

Blazor Lab 5

Dauer: 50 Minuten

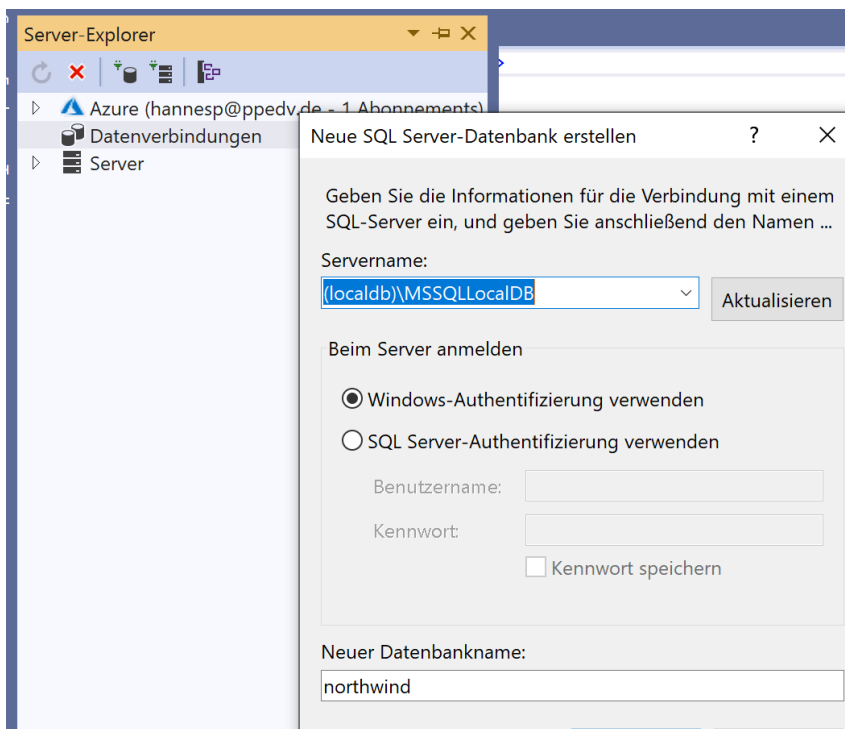
In diesem LAB wird eine SQL Datenbank angelegt, ein Entity Model Zugriffsschicht erzeugt und Daten aufgelistet und editiert. Auf den Schulungsrechnern ist die SQL Express Datenbank vorhanden. Ggf kann man über die Kommandozeile das prüfen mit folgenden Kommando

```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\hanne>sqllocaldb info
MSSQLLocalDB
```

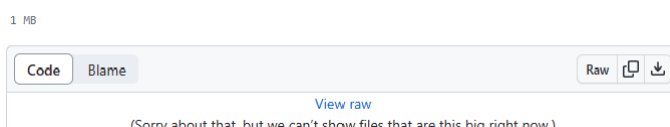
Erstellen sie eine neue **Blazor Web App** BlazorLab05. Interactive Mode Server.

Fügen Sie einen Neue Datenbank hinzu. Server Explorer links – Datenverbindungen -neue Datenbank.

