asp.net CORE - Blazor

Dipl.-Ing. Msc. Paul Panhofer Bsc.

15. November 2021

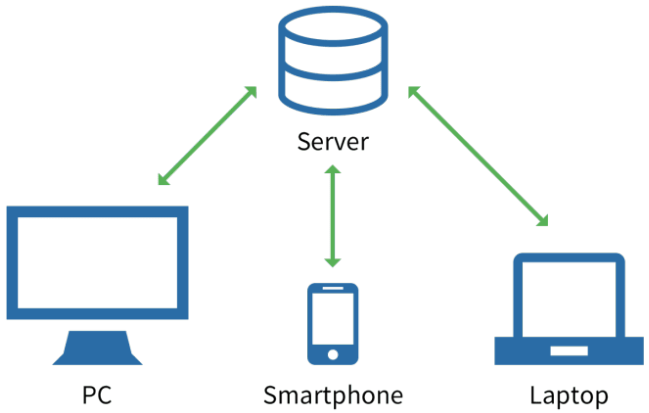
1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Blazor - Grundlagen

Client - Server Architektur

Webanwendungen unterteilen sich in eine Server- und Clientkomponente.



ASP.NET Core

Blazor - Grundlagen

Client - Server Architektur

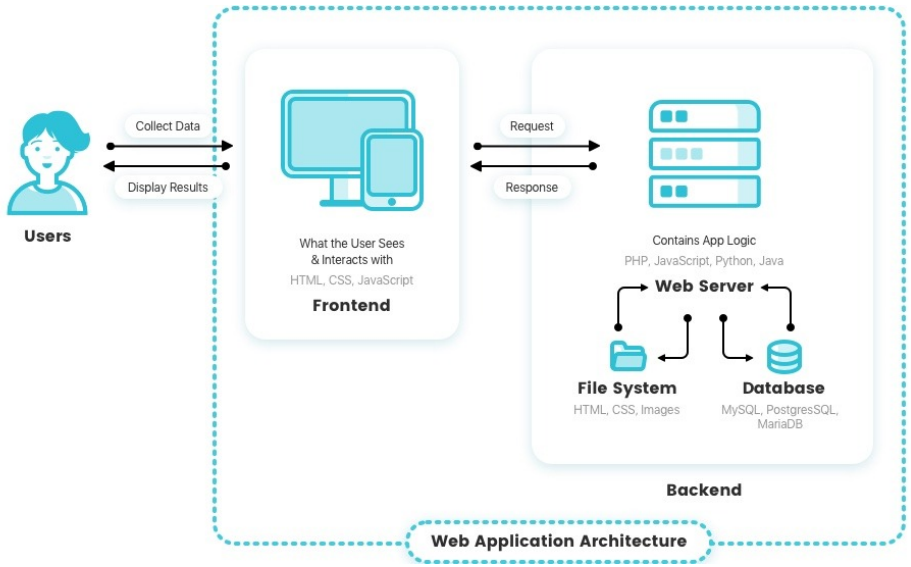
Die Geschäftslogik einer Anwendung wird serverseitig implementiert. Clients werden dabei in erster Linie zur **Darstellung**, der vom Server verwalteten Daten, eingesetzt.

ASP.NET Core

Blazor - Grundlagen

Frontend vs. Backend

Am Server liegende **Komponenten** werden synonym als Backend bezeichnet. Am Client liegende Komponenten bilden das Frontend.

