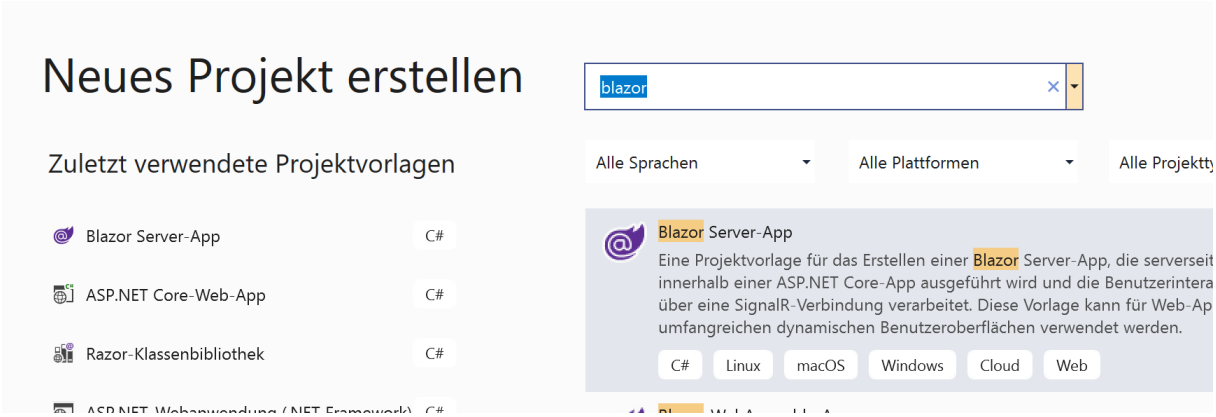
Blazor Lab 3 Todo Liste

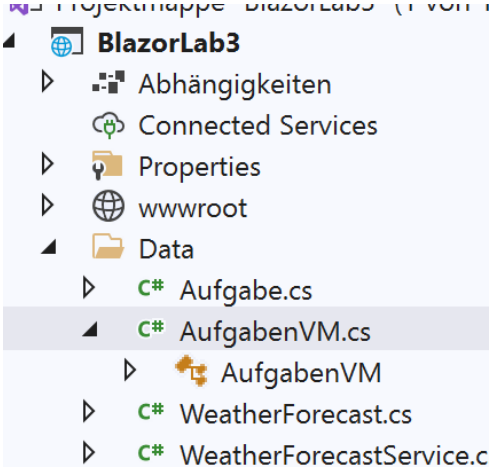
In dieser Übung werden Sie Schritt für Schritt einen ToDo App erstellen. Das Beispiel folgt dem Ansatz Daten in eine Viewmodel Klasse zu persistieren. Die Daten werden im Browser des Benutzers dauerhaft abgelegt.

Erzeugen sie ein neues Blazor Server Projekt mit Visual Studio mit dem Namen BlazorLab3



Datenmodell

Fügen Sie im Model Verzeichnis des Blazor Projektes die Klassen Aufgabe.cs und AufgabenVM.cs hinzu. Rechtsklick Projekt- hinzufügen- Klasse

