Blazor Lab 4 Httpclient

In diesem Lab werden Sie eine externen Web API Service konsumieren und als Liste darstellen. Dabei kommen beide Blazor Varianten zum Einsatz.

Erstellen Sie ein neues Visual Studio Blazor Server Projekt mit dem Namen BlazorLab4.

REST Todo

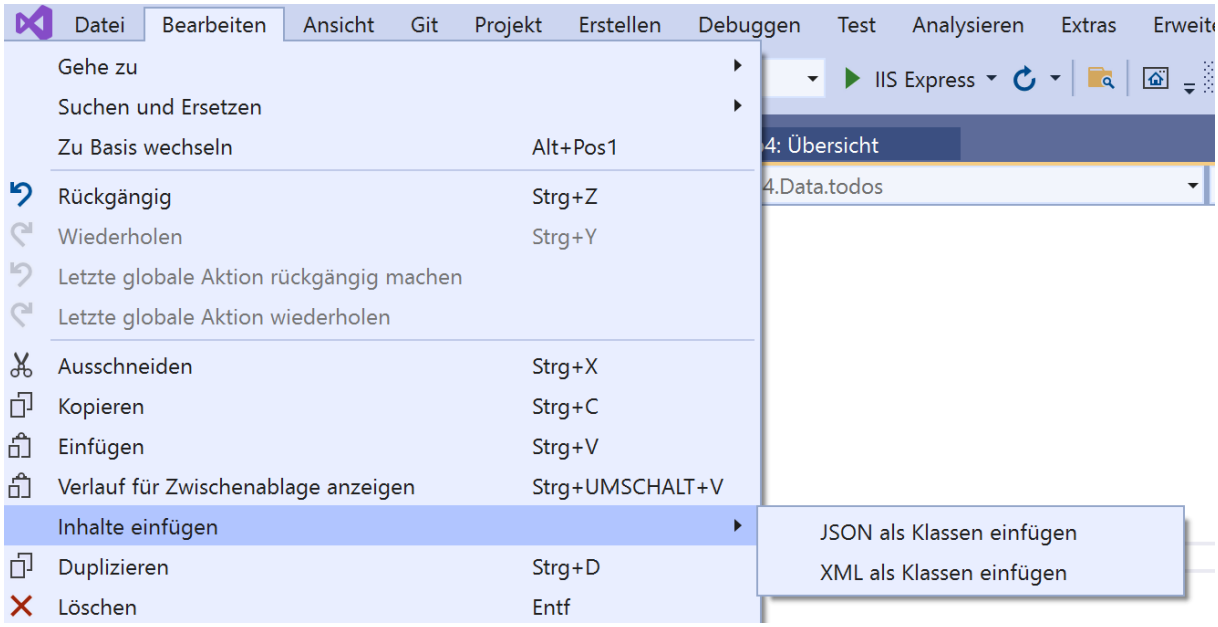
Rufen sie im Browser die Url des Fake Services auf

https://jsonplaceholder.typicode.com/todos

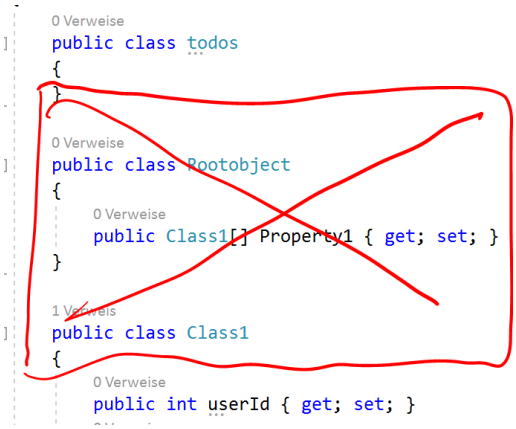
Kopieren sie den Text aus dem Browser in die Zwischenablage [STRG] + [A] , [STRG]+[C]

In Visual Studio Projekt im Verzeichnis Data eine neue Klasse todos.cs einfügen. Im Editor den Cursor unterhalb der Klasse platzieren (nach der } aber innerhalb des Namesspaces).

Wählen Sie im Menü -Bearbeiten – Inhalte einfügen- JSON als Klasseneinfügen (Englisch Paste special)



Entfernen sie den durchgestrichenen Bereich aus dem Code der Klasse und speichern sie.



