

## Inhaltsverzeichnis

1. Ziele des Praktikums:.....	1
2. Vorbedingungen .....	1
3. WPF App (Windows Presentation Foundation) .....	2
4. DataAdapter und DataSet erstellen .....	4
5. User Interface erstellen (Schaltflächen).....	8
6. Data-bound Control erstellen .....	10

Geben Sie zum Beweis, dass Sie das Praktikum vollständig durchgeführt haben, einen Screenshot des letzten erzeugten GUIs (im Ausführungszustand, inkl. Daten aus der DB) im Dokument DT8.pdf ab. Abgabetermin ist eine Woche nach Praktikumsdurchführung.

### 1. Ziele des Praktikums:

1. Verwenden eines DataAdapters und eines Datasets.
2. DataBinding der gelesenen Daten an ein WPF-Control erstellen.

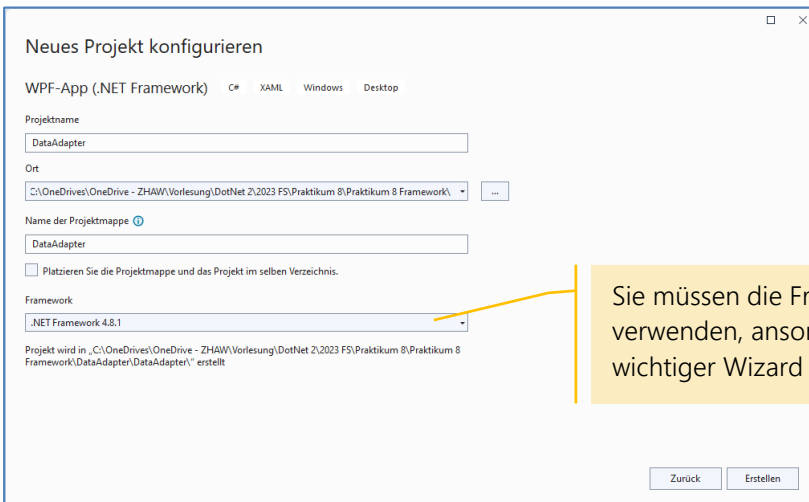
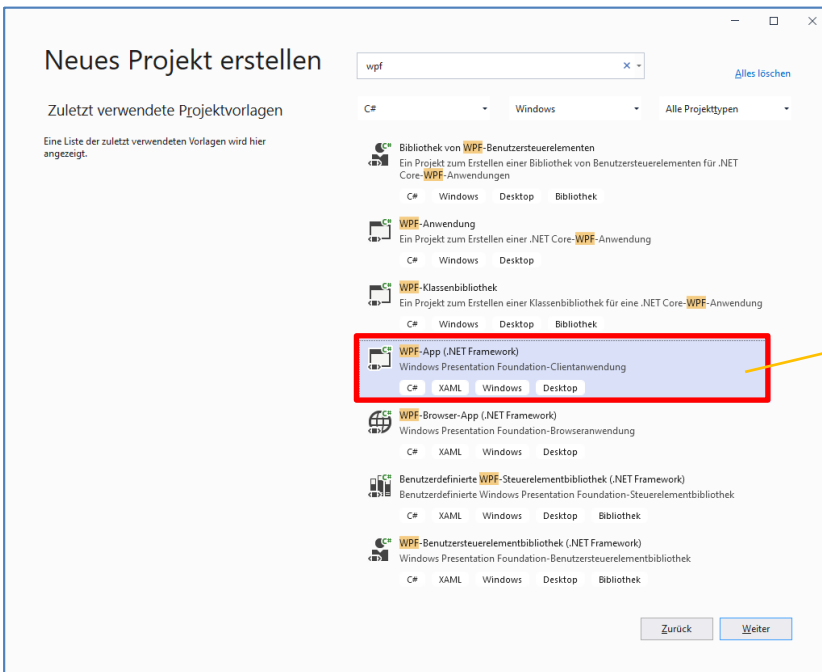
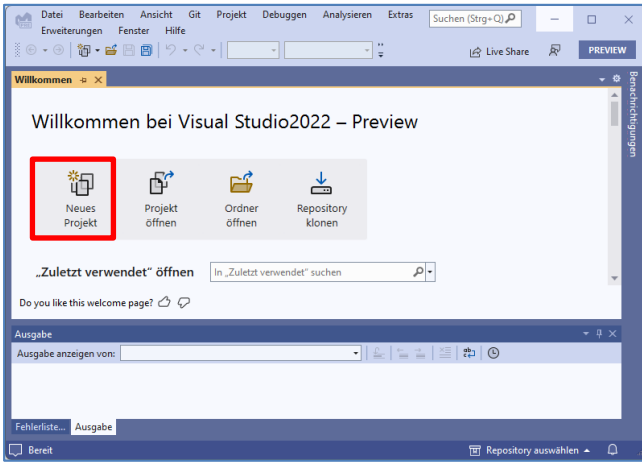
### 2. Vorbedingungen

Wir verwenden in der Übung die bereits im Praktikum 4/5 erzeugte FlughafenDB (inkl. SQL Server). Falls Sie diese nicht (mehr) haben, müssen Sie diese vorgängig wieder erstellen (siehe Praktikum 4).