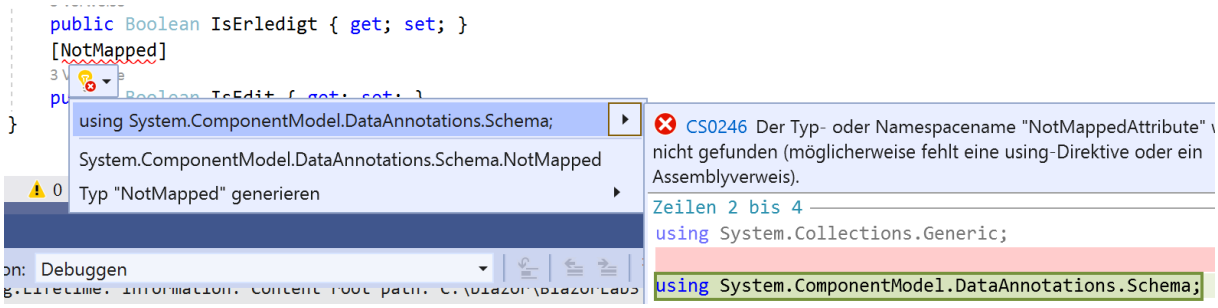


Editieren Sie in Visual Studio die Klasse Aufgabe.cs und fügen (grau hinterlegt) sie wie folgt Propertys hinzu. Der Visual Studio Editor hilft die Aufgabe schneller zu erledigen, indem sie PROP [TAB] [TAB] tippen und sich dann per TAB vom Typ in das Namensfeld bewegen.

```
public class Aufgabe
  {
      public int ID { get; set; }
      public string Text { get; set; }
      public DateTime Erledigt { get; set; }
      public Boolean IsErledigt { get; set; }
      [NotMapped]
      public Boolean IsEdit { get; set; }
  }
```

Das Attribut NotMapped erfordert den Namensraum
`System.ComponentModel.DataAnnotations.Schema;`

Auch hier sollten Sie sich von Editor helfen lassen, indem Sie den Mauszeiger über die rot unterwellte Stelle bewegen.



Wechseln Sie zum Editieren in die Klasse AufgabenVM. Das Viewmodell hält eine Liste der Aufgaben und Methoden zum speichern und aktualisieren.

```
public List<Aufgabe> ListeAufgaben { get; set; } = new List<Aufgabe>();

public void UpdateSpeichern(Aufgabe a)
        {
            ListeAufgaben[ListeAufgaben.FindIndex(x => x.ID == a.ID)] = a;
        }

public void NeuSpeichern(Aufgabe a)
        {
            var id = ListeAufgaben.Max(x => (int?)x.ID) ?? 0;
            a.ID = id + 1;
            ListeAufgaben.Add(a);
        }
```

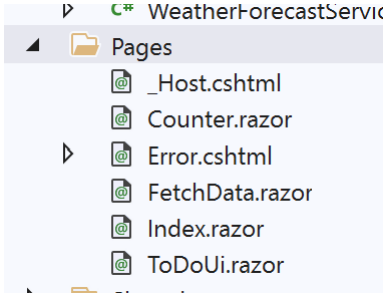
Öffne Sie im Blazor Projekt in die Datei programm.cs aus dem Stammverzeichnis. Ergänzen sie den Code (grau hinterlegt) um das Viewmodell dem DI Container bekannt zu machen.

```
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();
builder.services.AddScoped<Data.AufgabenVM>();
```

Am besten schließen Sie die.cs Dateien nun.

Blazor Page

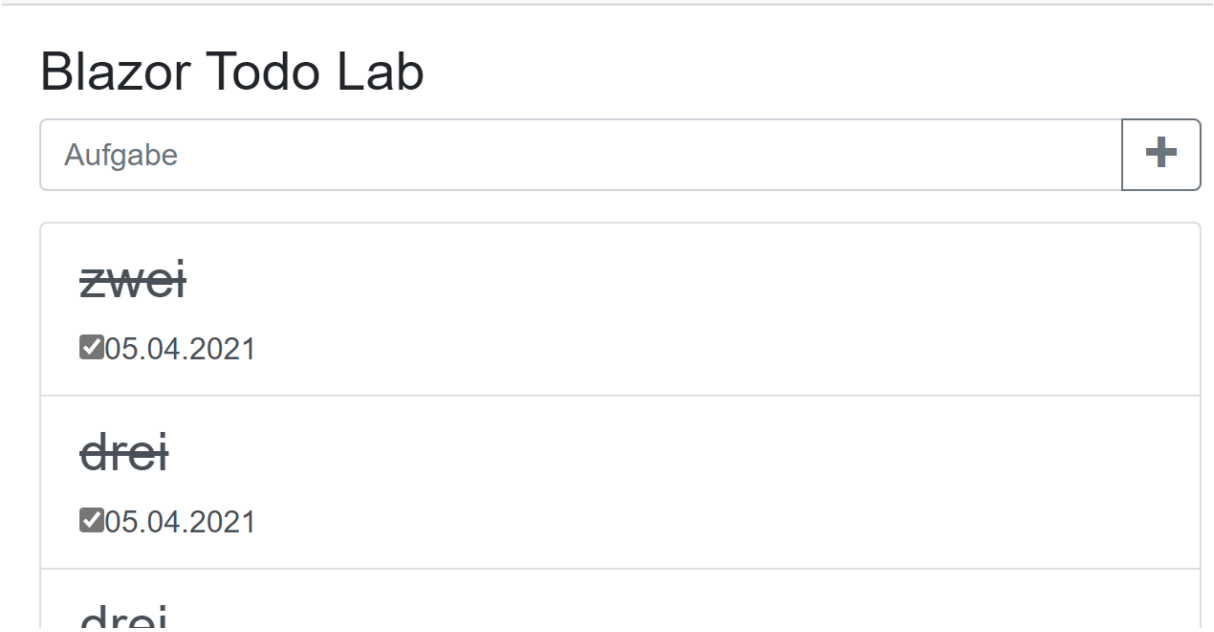
Fügen Sie dem Projekt eine Datei vom Typ Blazor Component hinzu. Rechtklick auf das Pages Verzeichnis- Hinzufügen- Blazor-Komponente – Als Namen wählen Sie ToDoUI.razor.