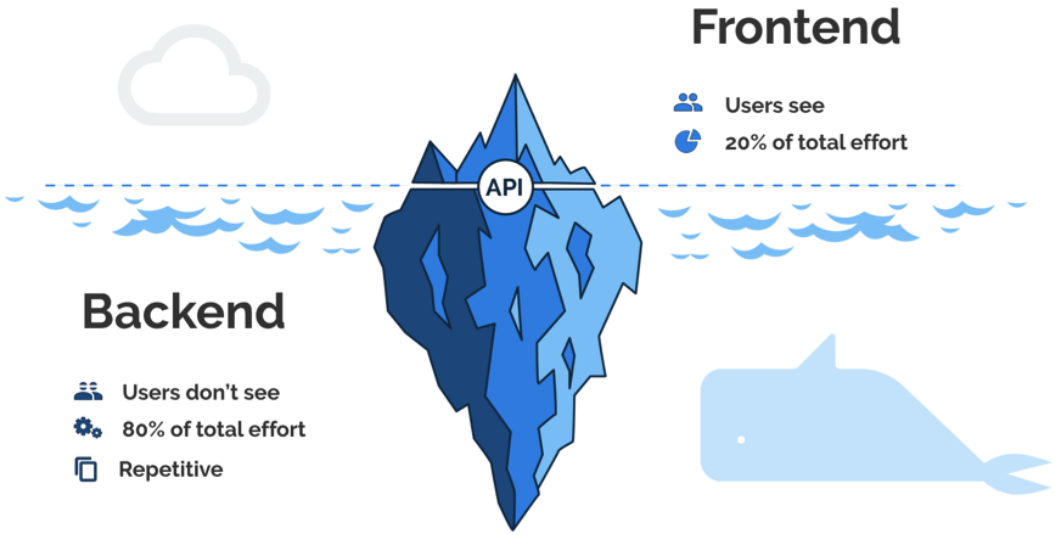


ASP.NET Core

Blazor - Grundlagen

Frontend vs. Backend

Blazor ist eine Frontendtechnologie für das asp.net CORE Framework.



ASP.NET Core

Blazor - Grundlagen

Frontend vs. Backend

ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Blazor - Grundlagen

