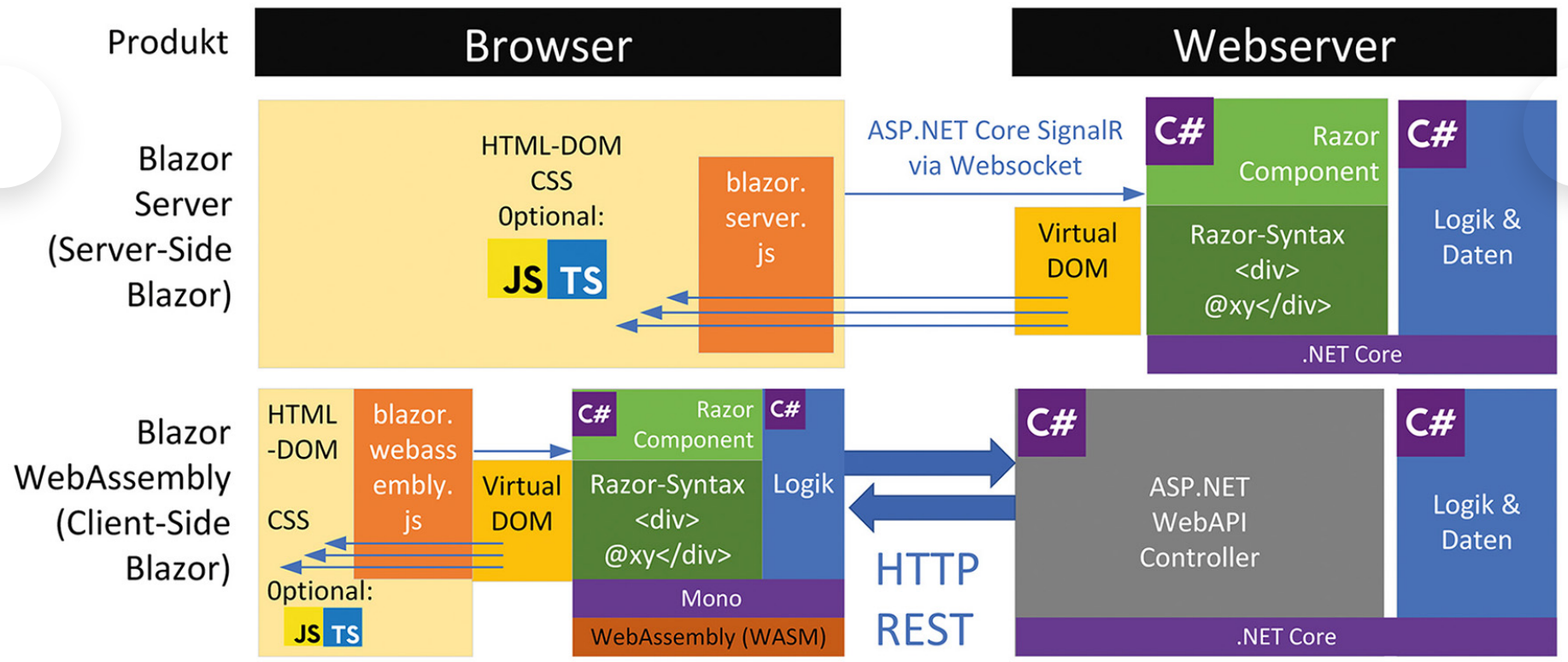
ASP.NET Core Blazor WebAssembly und Blazor Server

Spickzettel zu Blazor

von *Holger Schwichtenberg*

Softwarearchitektur



Wichtigste Namensräume für Blazor-Anwendungen

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Forms;
using Microsoft.AspNetCore.Components.Rendering;
using Microsoft.AspNetCore.Components.Routing
using Microsoft.AspNetCore.Components.Web;
using Microsoft.JSInterop;
using System.Net.Http;
using System.Net.Http.Json;
```

Startup-Code für Blazor WebAssembly

index.html:

```
<app>Loading …</app>
<script src="_framework/blazor.webassembly.js"></script>
```

Verzögerter Start mit autostart="false" und window.Blazor.start()

Program.cs:

```
var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("app");
await builder.Build().RunAsync();
```

Startup-Code für Blazor Server

_host.cshtml:

```
<app><component type="typeof(App)" render-mode="ServerPrerendered" /></app>
<script src="_framework/blazor.server.js"></script>
```

Verfügbare Modi: Static, Server, ServerPrerendered

Startup.cs:

```
services.AddServerSideBlazor().AddCircuitOptions(o =>
   {    if (this.Env.IsDevelopment())
        {     o.DetailedErrors = true;    }
   });
```

Globale Imports _imports.razor

Einbindung von Namensräumen mit @ using oder Dependency Injections (@ inject), die in allen Razor Components zur Verfügung stehen sollen. Sonstiger C#- sowie HTML-Code sind nicht erlaubt.

```
@using MeineBusinessLogik
@using Microsoft.AspNetCore.Components
@using Microsoft.JSInterop
@inject NavigationManager NavigationManager
@inject IJSRuntime JsRuntime
```

Razor Component ohne Code-Behind

Name razor:

```
<div>…</div>
@code { … }
```

Razor Component mit Code-Behind per Partial Class

Name.razor:

```
<div>…</div>
```

Name.razor.cs:

```
namespace Ordner.Unterordner
{ public partial class Name  { … }  }
```

Razor Component mit Code-Behind per Vererbung

Name.razor:

```
@inherits Namensraum.NameModel;
<div>…</div>
```

Name.razor.cs:

```
namespace Namensraum
{ public partial class NameModel  { … }  }
```

Razor Component nur mit C#-Code

Name.cs:

```
public partial class CodeOnlyComponent : ComponentBase
{  protected override void BuildRenderTree(RenderTreeBuilder builder)  { builder.AddMarkupContent(0, "<h2>Text</h2>"); } }
```

Routing und URL-Parameter

Seitendirektive mit Route:

```
@page "/a/b"
```

Seitendirektive mit Route und Parametern:

```
@page "/a/b/{x}/{y}"
```

Seitendirektive mit Route und typisierten Parametern:

```
@page "/a/b/{x:int}/{y:guid}"
```

Erlaubte Datentypen:

```
bool, datetime, decimal, double, float, guid, int, long
```

Optionale Parameter (mit ?) und Standardwerte sind bei Blazor nicht erlaubt, Mehrfachrouten aber möglich:

```
@page "/a/b/{x}/{y}"
@page "/ab/{x}/{y}"
@page "/ab/{x}"
```

Properties für Parameter:

```
[Parameter] public string x { get; set; }
[Parameter] public Guid y { get; set; }
```

Initialisierungswerte von Parameter-Properties (z. B. [Parameter] public string x { get; set; } = "n/a") werden immer beim Initialisieren der Parameter mit dem Standardwert für den jeweiligen Datentyp überschrieben, wenn die Parameter in dem URL nicht gesetzt wurden.

Navigation

Navigation per <a>-Tag:

```
<a href="/a/b/123">Text</a>
```

Navigation per <NavLink> (zuletzt geklickter Link erhält CSS-Klasse "active"):

```
<NavLink class="nav-link" href="/a/b/123">Text</NavLink>
```

Navigation im Code:

```
this.NavigationManager.NavigateTo("/a/b/123");
```

Aktueller URL:

```
this.NavigationManager.Uri
```

Komponenteneinbettung

```
<NameDerKomponenteOhneRazor Parameter="@Ausdruck">
```

Razor-Ausdrücke

Ausgabe von Variable, Field oder Property (HTML-Encoded):

```
@ausgabe oder @this.ausgabe
```

Ausgabe von HTML-Code (nicht HTML-Encoded):