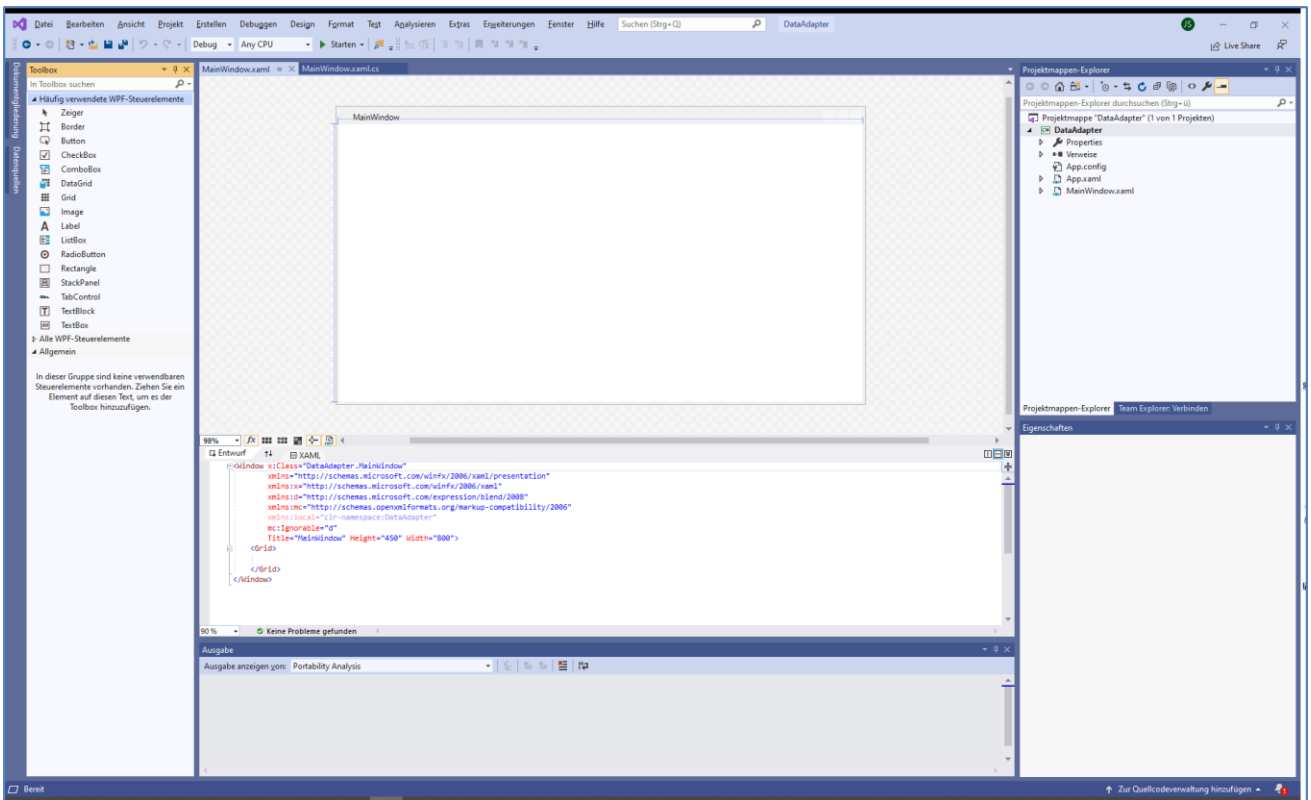
Im Folgenden wurde das Visual Studio Enterprise 2022 Version 17.6.0 Preview 3.0 (Deutsch) verwendet (sie müssen die Preview-Variante verwenden!), sowie das .NET Framework 4.8.1. Das Praktikum funktioniert nicht mit .NET Core – im Visual Studio werden notwendige Wizards nicht unterstützt... Sie müssten dazu dann alles von Hand programmieren.

### 3. WPF App (Windows Presentation Foundation)

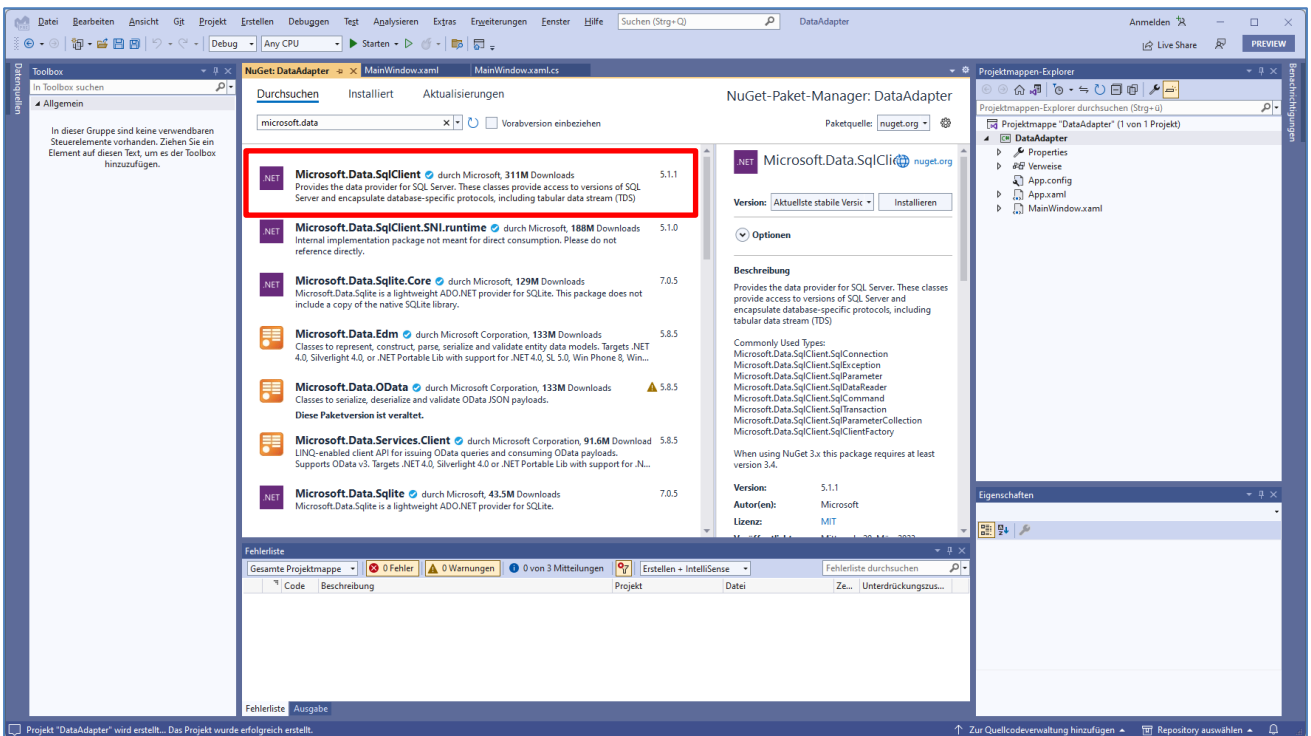
Wir erstellen eine WPF-App in welcher wir einen DataAdapter (plus DataSet) ausprobieren können:



Visual Studio erstellt das neue Projekt:

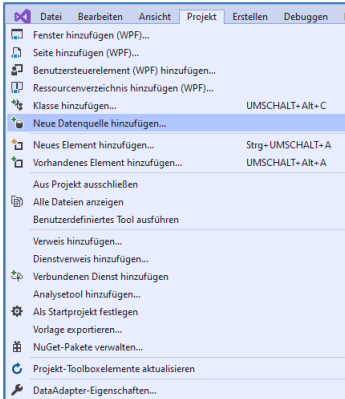


Installieren Sie das NuGet-Package Microsoft.Data.SqlClient (sonst erhalten Sie anschliessend Fehlermeldungen):

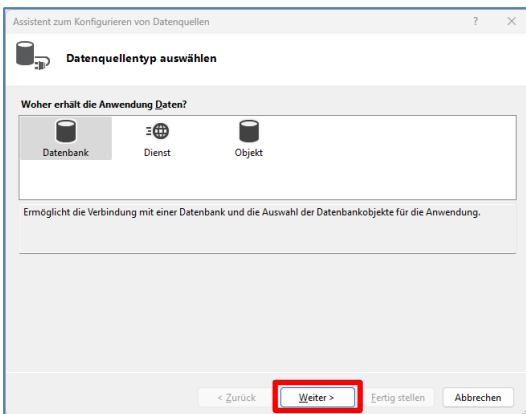


## 4. DataAdapter und DataSet erstellen

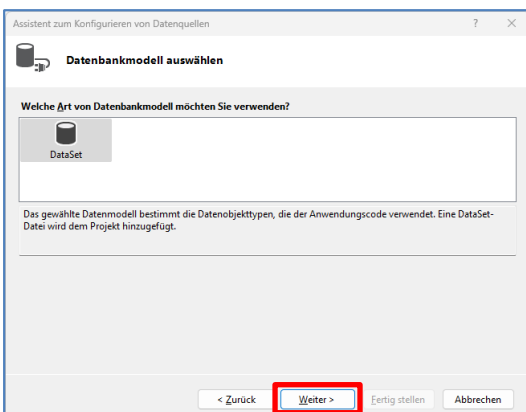
Bevor ein datengebundenes Control-Element angelegt werden kann, muss ein "Datenmodell" (DataAdapter und DataSet) für die Anwendung definiert und dem Data Sources-Fenster hinzugefügt werden. Um die Arbeit nicht manuell ausführen zu müssen, verwenden wir einen «Wizard». In der Core-Variante steht uns dieser nicht mehr zur Verfügung – wir müssten dort das Entity-Framework (nächste Lektion) verwenden.



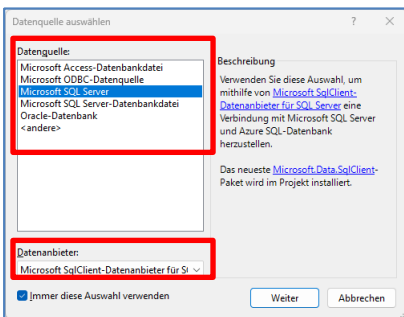
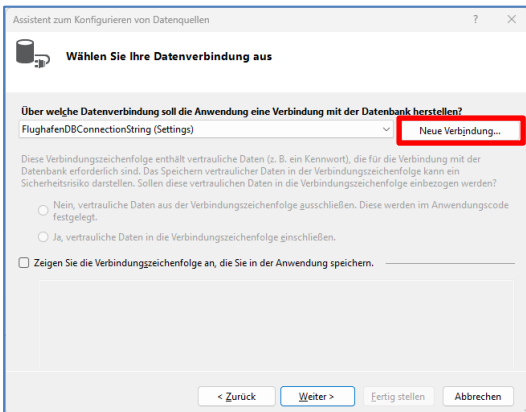
Im Wizard wählen wir die Datenbank aus:



Dann das Dataset (wir werden in der nächsten Vorlesung das Dataset genauer betrachten):

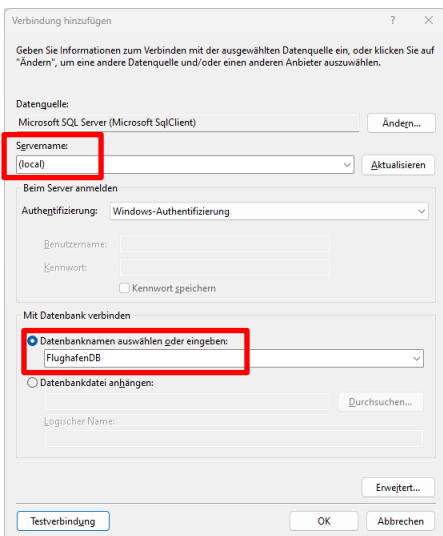


Schliesslich muss die Verbindung mit dem SQL Server und der Datenbank festgelegt werden. Sie können diese unter "New Connection" erstellen.