Das Userinterface für TODO beinhaltet folgende Funktionen

- neue Aufgabe
- Aufgabe als Erledigt markieren.
- Aufgabe editieren – in place

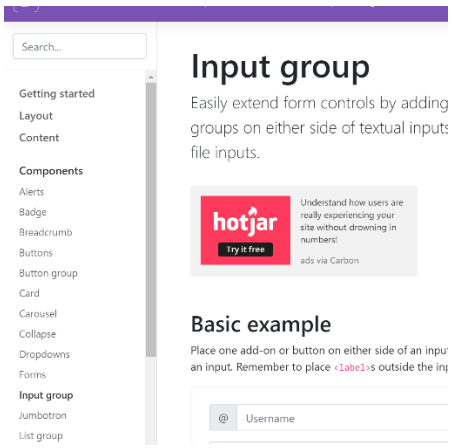
Als UI CSS Framework kommt Bootstrap 4 zum Einsatz



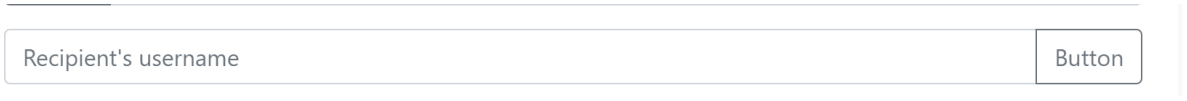
Wenn Sie in Bootstrap unerfahren sind und Zeit sparen wollen können sie den Code von der Website

[Introduction · Bootstrap v5.1 (getbootstrap.com)](getbootstrap.com) (zu erreichen über das Website Menu)

Ergänzen Sie den HTML Code um das Eingabefeld. In Bootstrap Suchdialog über „input group“ zu finden



Scrollen Sie bis ein passendes Design erscheint



Dann kopieren sie den HTML Code und verändern ihn

```
<div class="input-group mb-3">
  <input type="text" class="form-control" placeholder="Recipient's username" aria-label="Recipient's us
  <div class="input-group-append">
    <button class="btn btn-outline-secondary" type="button" id="button-addon2">Button</button>
  </div>
</div>
```

Ändern Sie den HTMl Code wie folgt oder tippen Sie ihn einfach ab.

```
<h3>Blazor Todo Lab</h3>
<div class="row">
    <div class="col">
        <div class="input-group mb-3">
            <input type="text" class="form-control" placeholder="Aufgabe"/>
            <div class="input-group-append"/>
                <button class="btn btn-outline-secondary" type="button" id="button-
addon2">
                    <span class="oi oi-plus"></span>
                </button>
            </div>
        </div>
    </div>
</div>


<div class="row">
    <div class="col">
        <div class="list-group">
            @foreach (var item in VM.ListeAufgaben)
            {
                <div class="list-group-item list-group-item-action" @key="item.ID">
                    @if (item.IsEdit)
                    {
                        <input @bind="item.Text" class="form-control" @onfocusout="()
=> Aktualisiere(item)" />
                    }
                    else
                    {
                        <h3 style='@(item.IsErledigt ? "text-decoration:line-through"
: "")'
                            @onclick="() => item.IsEdit = true">@item.Text</h3>
                        <input type="checkbox" checked="@item.IsErledigt"
                               @onchange="(_)=> { item.IsErledigt=(bool)_.Value;
Aktualisiere(item); }" />

                        @item.Erledigt.ToShortDateString()
                    }

                </div>
            }
        </div>

    </div>
</div>
```

