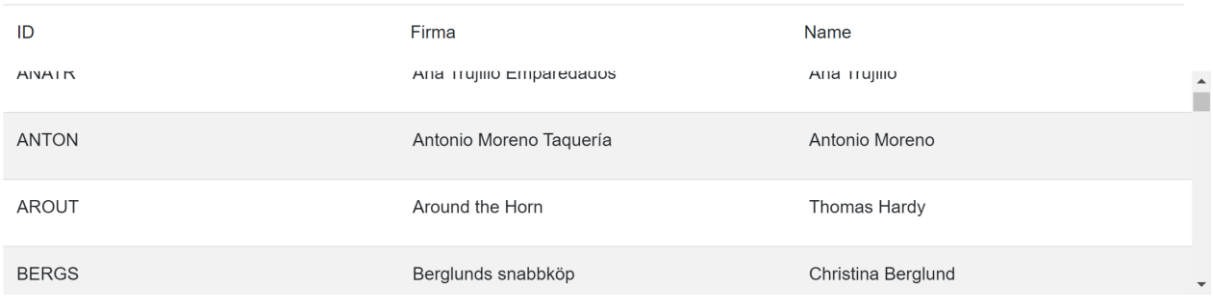
Blazor Lab 7b

Dauer: 25 Minuten

In diesem Lab wird eine universell verwendbare datengebundene scrollbare HTML Tabelle als Templated Component erstellt.



Ausgangspunkt ist ein voreingerichtetes Visual Studio Blazor Server Projekt.

- Installierte Nuget Pakete für Entity Framework
- Datenmodell Northwind
- DI DBcontext per programm.cs

Siehe Lab 5

Das Benutzersteuelement ala Component

Erstellen Sie im Pages Projekt Verzeichnis

- Neue Blazor Kompentente ScrollTable.razor
- CSS Datei ScrollTable.razor.css

Im HTML Code und im C# Teil der Komponente wird nun der Rumpf Code eingefügt. Dieser wird später komplettiert.

Der Typeparam dient dazu beliebige generische Listen übergeben zu können.

Der Bereich Attributes erlaubt beliebige HTML Attribute durchzureichen.

Es werden vier HTML Template Bereiche deklariert, per Renderfragment Parameter.

Um die Daten zu laden wird ein delegierter Funktionsaufruf Loader deklariert.

```
@typeparam TItem
<table @attributes="AdditionalAttributes">
    <thead>
        <tr>
        </tr>
    </thead>
    <tbody >
    </tbody>
</table>
@code { IEnumerable<TItem> items;

    [Parameter] public Func<Task<IEnumerable<TItem>>> Loader { get; set; }
```

```
    [Parameter] public RenderFragment Loading { get; set; }
    [Parameter] public RenderFragment Empty { get; set; }
    [Parameter] public RenderFragment<TItem> Body { get; set; }
    [Parameter] public RenderFragment Head { get; set; }


    [Parameter(CaptureUnmatchedValues = true)]
    public IDictionary<string, object> AdditionalAttributes { get; set; }

    protected override async Task OnParametersSetAsync()
    {
        items = await Loader();
    } }
```

Nehmen Sie sich kurz Zeit den Code zu verstehen. Obwohl noch nicht komplett, wird nun diese Component genutzt. Wechseln sie mit dem Visual Studio Editor in die Ansicht der index.razor Datei.

Zunächst komplettieren sie die C# Logik für das Laden der Daten

```
@page "/"
@using BlazorLab7.Models (je nach Projektname)
@inject northwindContext db

<html…

@code {

    async Task<IEnumerable<Customer>> Load()
    {
        return db.Customers;
    }
    }
```

Nun kann im HTML Bereich der Blazor Seite die Component genutzt, die HTML Templates (Empty,Loading,Head,Body) gefüllt werden. Am besten sie entfernen den vorhanden HTML Code.

```
  <ScrollTable Loader="Load"
                class="table table-striped" >
        <Loading>Loading...</Loading>
        <Empty>
            <h2>keine Daten</h2>
        </Empty>
        <Head>
            <td>ID</td>
            <td>Firma</td>
            <td>Name</td>


        </Head>
        <Body Context="item">
            <td>@item.CustomerId</td>
            <td>@item.CompanyName</td>
            <td>@item.ContactName</td>
        </Body>
    </ScrollTable>
```

Nun muss in der Componente ScrollTable.Razor noch definiert wo und wie zb das Empty Template im HTML Layout endgültig gerendert werden soll. Dies geschieht über die Parameter vom Typ Renderfragment die bereits angelegt worden sind.

Ergänzen sie nun den Code HTML (grau hinterlegt)

```
<table @attributes="AdditionalAttributes"
    <thead>
```

```
        <tr>
            @if (items == null)
            {
            @Loading }
            else if (!items.Any())
            {

            @Empty }
            else
            {
            @Head}
        </tr>
    </thead>
    <tbody style="height: calc( 100vh - 150px);">
        @if (items.Count() > 0)
        {
    @foreach (var item in items)
    {
    <tr> @Body(item)</tr>
}}

        }
    </tbody>
</table>
```

Sie können das Projekt bereits ausführen, werden aber feststellen, das die Tabelle nicht mit fixierten Kopfzeilen scrollbar ist. Dies wird über CSS Styles deklariert. Dazu legen Sie die Datei Scrolltable.razor.css an. (Blazor CSS Isolation!)

Ergänzen Sie diese Datei im Visual Studio Editor

```
tr {
    width: 100%;
    display: inline-table;
    height: 60px;
    table-layout: fixed;
}

tbody {
    overflow-y: scroll;
    position: absolute;
}
```