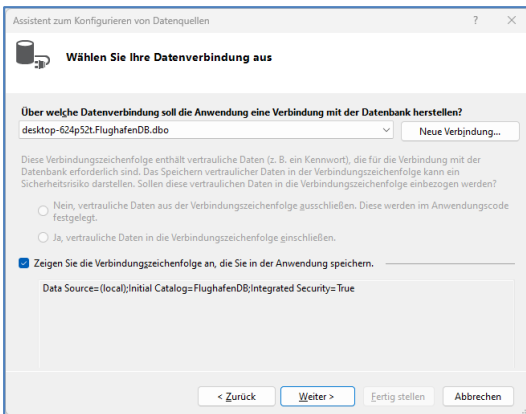


Wählen Sie jetzt den gewünschten SQL Server «(local)» und anschliessend die gewünschte Datenbank «FlughafenDB» aus. Achten Sie darauf, dass der SQL Server-Dienst läuft und dass Sie über die notwendigen Zugriffsrechte auf den SQL Server verfügen. Falls Sie den «Browser»-Dienst des SQL-Servers nicht installiert haben, oder dieser nicht aktiv ist, findet Visual Studio den SQL Server nicht ohne Hilfe. In diesem Fall müssen Sie den Namen (oder IP-Adresse) des Servers von Hand eingeben. Bei einer lokalen Instanz (mit Standardname) können Sie auch den Namen «(local)» verwenden.

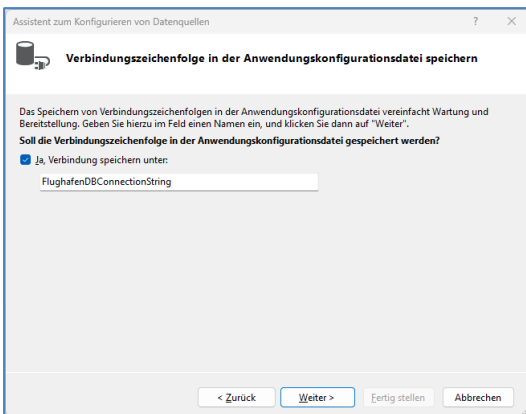
Mittels "Testverbindung" können Sie überprüfen, ob der Zugriff funktioniert. Danach wählen Sie «ok».



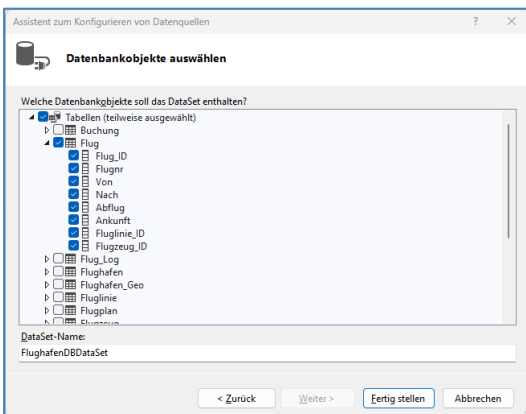
Sie sehen jetzt die Verbindung, sowie den Connection-String.



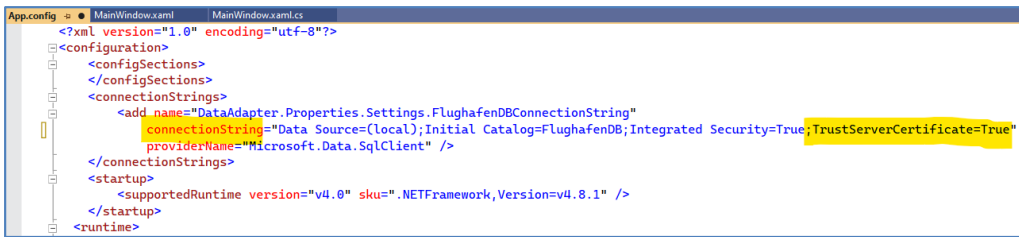
Der ConnectionString soll an zentraler Stelle in der Datei app.config gespeichert werden.



Wir wählen als zu verwendende Datenbankobjekte die Tabelle Flug und deren Attribute aus. Den DataSet-Namen übernehmen wir unverändert.



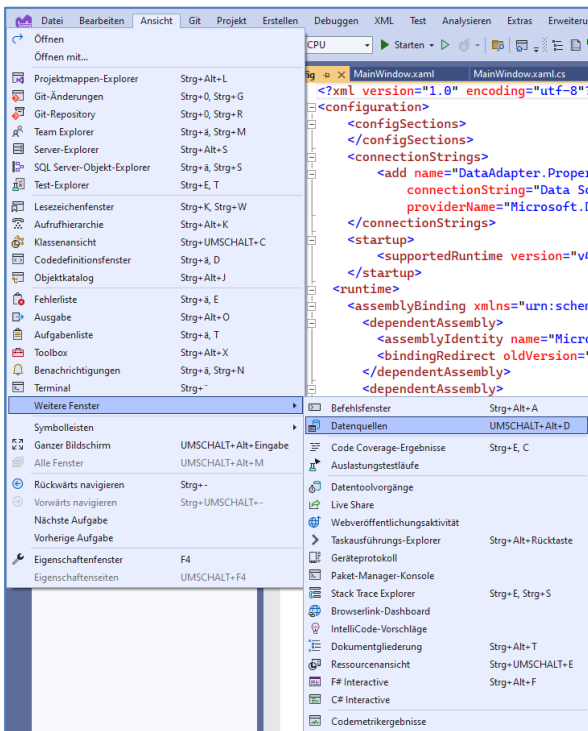
In Konfigurationsdatei "App.config" sieht man den erzeugten Connection-String.