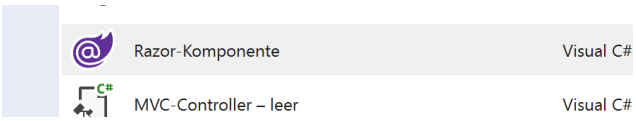
Sie können den Code auch tippen, aber er sollte auch so wie folgt aussehen

```
public class todos
 {
      public int userId { get; set; }
      public int id { get; set; }
      public string title { get; set; }
      public bool completed { get; set; }
 }
```

Editieren Sie die Datei im Blazor Projekt Stammverzeichnis programm.cs und fügen die grau hinterlegte Code Zeile hinzu

```
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();
builder.services.AddHttpClient();
```

Wählen sie per Rechtsklick im Verzeichnis Pages die Option hinzufügen und erstellen so eine neue Blazor Komponente mit dem Namen Todorest.razor



Deklarieren Sie den HTML Teil um die Daten als Liste darzustellen. Statt einem @foreach nun per Virtualize Component um die virtualiserte Darstellung der Liste zu nutzen und damit eine schneller Blazor App zu bieten.

```
<h3>Rest ToDo</h3>
<button @onclick="laden">laden</button>

    <ul class="list-group">
        <Virtualize Items="@TodoListe" Context="item">
            <ItemContent>  <li class="list-group-item">@item.title</li></ItemContent>
            <Placeholder>laden...</Placeholder>
        </Virtualize>
    </ul>
```

Im top Code Block der Seite ergänzen sie Route, Usings und per DI den Zugriff auf die HttpClient Factory

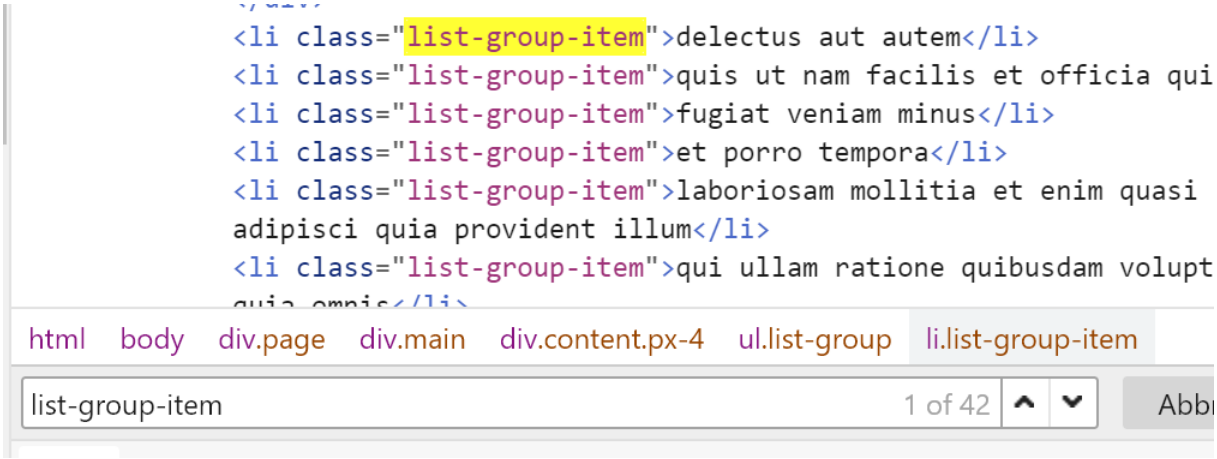
```
@page "/todo"
@using BlazorLab4.Data
@using System.Text.Json
@inject IHttpClientFactory http
```

Im Page Logik Code Block wird eine Instanz des HttpClient aus der Factory generiert, ein http get abgesetzt und im Erfolsfall per Json deserialisierer in eine generische Liste (todoliste) um gewandelt.

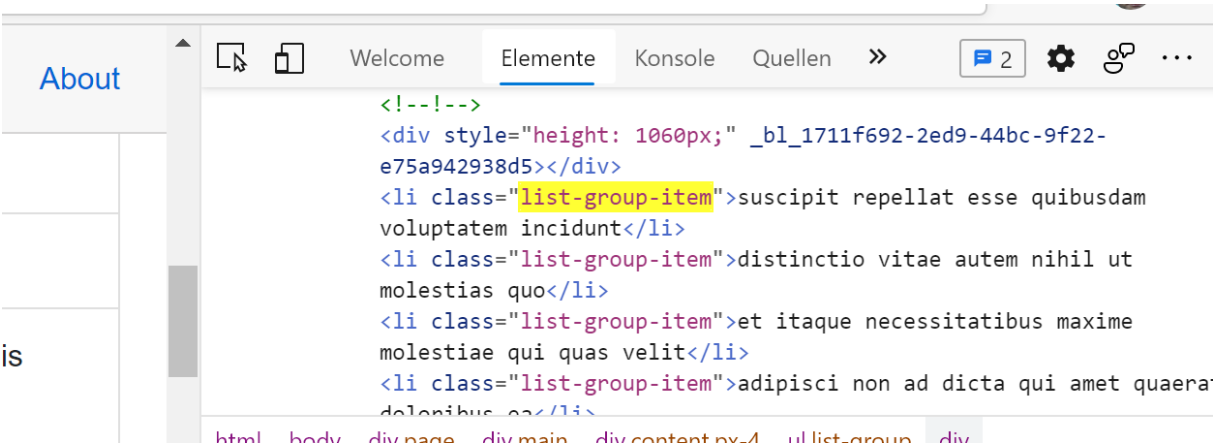
```
@code {
    public List<todos> TodoListe { get; set; } = new List<todos>();
    void laden()
    {
        var client = http.CreateClient();
        var response =
client.GetAsync(@"https://jsonplaceholder.typicode.com/todos").Result;
        if (response.IsSuccessStatusCode)
        {
            var s = response.Content.ReadAsStringAsync().Result;
            TodoListe = JsonSerializer.Deserialize<List<todos>>(s);
        }

    }
```