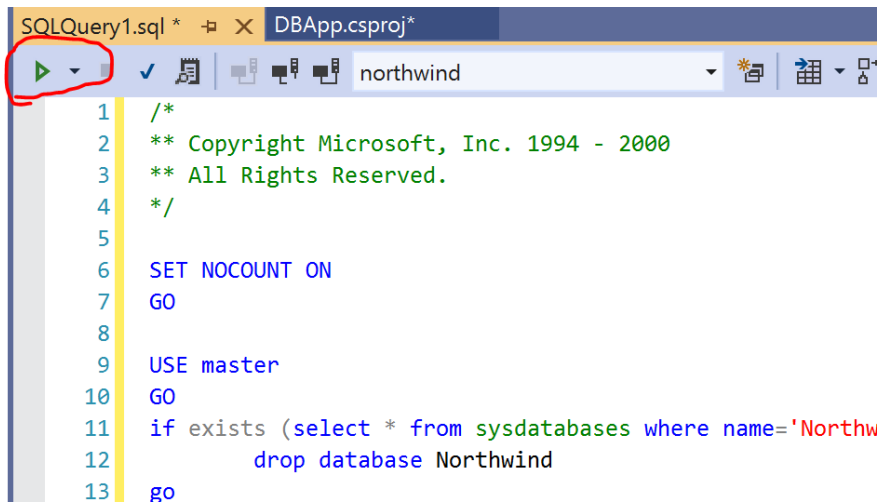


Rechtsklick -neue Abfrage- kopieren sie den SQL Text (RAW) in den Editor von

<https://github.com/microsoft/sql-server-samples/blob/master/samples/databases/northwind-pubs/instnwnd.sql>

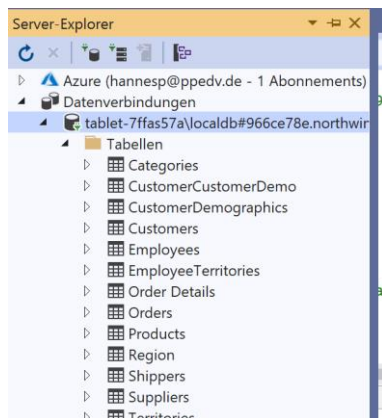


Pro Tipp Google suche „instnwnd“



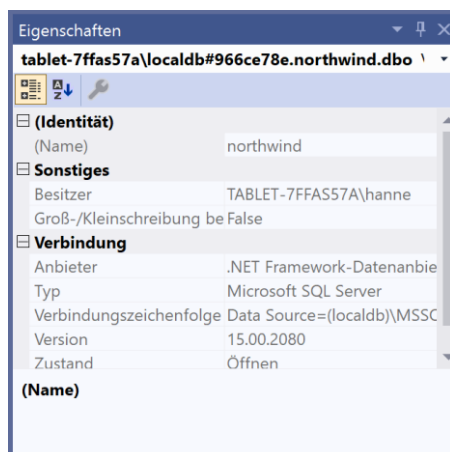
```
1  /*
2  ** Copyright Microsoft, Inc. 1994 - 2000
3  ** All Rights Reserved.
4  */
5
6  SET NOCOUNT ON
7  GO
8
9  USE master
10 GO
11 if exists (select * from sysdatabases where name='Northw
12           drop database Northwind
13 go
```

Führen sie das Script aus, dauert vielleicht 1-2Minuten. Prüfen sie die Datenbank im Visual Studio Server Explorer



Die Welt ist gut. Wenn nicht Mikro an und Hilfe rufen!

Lassen sie den Datenbank Server markiert und wechseln in das Eigenschaftsfenster. Dies ist vermutlich auf der rechten Seite unten in ihrem Visual Studio



Dort kopieren Sie die Verbindungszeichenfolge (Connection String). Er wird so aussehen muss es aber nicht, da es sich um lokale Einstellungen auf Ihrem Computer handelt.

```
Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=Northwind;Integrated Security=True;Pooling=False
```

Fügen sie folgende 3 Nuget Pakete hinzu. Das können sie im Paket Manager oder in der Paket Manager Console tun.



```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

```
Install-Package Microsoft.EntityFrameworkCore.Design
```

Nun können sie die Entity Framework Zugriffsklassen generieren lassen.

Führen sie das Kommando in der Paket Manager Konsole aus mit dem Connection String von vorher

```
Scaffold-DbContext " Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=Northwind;Integrated Security=True; "
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Im Verzeichnis Models werden zahlreiche cs Dateien erzeugt.