Frontend Technologien

Grundlegende Frontend Technologien: JavaScript, HTML, CSS.

ASP.NET Core

Blazor - Grundlagen

Webassembly

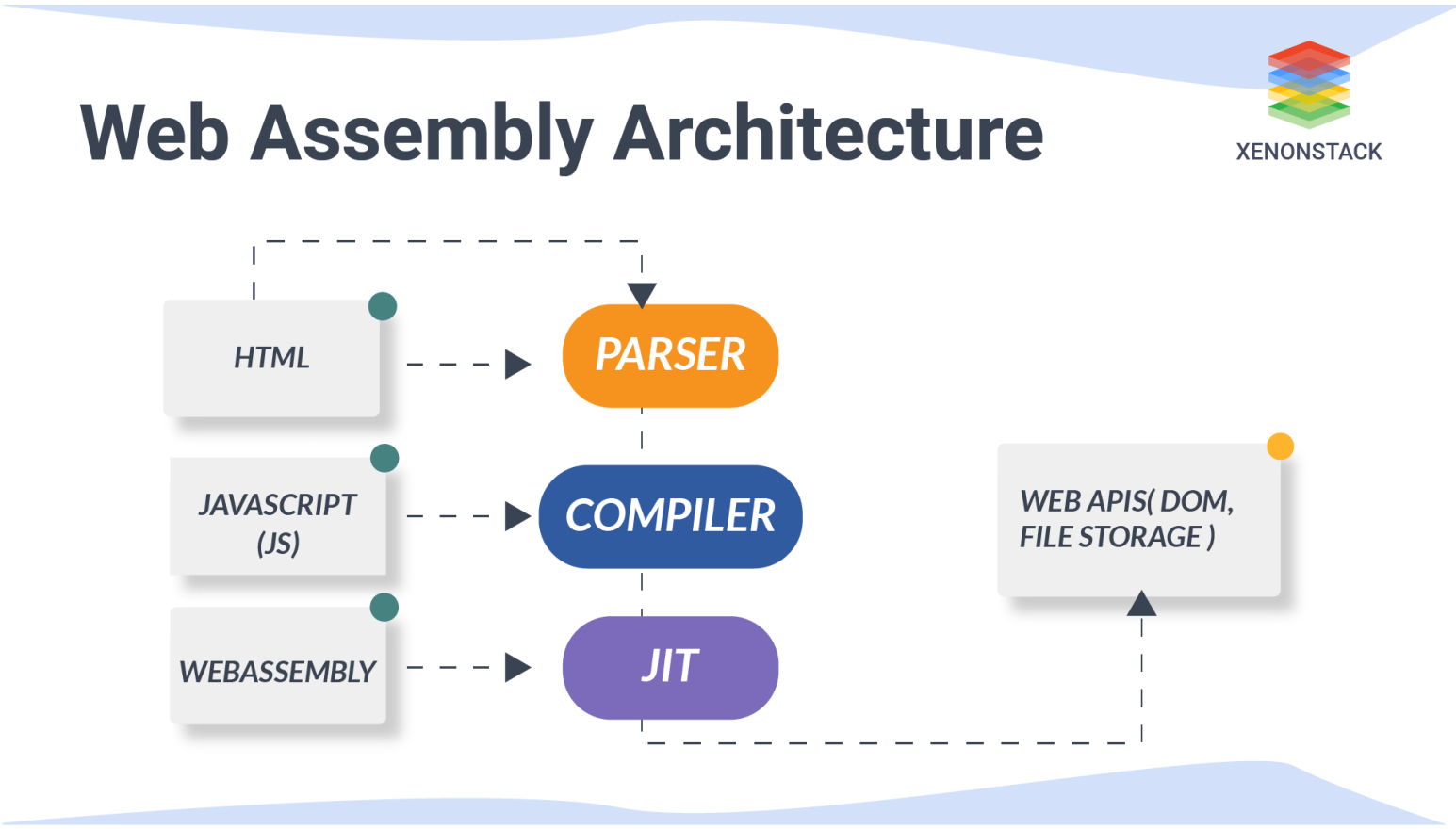
Als Webassembly wird ein spezieller **Bytecode** bezeichnet, der in einem Webbrowser ausgeführt werden kann.