```
@((MarkupStringMarkupString)ausgabe)
```

Durch das Leerzeichen ist der Punkt reiner Text:

```
Dies ist die @anz. Seite!
```

Hier werden jeweils Klammern @(…) zwingend gebraucht:

```
Dies ist die @(anz).Seite!
Ergebnis: @(anz * 10 + 2)
@(new Autor("Holger Schwichtenberg").GetInfo<string>())
```

Escape für @:

```
Die Variable @@anz enthält @anz.
```

Bedingte Formatierung

Dynamische Festlegung einer CSS-Klasse:

```
@{ string cssklasse = "bg-success"; }
<div class="@cssklasse">Text</div>
```

Bedingte Festlegung einer CSS-Klasse:

```
<div class="@(anz < 5 ? "bg-success" : "bg-danger")">Text</div>
```

Wenn cssklasse null ist, verschwindet das class-Attribut:

```
@{ cssklasse = null; }
<div class="@cssklasse">Text</div>
```

aber nicht, wenn es weitere CSS-Klassen gibt:

```
<div class="@cssklasse kursiv">kursiver Text</div>
```

Bedingter Style: nur Wert dynamisch:

```
<div style='color:@( anz<5 && orte != null ? "red" : "green" )'>
```

Bedingter Style: Attribut und Wert dynamisch:

```
<div style='@((anz <5 && orte != null ? "color:red" : "Background-color:red"))'>
```

Razor-Blöcke

```
@* Dies ist ein mehrzeiliger
 Kommentarblock *@
@code{  // Codeblock
  int schrittweite = 2;
  for (int i = 0; i < anz; i += schrittweite)
  {  <div>@start.AddDays(i)</div>  }
 }
```

Block mit reinem Text (ohne HTML-Tag) mit <text>:

```
… {
   <text> Anzahl: @anz
   noch mehr reiner Text </text>
}
```

oder mit @:

```
… {
   @: Anzahl: @anz   @: noch mehr reiner Text
} // Klammer muss in diesem Fall in getrennte Zeile!
```

Razor-Funktionen

```
@functions
{
 string GetProzent(int anz)  {  return anz.ToString() + "%";  }
}
Erledigt sind @GetProzent(10)!
```

Razor-Bedingungen

```
@if (anz < 100)
 {<text>reiner Text und Ausdruck @anz</text> }
else
 { <div>Tag und Ausdruck @anz</div>}
```

Razor-Schleifen

```
@for (int i = 0; i < anz; i++)
{  <div>@start.AddDays(i)</div> }
@while (anz*10 < 100)
{ anz++;
 <div>@(anz*10)% erledigt</div> }
@foreach (var ort in orte)
{ <a href="/flug/@ort">@ort</a> }
```

Datenbindung

Deklaration Property (oder Field):

```
public decimal Zahl { get; set; } = 1.23m;
```

Zwei-Wege-Bindung im Eingabesteuerelement mit @bind:

```
<input type="number" @bind="Zahl" />
```

Komponentenlebenszyklusereignisse

```
protected override void OnInitialized()  {  }
protected async override Task OnInitializedAsync()  {  }
protected override void OnParametersSet()  {  }
protected async override Task OnParametersSetAsync()  { }
protected override void OnAfterRender(bool first)  {  }
protected async override Task OnAfterRenderAsync(bool first)  {  }
```

Binding an DOM-Ereignisse

Ereignisbehandlung ohne Parameter:

```
<li @onclick="Methode">
```

Ereignisbehandlung mit Parameter:

```
<li @onclick="() => Methode(x,y)">
```

Ereignisbindung mit Ereignisdetails an asynchrone Methode:

```
<button @onclick="async(e) => await Methode(x,y, e)"> Klick mich</button>
```

Ereignisbindung mit Unterdrückung der Standardaktion:

```
<input value="@counter" @onkeypress="Methode"
@onkeypress:preventDefault />
```

Ereignisbindung mit Unterdrückung der Ereignisweitergabe:

```
<div @onclick="Methode" @onclick:stopPropagation="true">
```

Eingabesteuerelemente

Tabelle 1 zeigt die in Blazor eingebauten Komponenten für Formulare.