Hinweis bei Fehler Zertifikat TrustServerCertificate=true; im Connection String anhängen

Es gibt verschiedene Arten Verbindungszeichenfolgen zu speichern. Da diese in der Regel sensitive Daten wie Passwörter enthalten, auf keinen Fall in den Source Code. In unserem Beispiel wird der angemeldete Benutzer verwendet und wir legen diese Daten in die Datei appsettings.json

```
{
  "ConnectionStrings": {
    "Northwind": "Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=Northwind;Integrated Security=True"
  },
}
```

Im Sinne von .NET Core Dependency Injection wird das Datenbank Zugriffsobjekt (dbContext) in der programm.cs auf den DI Container angemeldet.

Fügen sie folgende Zeile in der Methode Configuration ein

```
builder.Services.AddDbContext<NorthwindContext>(o =>
{
    o.UseSqlServer(builder.Configuration.GetConnectionString("Northwind"));
});
```

Wahrscheinlich werden 2 Stellen rot unterwühlt. Lassen sie sich von Intellisense die fehlenden Namensräume ergänzen

```
builder.Services.AddDbContext<NorthwindContext>(o =>
{
    o.UseSqlServer(builder.Configuration.GetConnectionString("Northwind"));
});
var app = builder.Build();
```

Erstellen sie jetzt im Pages Verzeichnis die Blazor Komponente Kunden.Razor.

Pro Tipp um die Page Live Cycle Methoden automatisch erstellen zu lassen tippen Sie protected override LEERTASTE

```
@code
{
    protected override
    {
        * OnAfterRender(bool firstRender)
        * OnAfterRenderAsync(bool firstRender)
        * OnInitialized()
    }
}
```

```
@page "/Kunden"
@using BlazorLab05.Models
<h1>Kunden</h1>
<ul>
    @foreach (var item in listekunden)
    {
        <li>@item.CompanyName</li>
    }
</ul>
@code
{
    List<Customer> listekunden = new List<Customer>();

    protected override async Task OnInitializedAsync()
    {
        var ef = new NorthwindContext();
        listekunden = ef.Customers.ToList();
    }
}
```

Führen sie das Projekt aus und rufen die Url /kunden auf.

Ergänzen Sie in Zeile 2 die Anweisung um das Entity Datenmodell per DI Injection zu nutzen

```
@inject NorthwindContext db
```

Ändern Sie den Code zum laden der Customer Tabelle in

```
listekunden = db.Customers.ToList();
```

Die Instanz des NorthwindContext Objektes mit New ist dann ebenso nicht mehr nötig.

[Editierbar](#)

Erstellen Sie eine neue Razor Komponente EditKunden.razor im Pages Projekt Verzeichnis.

Ergänzen sie den HTML Teil um das EditForm Control.

```
@inject NorthwindContext db
```

```
<EditForm Model="Kunde" OnSubmit="HandleSubmit">
  <DataAnnotationsValidator />
  @Kunde.CustomerId
  <InputText id="name" @bind-Value="Kunde.CompanyName" class="form-control" />
  <ValidationMessage For="()"=>Kunde.CompanyName />
  <button type="submit" class="btn btn-outline-primary">Submit</button>
</EditForm>
```

Es gibt verschiedene Ansätze um die Customer Entität zu laden. Hier wird ein Cascadierendes Property eingesetzt. Dazu müssen sie in der Datei Kunden.razor folgende grau hinterlegten Änderungen vornehmen. Das Item der listekunden wird zum Property und kann von allen auf allen Hierarchieebenen geschachtelten Komponenten ausgelesen werden.

```
@foreach (var item in listekunden)
{
  <li>
    <CascadingValue Value="item">
      <EditKunden></EditKunden>
    </CascadingValue>
  </li>
}
```

Dieser Cascading Parameter wird in der EditKunden.razor Page Logik nun ausgelesen. Dazu wird die Submit Methode mit der EF Datenbank Update Logik ausgeführt.

```
@code {
  [CascadingParameter]
  protected Customer Kunde { get; set; }

  private async Task HandleSubmit(EditContext edit)
  {
    var isValid = edit.Validate();

    if (isValid)
    {
      var x = (Customer)edit.Model;
      Kunde = x;
      db.SaveChanges();
    }
  }
}
```

Starten sie nun das Projekt in Visual Studio und wechseln sie im Browser auf die Url Kunden.