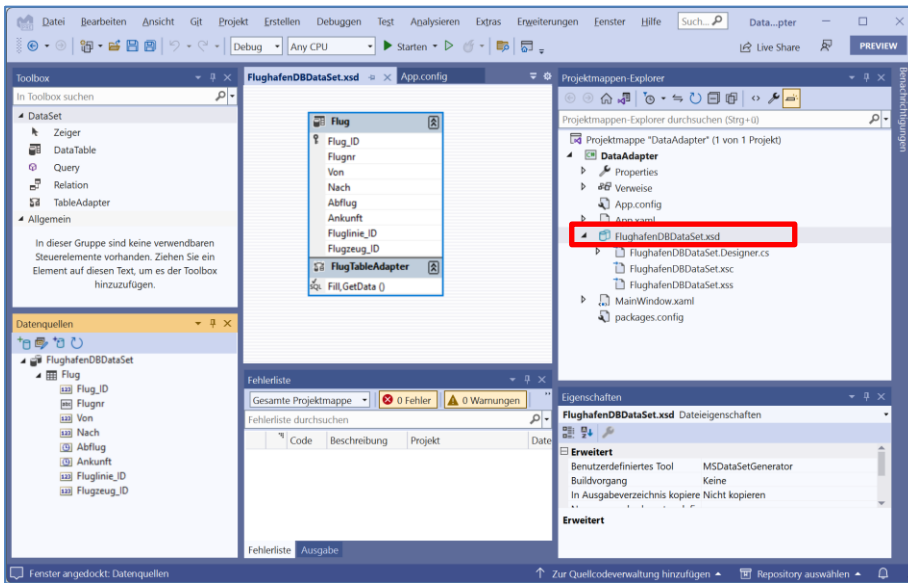
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="DataAdapter.Properties.Settings.FlughafenDBConnectionString"
      connectionString="Data Source=(local);Initial Catalog=FlughafenDB;Integrated Security=True;TrustServerCertificate=True"
      providerName="Microsoft.Data.SqlClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8.1" />
  </startup>
</runtime>
```

Beachten Sie, dass der ConnectionString im Bild am Ende mit TrustServerCertificate=True ergänzt wurde (früher in System.Data war dies noch nicht notwendig). Diesen Parameter müssen Sie von Hand ergänzen. Dadurch können wir mit ADO.NET auf den lokalen SQL-Server zugreifen, obwohl dieser kein «offizielles» Zertifikat hat. Damit ist die Data Source erstellt.

Aktivieren Sie die Data Sources Ansicht:



Sie sehen jetzt die Datenquellen – gleichzeitig können sie das grafische Modell im VS betrachten (FlughafenDB-DataSet.xsd mit Doppelklick öffnen). Zugegeben, ein sehr triviales Modell.



## 5. User Interface erstellen (Schaltflächen)

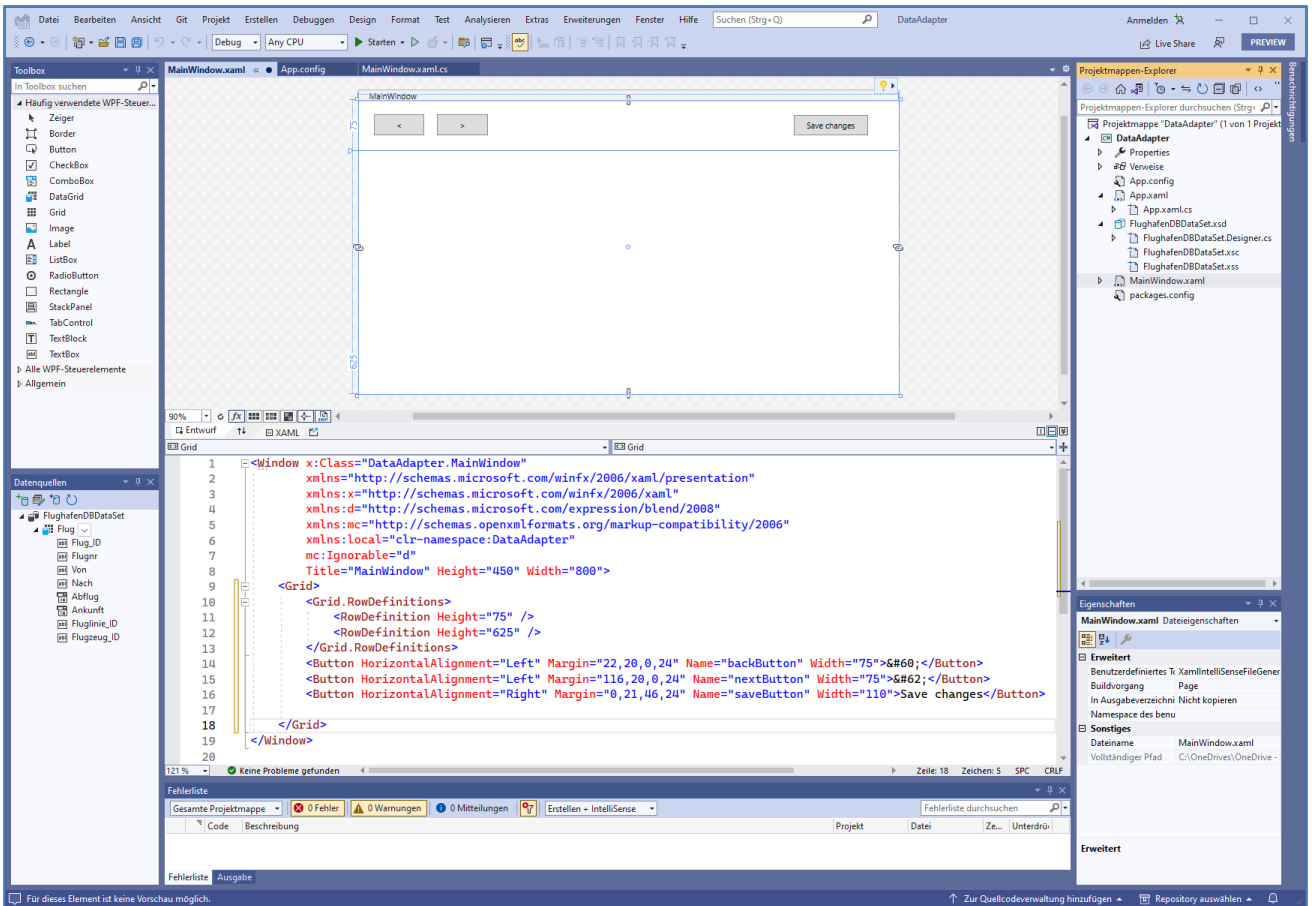
Wir fügen dem Fenster mehrere Schaltflächen hinzu, indem Sie das XAML im WPF-Designer ändern. Später werden wir Code hinzufügen, der es Benutzern ermöglicht, mit Hilfe dieser Schaltflächen durch die Flugdatensätze zu scrollen und Änderungen zu speichern.

Doppelklicken Sie im Projektmappe-Explorer auf MainWindow.xaml. Das Fenster öffnet sich im WPF-Designer. Fügen Sie in der XAML-Ansicht des Designers den folgenden Code zwischen den <Grid>-Tags hinzu:

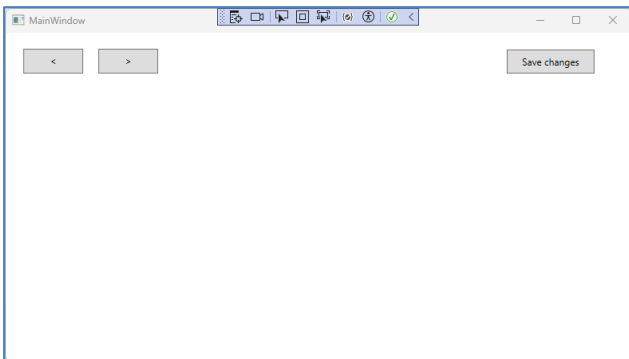
```
<Grid.RowDefinitions>
  <RowDefinition Height="75" />
  <RowDefinition Height="625" />
</Grid.RowDefinitions>
<Button HorizontalAlignment="Left" Margin="22,20,0,24" Name="backButton" Width="75">&#60;</Button>
<Button HorizontalAlignment="Left" Margin="116,20,0,24" Name="nextButton" Width="75">&#62;</Button>
<Button HorizontalAlignment="Right" Margin="0,21,46,24" Name="saveButton" Width="110">Save changes</Button>
```

Nach der Änderung sieht der XAML-Code wie folgt aus:





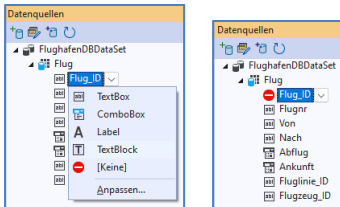
Das Projekt sollte sich starten lassen und folgendes Fenster erscheinen:



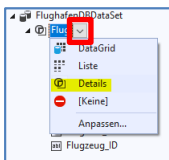
## 6. Data-bound Control erstellen

In den folgenden Schritten erstellen wir das Control, welches mit der DataSource (dem DataSet des DataAdapters) verbunden wird.

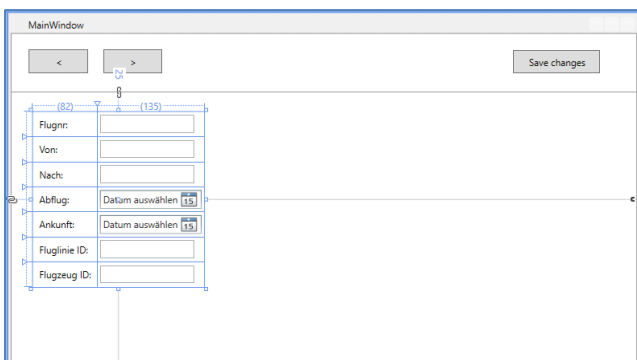
Stellen Sie sicher, dass das Fenster Data Sources (Datenquellen) sichtbar ist. Die Flug\_ID soll nicht angezeigt werden, deaktivieren Sie diese:



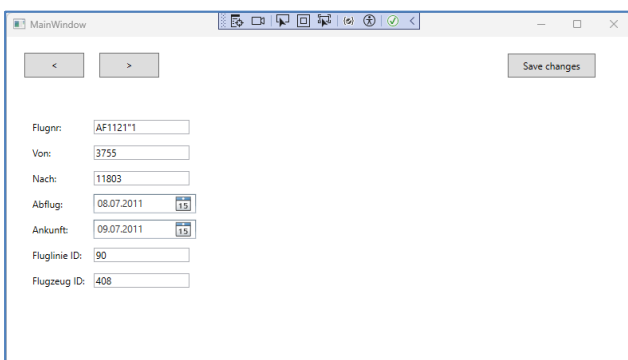
Klicken Sie im Fenster Data Sources (Datenquellen) auf das Dropdown-Menü rechts vom Flugknoten und wählen Sie Details (wir möchten die Daten/Details eines einzelnen Datensatzes anzeigen, nicht eine Liste):



Ziehen Sie den Flug-Knoten in die untere Zeile im MainWindow:



Starten Sie das Projekt. Es sollten jetzt Daten (der erste Datensatz) im MainWindow angezeigt werden, der User kann aber noch nicht mit den Buttons durch die Daten blättern.



Stoppen Sie die Programmausführung. Ergänzen Sie den Code so, dass flughafenDBDataSet, flughafenDBDataSetFlugTableAdapter und flugViewSource als private Attribute der Klasse MainWindow verfügbar sind (nicht vergessen: Deklarationen der Variablen in der Methode Window\_Loaded entfernen):