Die Technologie selbst versteht sich dabei als **Alternative** zu JavaScript. Webassembly kann im Browser siginifikant schneller ausgeführt werden.

ASP.NET Core

Blazor - Grundlagen

Webassembly - **J**ust **i**n **T**ime Compiler



ASP.NET Core

Blazor - Grundlagen

Webassembly - **J**ust **i**n **T**ime Compiler

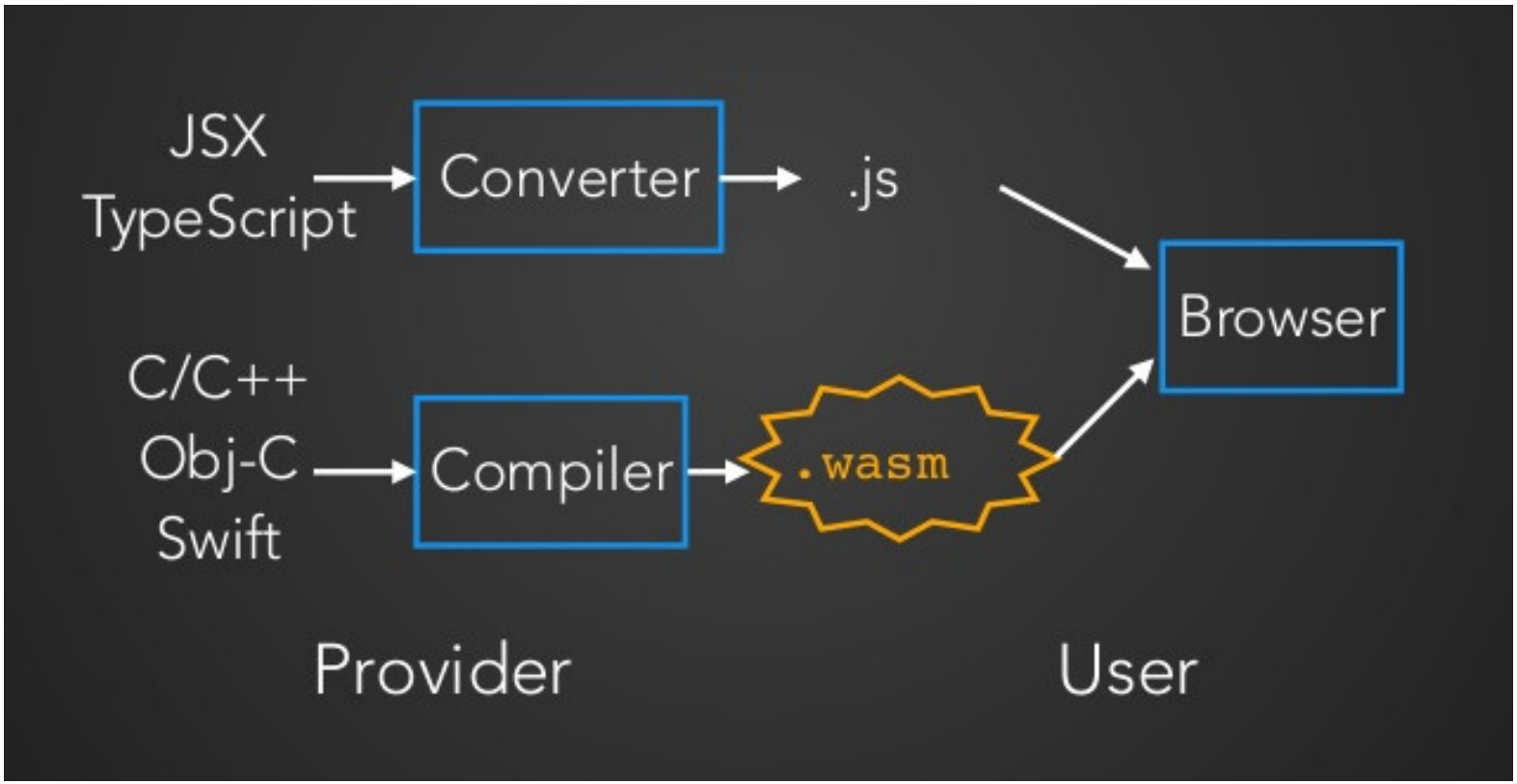
Mit Webassembly ist es möglich den Quellcode einer **höheren Programmiersprache** (Java, C, C++ bzw. die Sprachen der .net Familie) im Browser auszuführen.

Bytecode darf dabei nicht mit Maschinencode verwechselt werden, der direkt von einem Prozessor ausgeführt werden kann.