Im HTML Dialog darunter werden dann die Aufgaben als Bootstrap List-Group angeordnet. Auch das findet sich in der Boostrap Doku

```
<div class="row">
    <div class="col">
        <div class="list-group">

      <div class="list-group-item list-group-item-action" >
              <h3>Text</h3>
```

```
                <input type="checkbox"/>Aufgabe
      </div>
   </div>
 </div>
</div>
```

Business Logik Todo

Die Blazor Komponenten wird nun zur Blazor Page mit Route und der benötigen Logik um Aufgaben per Viewmodell Binding zu visualisieren und editieren.

Fügen Sie folgende Code am Anfang der Datei ein

```
@page "/todo"
@using BlazorLab3.Data
@using  Microsoft.AspNetCore.Components.Server.ProtectedBrowserStorage
@inject AufgabenVM VM
@inject ProtectedLocalStorage _ProtectedLocalStorage
```

Am Ende der Datei todoui.razor hinter dem @Code Block schreiben sie die Methode für eine neue Aufgabe hinzu und für die aktualisierung.

```
string NeuAufgabe;
void Neu()
    {
        Aufgabe a = new Aufgabe();
        a.Text = NeuAufgabe;
        a.Erledigt = DateTime.Now.AddDays(2);
        VM.NeuSpeichern(a);
    }

void Aktualisiere(Aufgabe a)
    {
        a.IsEdit = false;
        VM.UpdateSpeichern(a);
    }
```

Das reicht um sich nun dem UI final zuzuwenden. Sie editieren ergänzen den den HTML Code (grau hinterlegt)

```
<div class="row">
    <div class="col">
        <div class="input-group mb-3">
            <input type="text" class="form-control" placeholder="Aufgabe"
@bind="NeuAufgabe"/>
            <div class="input-group-append">
                <button class="btn btn-outline-secondary" type="button" id="button-
addon2" @onclick="Neu">
                    <span class="oi oi-plus"></span>
                </button>
            </div>
        </div>
    </div>
</div>
<div class="row">
    <div class="col">
        <div class="list-group">
            @foreach (var item in VM.ListeAufgaben)
            {
                <div class="list-group-item list-group-item-action" @key="item.ID">
                    <h3 style='@(item.IsErledigt ? "text-decoration:line-through"
: "")'
```

```
                        >@item.Text</h3>
                    <input type="checkbox" checked="@item.IsErledigt"
                           @onchange="(_)=> { item.IsErledigt=(bool)_.Value;
Aktualisiere(item); }" />
                    @item.Erledigt.ToShortDateString()


                </div>
            }
        </div>

    </div>
</div>
```

Starten Sie nun das Projekt in Visual Studio wechseln die Url zu *todo* und betrachten das Ergebnis im Browser. Sie können neue Einträge einfügen und per Checkbox diesen in der Liste als erledigt markieren.

Daten persistieren und Editieren

Im lokalen Browser gibt es den Localstorage in die nun die Todo Einträge abgelegt werden.