```
private DataAdapter.FlughafenDBDataSet FlughafenDBDataSet;
private DataAdapter.FlughafenDBDataSetTableAdapters.FlugTableAdapter flughafenDBDataSetFlugTableAdapter;
private System.Windows.Data.CollectionViewSource flugViewSource;
```

Die partielle Klasse MainWindows sieht dann wie folgt aus:

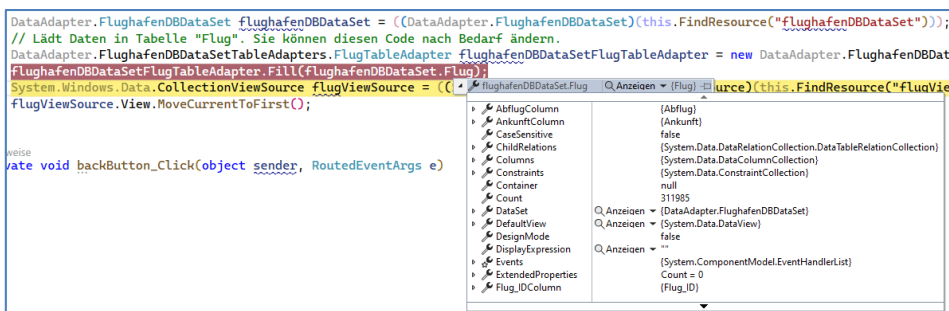
```
public partial class MainWindow : Window
{
    private DataAdapter.FlughafenDBDataSet FlughafenDBDataSet;
    private DataAdapter.FlughafenDBDataSetTableAdapters.FlugTableAdapter flughafenDBDataSetFlugTableAdapter;
    private System.Windows.Data.CollectionViewSource flugViewSource;