ASP.NET Core

Blazor - Grundlagen

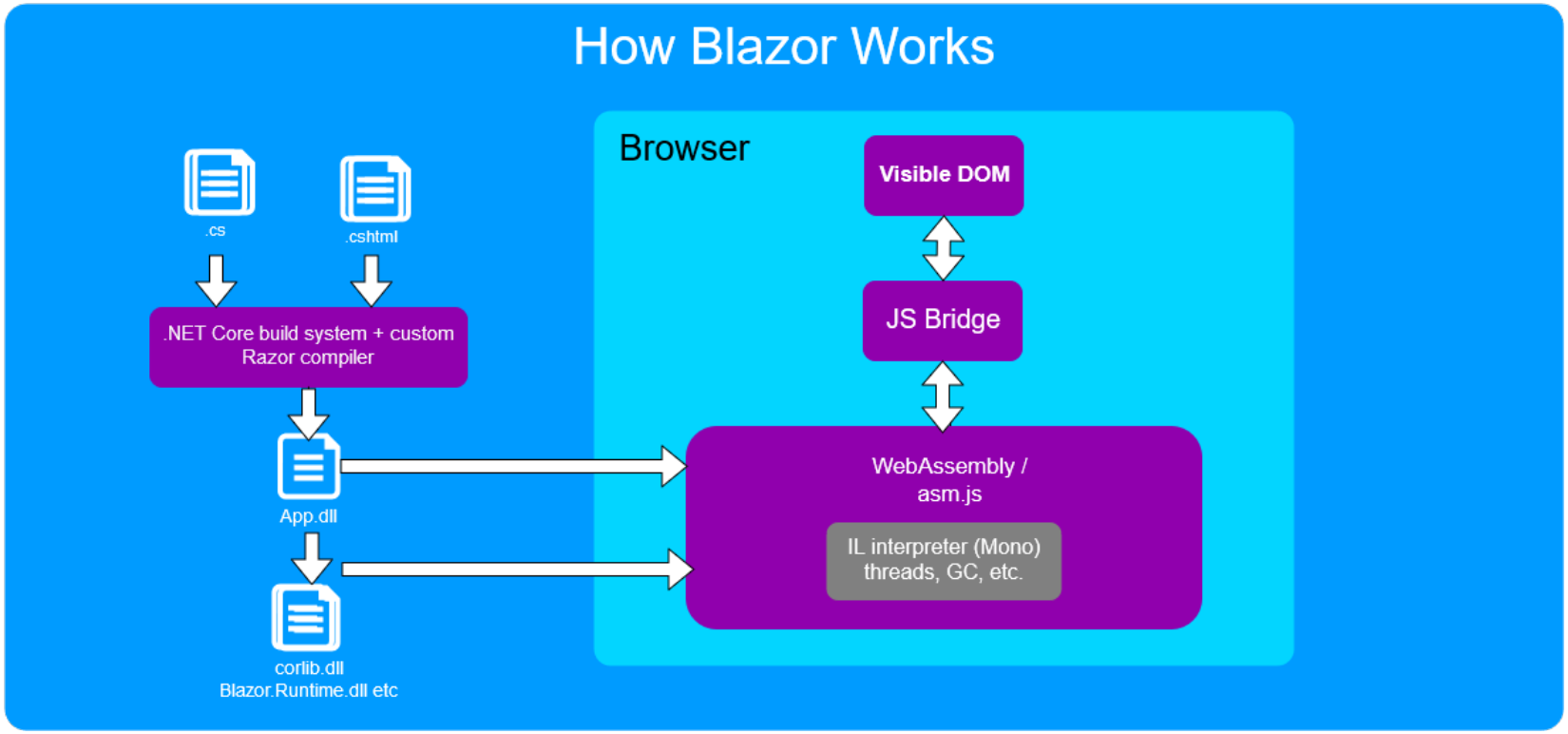
Typescript vs. Webassembly



ASP.NET Core

Blazor - Grundlagen

Typescript vs. Webassembly



ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Blazor

ASP.NET Core

Blazor

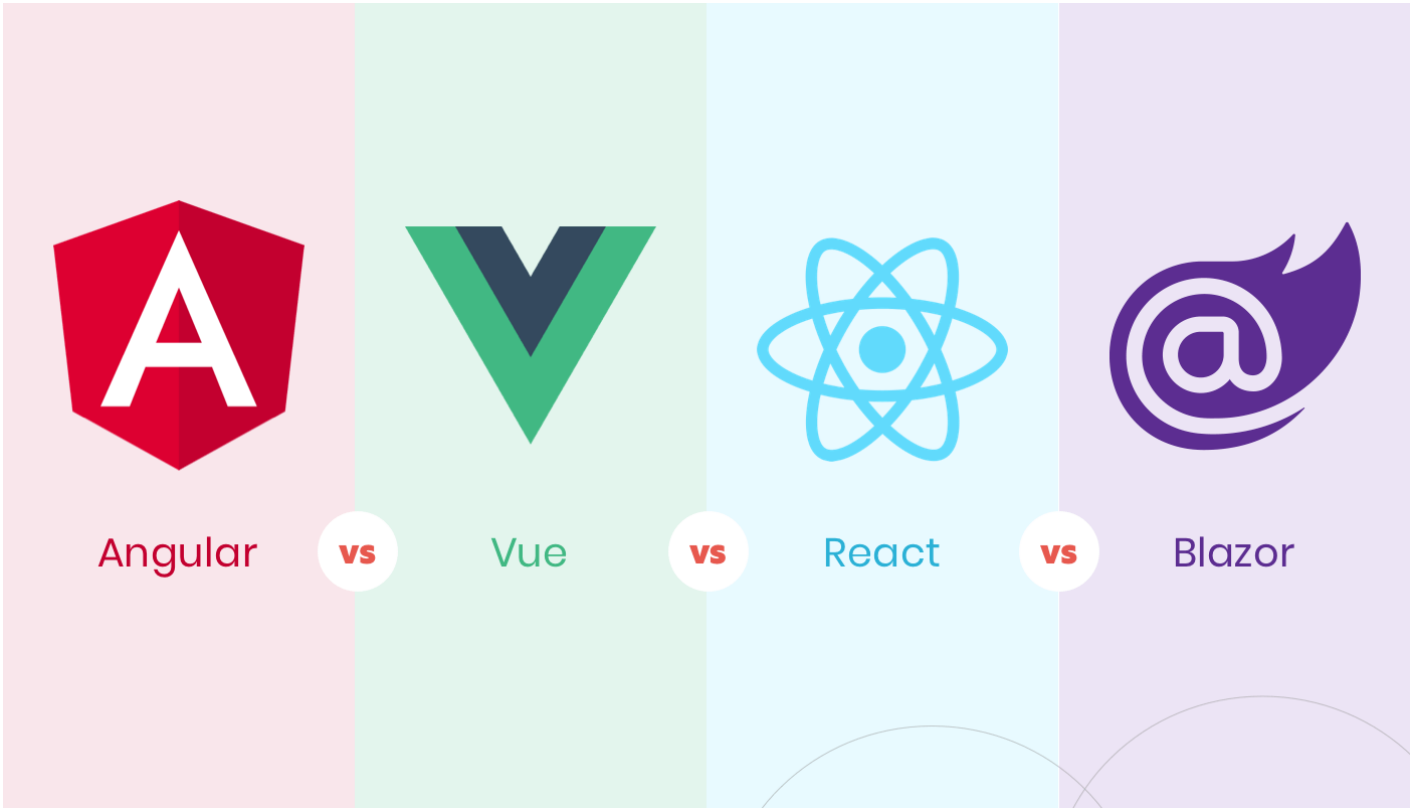
Blazor ist ein Framework zum Erstellen interaktiver **Oberflächen** für Webanwendungen.

Blazor ist damit vergleichbar mit Technologien wie angular oder react.

ASP.NET Core

Blazor

Webframeworks: Singlepage Frameworks



ASP.NET Core

Blazor

Motivation

- Frontend und Backend einer Anwendung können in einer einzelnen **Technologie** (.net CORE) entwickelt werden.