Starten Sie ihr Projekt wechseln im Browser die URL auf /todo und drücken den Lade Button.

Um die Funktion der virtualsieren Liste auch nachverfolgen zu können, drücken Sie F12 und öffen die Browser Developer Tools. Im Reiter *Elemente* können sie den HTML Source der Listen Einträge als LI Elemente sehen. Ein erster Vergleich mit dem Resulat des Fake API Service zeigt, das 200 Einträge geladen sind, aber nur 42 im Browser gerndert wurden



Lassen Sie die F12 Tools offen und scrollen im Browser nach unten. Sie werden beobachten, das sich der Text der LI Einträge an identer Stelle ändert, sobald ca 30 Element gescrollt wurde



Rewrite als PWA

Eine Blazor Webassembly App lässt sich mit entsprechender Konfiguration auf Klick auch installieren und ohne Browser nutzen. Das Beispiel von vorhin wird nun zu einer Progressive Web App umgeschrieben.

Erstellen Sie mit Visual Studio ein neues Projekt vom Typ Blazor WebAssembly und aktivieren sie den Haken für Proressive Webanwendung.

Kopieren sie aus dem Dateiexlplorer aus dem vorangegangen Projekt die Datei todos.cs in das Projektstammverzeichnis. Passen sie den Namensraum an.

```
namespace BlazorLab4PWA
{
    public class todos
    {
         public int userId { get; set; }
        public int id { get; set; }
        public string title { get; set; }
        public bool completed { get; set; }
    }

}
```

Kopieren Sie die Datei todorest.razor in das Pages Verzeichnis und ändern den Code wie folgt

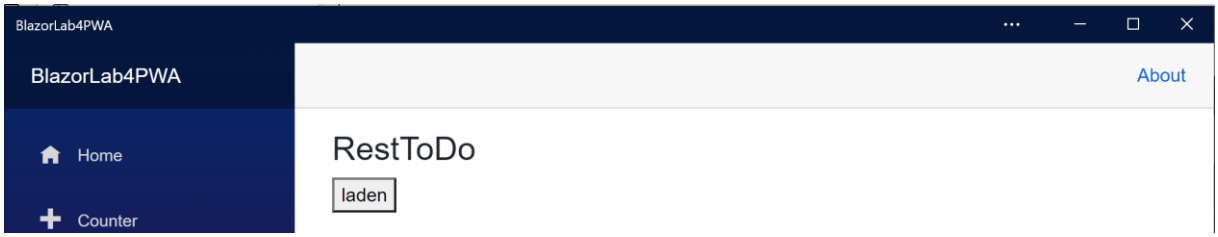
```
@page "/todo"
@using System.Text.Json
@inject HttpClient Http

<h3>RestToDo</h3>
<button @onclick="laden">laden</button>

<ul class="list-group">
    <Virtualize Items="@TodoListe" Context="item" ItemSize="25">
        <ItemContent>  <li class="list-group-item">@item.title</li></ItemContent>
        <Placeholder>laden...</Placeholder>
    </Virtualize>
</ul>
```

```
@code {
    public List<todos> TodoListe { get; set; } = new List<todos>();
  async  void laden()
    {
        TodoListe =await
Http.GetFromJsonAsync<List<todos>>(@"https://jsonplaceholder.typicode.com/todos");
        StateHasChanged();
    }

}
```

Der HttpClient ist völlig unterschiedlich implementiert in der Server und WebAssembly Variante. Der Code wird dadurch eminent kürzer und der Deserialiserer obsolet.

Starten Sie die Anwendung. Der Start dauert merklich länger.

In der Browser URL Zeile befindet sich rechts ein Symbol, je nach Browser unterschiedlich, mit dem sie die PWA App installieren lässt. Tun sie das. Sie finden die App anschliessend im Windows Menü und können diese auch ohne Visual Studio benutzen.



Die Konfiguration der Manifest Datei kann im Verzeichnis wwwroot in manfiest.json gefunden werden. PWA ist nicht Bestandteil dieses Kurses.