    0 references
    public MainWindow()
    {
        InitializeComponent();
    }

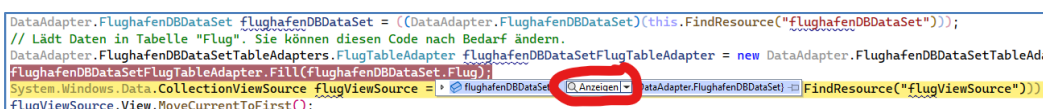
    1 reference
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        FlughafenDBDataSet = ((DataAdapter.FlughafenDBDataSet)(this.FindResource("flughafenDBDataSet")));
        // Load data into the table Flug. You can modify this code as needed.
        flughafenDBDataSetFlugTableAdapter = new DataAdapter.FlughafenDBDataSetTableAdapters.FlugTableAdapter();
        flughafenDBDataSetFlugTableAdapter.Fill(flughafenDBDataSet.Flug);
        flugViewSource = ((System.Windows.Data.CollectionViewSource)(this.FindResource("flugViewSource")));
        flugViewSource.View.MoveCurrentToFirst();
    }

    1 reference
    private void BackButton_Click(object sender, RoutedEventArgs e)
    {
    }
}
```

Setzen Sie einen Breakpoint bei Anweisung flughafenDBDataSetFlugTableAdapter.Fill(flughafenDBDataSet.Flug). Hier werden die Daten aus der Tabelle Flug durch den DataAdapter flughafenDBDataSetFlugTableAdapter in die Tabelle Flug des DataSets flughafenDBDataSet geladen. Starten Sie das Programm, führen Sie dann das Fill-Statement im Einzelschritt aus. Das dauert ein zwei drei Sekunden – wir laden ja auch alle Daten und das sind über 300'000 Datensätze. Es sind sämtliche Daten der Tabelle im Memory. Schauen Sie sich das Dataset an.



Wenn Sie auf die Lupe klicken, können Sie die Daten im Dataset betrachten.



Entfernen Sie den Breakpoint wieder.

Stoppen Sie die Programmausführung. Erzeugen Sie die den BackButton\_Click Event (Doppelklick auf den Button) und ergänzen Sie ihn mit folgendem Code: flugViewSource.View.MoveCurrentToPrevious(); Erzeugen Sie die den NextButton\_Click Event (Doppelklick auf den Button) und ergänzen Sie ihn mit folgendem Code: flugViewSource.View.MoveCurrentToNext();

```

public partial class MainWindow : Window
{
    private System.Windows.Data.CollectionViewSource flugViewSource;
    private DataAdapter.FlughafenDBDataSet flughafenDBDataSet;
    private DataAdapter.FlughafenDBDataSetTableAdapters.FlugTableAdapter flughafenDBDataSetFlugTableAdapter;

    //Vererbung
    public MainWindow()
    {
        InitializeComponent();
    }

    //Ereignisse
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        flughafenDBDataSet = ((DataAdapter.FlughafenDBDataSet)(this.FindResource("FlughafenDBDataSet")));
        // Lädt Daten in Tabelle "Flug". Sie können diesen Code nach Bedarf ändern.
        flughafenDBDataSetFlugTableAdapter = new DataAdapter.FlughafenDBDataSetTableAdapters.FlugTableAdapter();
        flughafenDBDataSetFlugTableAdapter.Fill(flughafenDBDataSet.Flug);
        flugViewSource = ((System.Windows.Data.CollectionViewSource)(this.FindResource("FlugViewSource")));
        flugViewSource.View.MoveCurrentToFirst();
    }

    //Ereignisse
    private void backButton_Click(object sender, RoutedEventArgs e)
    {
        flugViewSource.View.MoveCurrentToPrevious();
    }