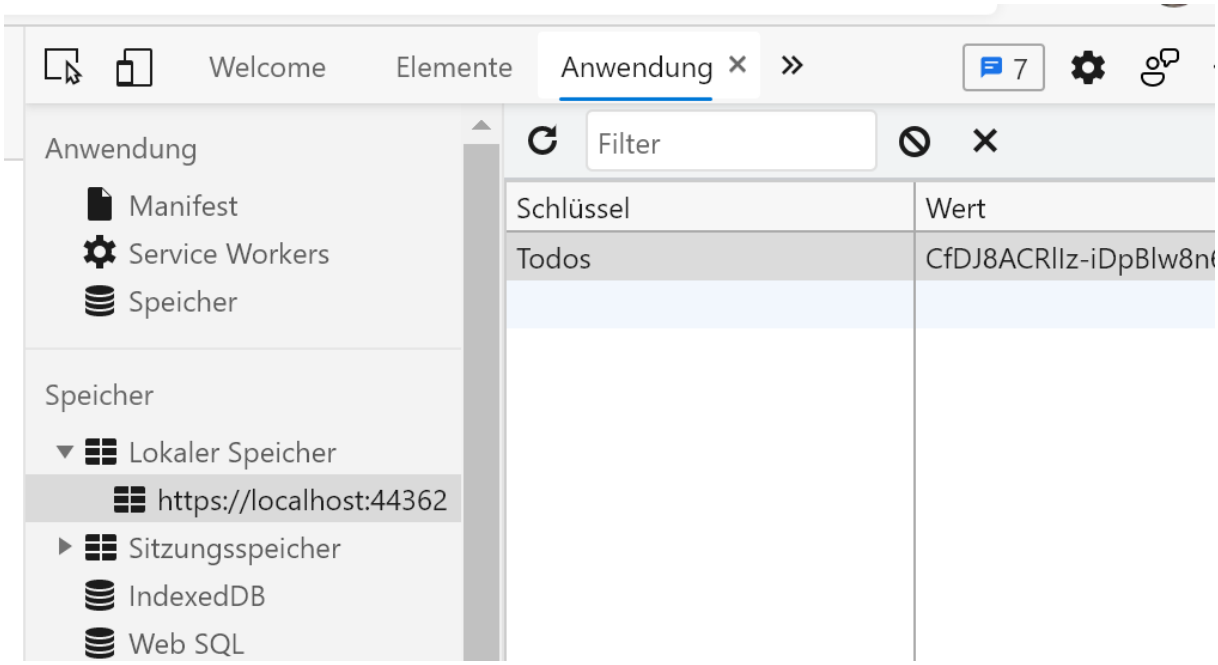
```
    async void AufgabeInLocalStorage(Aufgabe a)
    {
        await _ProtectedLocalStorage.SetAsync("Todos", VM.ListeAufgaben);
    }

    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            var lokal = await _ProtectedLocalStorage.GetAsync<List<Aufgabe>>("Todos");
            if (lokal.Success)
            {
                VM.ListeAufgaben = lokal.Value;
                StateHasChanged();
            }
        }
    }
```

Für den Lerneffekt können Sie später testweise die Zeile StateHaschanged auskommentieren mit //

Dazu muss in den bestehenden Code noch jeweils eine Zeile eingefügt werden (grau)

```
  void Aktualisiere(Aufgabe a)
    {
        a.IsEdit = false;
        VM.UpdateSpeichern(a);
        AufgabeInLocalStorage(a);
    }
    void Neu()
    {
        Aufgabe a = new Aufgabe();
        a.Text = NeuAufgabe;
        a.Erledigt = DateTime.Now.AddDays(2);
        VM.NeuSpeichern(a);
        AufgabeInLocalStorage(a);
    }
```

Starten Sie nun das Projekt erneut und fügen sie 2-3 Aufgaben hinzu. Während die Anwendung noch läuft drücken Sie im Browser (hier EDGE) F12- Reiter Anwendung- Dort sehen Sie den Lokalen Speicher dieser Anwendung und könnten ihn auch wieder leeren.



Als nächstes erweitern sie den HTMl Code um die Blazor Logik für Inplace Editing einer Aufgabe einzubauen

```
<div class="list-group-item list-group-item-action" @key="item.ID">
    @if (item.IsEdit)
    {
        <input @bind="item.Text" class="form-control" @onfocusout="() =>
Aktualisiere(item)" />
    }
    else
    {
        <h3 style='@(item.IsErledigt ? "text-decoration:line-through" : "")'
            @onclick="() => item.IsEdit = true">@item.Text</h3>
        <input type="checkbox" checked="@item.IsErledigt"
               @onchange="(_)=> { item.IsErledigt=(bool)_.Value; Aktualisiere(item);
}" />
        @item.Erledigt.ToShortDateString()
    }

</div>
```

Starten Sie das Projekt und klicken auf einen Texteintrag in der Aufgabenliste

Blazor Todo Lab

Aufgabe +

zweihundertausen

~~drei~~

05.04.2021

drei

05.04.2021