Ändern Sie den Eintrag Alfreds Futterkiste geringfügig und drücken Sie den zugehörigen Submit Button.

Idee alle Fehler?

Kunden

- ALFKI
- ANATR
- ANTON
- AROUT

Probleme fixen

Wir erkennen

1. zu viele Buttons
2. am besten nur ein Button, wenn Eintrag geändert
3. Eingaben sollen dem Datenbank Schema entsprechen
4. grün eckige Umrandung der Textbox entspricht nicht dem Bootstrap Layout

Ergänzen Sie die Datei im @Code Block

```
EditContext editContext;  
protected override void OnInitialized()  
{  
    editContext = new EditContext(Kunde);  
}
```

Dadurch wird das Formular von Model zum EditContext Property umgestellt. Dies müssen sie neben einigen anderen Änderungen (grau) nun vornehmen

```
<EditForm EditContext="editContext" OnSubmit="HandleSubmit">  
    <DataAnnotationsValidator />  
  
    @Kunde.CustomerId  
    <InputText id="name" @bind-Value="Kunde.CompanyName" class="form-control" />  
    <ValidationMessage For="@(( ) => Kunde.CompanyName)" />  
    @if (context.IsModified())  
    {  
        <button type="submit" class="btn btn-outline-primary">Submit</button>  
    }  
</EditForm>
```

Nun wird erkannt ob ein Input HTML Feld intern als modified markiert ist und entsprechend der Button angezeigt. Um den Modified Status wieder aufzuheben fügen sie noch folgende Zeile in HandleSubmit ein

```
db.SaveChanges();
```

```
editContext.MarkAsUnmodified();
```

Starten Sie das Projekt

Kunden

- ALFKI

Alfreds Futterkiste ✓

Submit

- ANATR

Ana Trujillo Emparedados ✓

- ANTON

Antonio Moreno Taquería ✓

- AROUT

Around the Horn ✓

- BERGS

Berglunds snabbköp ✓

- BLAUS

Bootstrap CSS Fix (optional)

Das User Interface kennt aktuell nur grün. Zunächst wechseln sie in die Datei Customer.cs im Models Verzeichnis und fügen Sie Annotationen hinzu, die dem Tabellen Schema entsprechen. Folgendes Beispiel zum Test (Using per Intellisense ergänzen lassen)

```
public string CustomerId { get; set; }
[Required(ErrorMessage = "{0} erforderlich")]
[StringLength(30, MinimumLength = 3,
ErrorMessage = "Maximal 30 Zeichen")]
[DataType(DataType.Text)]
public string CompanyName { get; set; }
```

Erzeugen sie nun (der Einfachheit halber im Pages Verzeichnis) die Klassendatei

BootstrapCSSProvider.cs. Ergänzen Sie Interface und Code wie folgt. Damit werden die CSS Attribute für gültige und ungültige Eingaben dem Schema aus Bootstrap 4 angepasst.

```
public class BootstrapCSSProvider: FieldCssClassProvider
{
    public override string GetFieldCssClass(EditContext editContext,
        in FieldIdentifier fieldIdentifier)
    {
        var isValid =
!editContext.GetValidationMessages(fieldIdentifier).Any();

        return isValid ? "is-valid" : "is-invalid";
    }
}
```

Um den neuen Provider im Editor formular zu nutzen, ergänzen sie in der Datei EditKunden.Razor

```
protected override void OnInitialized()
{
    editContext = new EditContext(Kunde);
    editContext.SetFieldCssClassProvider(new BootstrapCSSProvider());
}
```

Starten Sie das Projekt. Entfernen sie den Wert aus dem Eingabefeld und verlassen es mit TAB

Kunden

- - CompanyName erforderlich

ALFKI

CompanyName erforderlich

Submit

- ANATR

Ana Trujillo Emparedados

- ANTON

Antonio Moreno Taquería