogelijk data klassen in je Presentation layer, deze horen thuis in de Domain layer. Hou de presentation layer zo eenvoudig mogelijk en koppel zo rap mogelijk terug naar de DomainController.
3. Zorg ervoor dat je namespaces altijd overeenkomen met de project en folderstructuur.
4. Werk in je Domain layer steeds met verwijzingen naar de repository interfaces, nooit rechtstreeks met de mapper klasse.