```
private EditContext editContext { get; set; }
private Person p { get; set; };
this.editContext = new EditContext(this.p);
var isValid = editContext.Validate();
…
<EditForm EditContext="@editContext" OnValidSubmit="@Save" OnInvalid-Submit="@ShowValidationError">
<DataAnnotationsValidator />
<InputText id="C_Name" @bind-Value="p.Name" class="form-control" placeholder="Your Name (max 20 letters)" />
<ValidationMessage For="@(() => p.Name)" />
<InputSelect id="C_JobTitle" @bind-Value="p.JobTitle_IDString" class="form-control">
@foreach (var s in Enum.GetValues(typeof(JobTitle)).Cast<JobTitle>().ToList())
  {
  <option value="@((int)s)">@s.ToString() (@((int)s))</option>
  }
</InputSelect>
<button type="submit" class="btn …">Save</button>
</EditForm>
```

Razor Component	HTML-Repräsentation
<EditForm>	<form>
<InputText>	<input>
<InputTextArea>	<textarea>
<InputSelect>	<select>
<InputNumber>	<input type="number">
<InputCheckbox>	<input type="checkbox">
<InputDate>	<input type="date">
<DataAnnotationsValidator>	-
<ValidationSummary>	<ul class="validation-errorsvalidation-errors"><li class="validation-message">…
<ValidationMessage>	…

Tabelle 1: Eingabesteuerelemente

Aufruf von C# zu JavaScript

JavaScript-Funktion mit Rückgabewert aufrufen:

```
bool a = await JSRuntime.InvokeAsync<bool>("confirm", "text");
```

JavaScript-Funktion ohne Rückgabewert aufrufen:

```
await JSRuntime.InvokeVoidAsync("alert", "Antwort war: " + a);
```

Übergabe einer HTML-Elementreferenz:

```
<div id="eRef" @ref="eRef">@output</div>
private ElementReference eRef;
await JSRuntime.InvokeVoidAsync("JSFunktion", eRef);
```

Übergabe einer sonstigen Objektreferenz:

```
private DotNetObjectReference<Klasse> objRef;
objRef = DotNetObjectReference.Create(this);
await JSRuntime.InvokeVoidAsync("JSFunktion", objRef);
```

Eigene JavaScript-Dateien dürfen nicht in Komponenten, sondern nur global in index.html bzw _host.cshtml eingebunden werden!

```
<script src="MeinSkript.js"></script>
```

Aufruf von JavaScript zu C#

Statische .NET-Methode aufrufen:

```
DotNet.invokeMethodAsync("assembly", "methode", daten);
```

Instanzmethode aufrufen:

```
function JSFunktion(dotnetHelper) {
  dotnetHelper.invokeMethodAsync("NetFunktion", daten); }
```

NetFunktion muss mit [JSInvokable] annotiert sein!

Dr. Holger Schwichtenberg, alias der „DOTNET-DOKTOR“, gehört zu den bekanntesten Experten für .NET und Webtechniken in Deutschland. Er hat zahlreiche Fachbücher veröffentlicht und spricht regelmäßig auf Fachkonferenzen wie der BASTA!. Sie können ihn und seine ebenso kompetenten Kollegen für Entwicklungsarbeiten, Schulungen, Beratungen und Coaching buchen.

Webadressen

Offizielle Website für Blazor:

dotnet.microsoft.com/apps/aspnet/web-apps/blazor

Quellcode und Issues:

github.com/aspnet/AspNetCore

Dokumentation (englische Version empfohlen!):

docs.microsoft.com/en-us/aspnet/core/blazor

Deutsches Buch zu Blazor:

www.IT-Visions.de/BlazorBuch