Opgave Employees

**Run het SQL Bestand zodat je tabel gegenereerd worden.**

# Doelstelling

We gaan een applicatie ontwikkelen die de simulatie van gegevensbeheer doet.

De applicatie bestaat uit 3 delen:

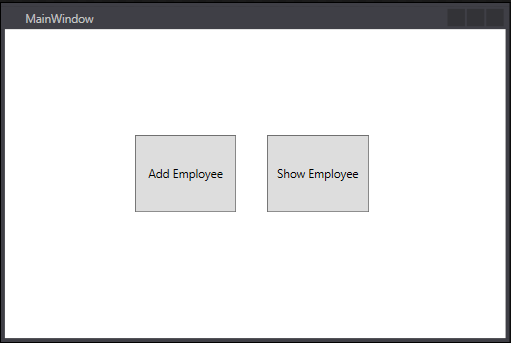
* Mainwindow dat het homescreen van de applicatie is
* AddEmployeeWindow, hier gaan we een employee toevoegen
* ShowEmployeeWindow, hier kunnen we de employees die we toegevoegd hebben bekijken
* EditEmployeeWindow, hier kunnen we aanpassingen doen aan de employee.

# Gevraagd:

Via het Drielagenmodel ga je de nodige handelingen doen om een employee toe te voegen, editen of te verwijderen.

Zorg voor een strikte scheiding van de business klassen en de GUI.

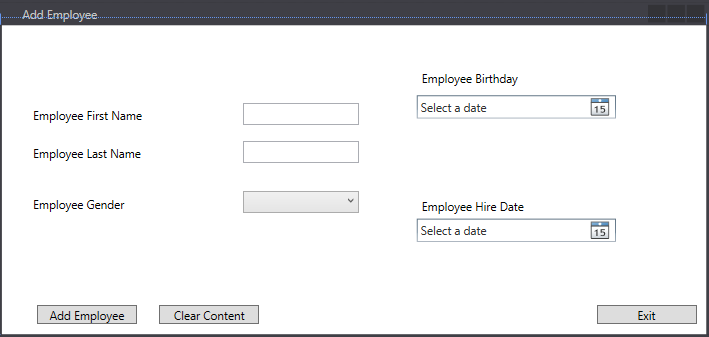
# Deel 1: Mainwindow



Zorg ervoor dat de knoppen van het mainwindow naar het juiste window verwijzen.  
In een later stadium als je de nodige functionaliteit geïmplementeerd hebt zorg je ervoor dat de Show employee-button slechts selecteerbaar is als je minstens 1 employee in je databank hebt steken.

Zorg er dus voor dat je in dit venster alle Employees uit de databank haalt en deze in een List steekt, we gaan dus niet rechtstreeks op de databank werken maar met een tussenvariabele (List).

# Deel 2:



Implementeer hier de nodige functionaliteit om ervoor te zorgen dat je Employees kunt toevoegen.  
Je zorgt ervoor dat alle velden gecontroleerd worden of ze ingevuld zijn en voorziet een methode om alle velden automatisch leeg te maken.

De methode om de velden leeg te maken zet je afzonderlijk in een aparte klasse Utility, zorg ervoor dat je deze methode kan aanspreken via Utility.ClearControls() met de nodige parameters indien nodig.

Als je op de exit-knop drukt dan moet je teruggaan naar het mainwindow.

Wanneer je een Werknemer hebt toegevoegd worden de controls automatisch leeg gemaakt.

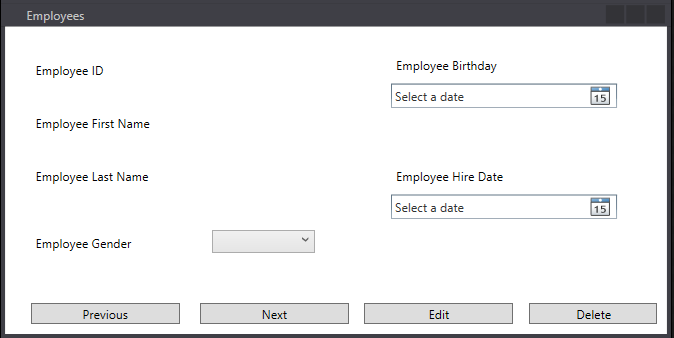
## Het toevoegen van een Employee

Een employee bestaat uit de volgende data:

* emp\_no
* first\_name
* last\_name
* gender
* birth\_date
* hire\_date

Zorg ervoor dat voor het emp\_no automatisch het volgende nummer wordt voorzien door een aparte methode GetNextID() te schrijven.

# Deel 3:

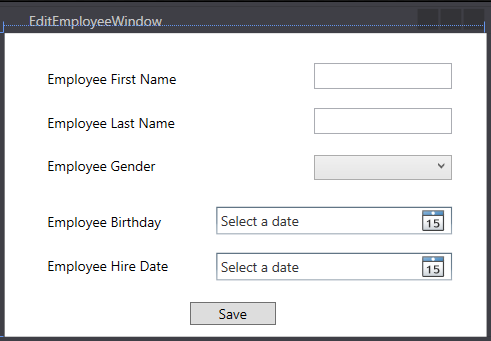


In het ShowEmployeeWindow ga je record per record de gegevens van de persoon laten zien, je mag hiervoor gewoon de List gebruiken, zorg ervoor dat je de List van het mainwindow doorgeeft aan dit window zodat wijzigingen aan de list op alle windows doorgevoerd wordt.

Je zorgt ervoor dat je in dit window niets kunt aanpassen aan de data, hiervoor gebruiken we het edit window, dit window is dus puur voor het tonen van de data )

Maak gebruik van databinding om de data op de componenten te tonen.

# Deel 4



In dit window zal je de data van de geselecteerde klant kunnen aanpassen.

Zorg ervoor dat je de aanpast in de List en vervolgens het juiste Employee object doorgeeft aan de methode die je database gaat updaten.

Zorg er ook voor dat je de list correct update, als je op save drukt zal je terugkeren naar het showemployees window en moet de data ook daar aangepast zijn.

# Wat doen?

* Zorg voor een Insert methode 🡺 void InsertEmployee(Employee emp)
* Zorg voor een Methode die alle employees ophaalt 🡺 List<Employee> GetEmployees()
* Zorg voor een methode die de volgende ID in de database opvraagt voor emp\_no 🡺   
  int GetNextID()
* Zorg voor een Update methode die de employee kan updaten 🡺 void UpdateEmployee(Employee emp)
* Zorg voor een delete methode om een employee te verwijderen 🡺 void DeleteEmployee(Employee emp)
* Alle databank methoden zijn static
* Zorg ervoor dat je beveiliging goed is 🡪 geen sql injectie.
* Zorg ervoor dat je gebruik maakt van Commits en rollbacks waar nodig.
* Zorg voor een klasse Employee met de nodige variabelen/methoden
* Zorg voor een klasse Utility deze bevat de volgende methoden:
  + AllFilled(Parameters..)
  + ClearFunction
  + UpdateGUI
* Maak een connectie met je databank via de klasse DBConnection