    //Ereignisse
    private void nextButton_Click(object sender, RoutedEventArgs e)
    {
        flugViewSource.View.MoveCurrentToNext();
    }
}

```

Starten Sie die Anwendung. Sie müssten jetzt mit den Buttons vorwärts und rückwärts navigieren können.

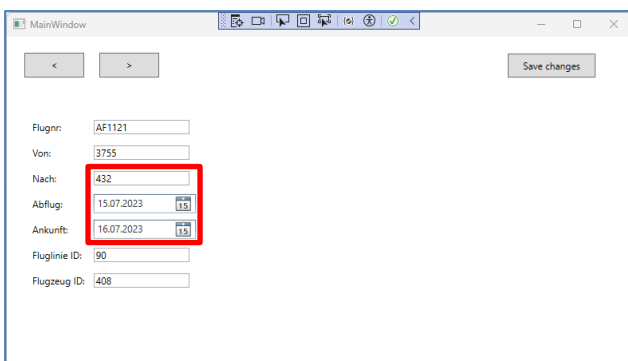
Es fehlt jetzt noch der «Save changes»-Button. Erzeugen Sie mittels Doppelklicks auf den «Save changes»-Button den SaveButton\_Click-Event. Ergänzen Sie diesem mit folgendem Code (speichert die Änderungen in der Tabelle Flug des Datasets flughafenDBDataSet im SQL Server):  
 flughafenDBDataSetFlugTableAdapter.Update(flughafenDBDataSet.Flug);

```

//Ereignisse
private void saveButton_Click(object sender, RoutedEventArgs e)
{
    flughafenDBDataSetFlugTableAdapter.Update(flughafenDBDataSet.Flug);
}

```

Starten Sie die Anwendung. Ändern Sie im zweiten Flug die Daten wie unten angegeben – siehe Bild (Achtung die Daten in der DB sind fehlerhaft, sie müssen die drei Werte anpassen, bei falschen Daten stürzt die Anwendung ab).



Speichern Sie die Änderung, starten Sie die Anwendung erneut (damit die Daten erneut geladen werden) und überprüfen Sie, ob die Daten gespeichert wurden. Achtung – da die Daten nicht nach einem bestimmten Kriterium sortiert geladen werden, kann es sein, dass der Datensatz NICHT mehr an der zweiten Stelle steht... viel Spass beim Suchen 😊. Notfalls direkt in der DB überprüfen.

Wenn alles funktioniert hat, hat der DataAdapter die Daten geladen und erfolgreich geändert.

Wir fügen jetzt noch eine Dritte Zeile ein. Ändern Sie die zunächst Höhe des Hauptfensters auf 800, die Höhe der zweiten Zeile auf 300 und fügen Sie die dritte Zeile ein (es wird eine dritte Zeile in der Vorschau sichtbar):

```

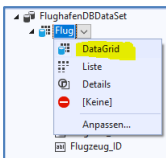
<Window x:Class="DataAdapter.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:DataAdapter"
  mc:Ignorable="d"
  Title="MainWindow" Height="800" Width="800" Loaded="Window_Loaded">
  <Window.Resources>
    <local:FlughafenDBDataSet x:Key="flughafenDBDataSet"/>
    <CollectionViewSource x:Key="flugViewSource" Source="{Binding Flug, Source={StaticResource flughafenDBDataSet}}"/>
  </Window.Resources>
  <Grid DataContext="{StaticResource flugViewSource}">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="287*" />
      <ColumnDefinition Width="505*" />
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
      <RowDefinition Height="75" />
      <RowDefinition Height="300" />
      <RowDefinition />
    </Grid.RowDefinitions>
  </Grid>

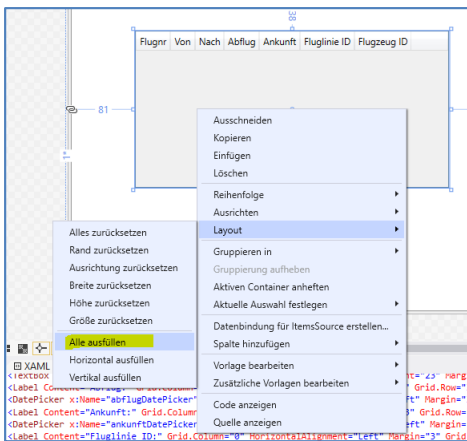
```

Neue 3. Zeile

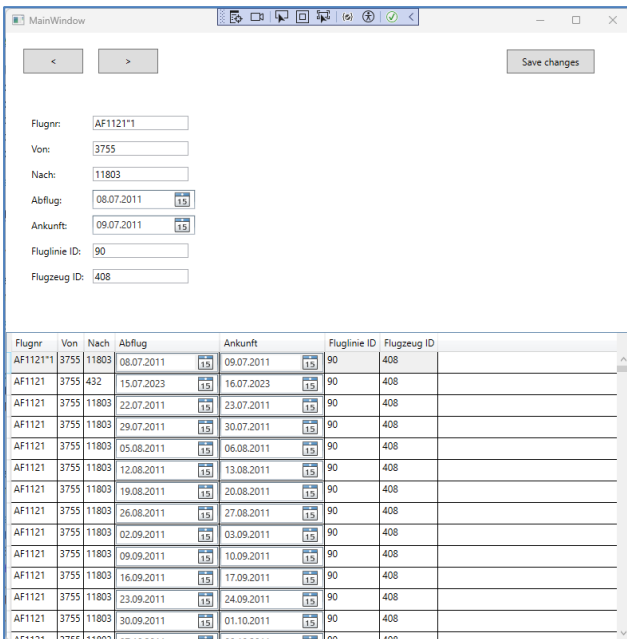
Ziehen Sie dieses Mal das DataGrid des Flug-Knotens der Data Source in die dritte Zeile (zunächst DataGrid auswählen und dann Drag and Drop des Knotens).



Füllen Sie den gesamten Bereich mit FillAll:



Nach dem Starten der Anwendung haben wir jetzt zusätzlich eine Listenansicht.



In diesem Praktikum haben wir fast alles mit Wizards generiert. Das erzeugt viel Code. Mit etwas Geduld könnte eine wesentlich schlankere und verständlichere Anwendung erstellt werden.

Zu schnell fertig? Erweitern Sie Ihre Anwendung so, dass Sie auch Daten einfügen und löschen können -> Achtung, die Flug\_ID ist in der DB auf autoinkrement gesetzt, diese wird automatisch gesetzt.