- **.net Sprachen** im Browser anstatt JavaScript.
- Erhöhte **Leistung** und Sicherheit vergliechen mit JavaScript

ASP.NET Core

Blazor

Hostingmodelle: Webassembly vs. asp.net CORE

Blazoranwendungen (Webassembly) können isoliert von einer Serverkomponente (asp.net Core) ausgeführt werden.

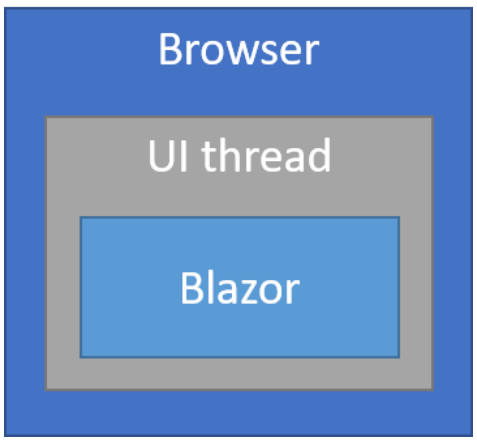
Beim Erstellen einer Anwendung muss von Anfang an eines der beiden Hostingmodelle gewählt werden.

ASP.NET Core

Blazor

Blazor Webassembly

Blazor Webassembly Anwendungen werden **clientseitig** im Browser gehostet. Die Anwendung wird dabei im UI Thread des Browsers ausgeführt.



ASP.NET Core

1 Blazor - grundlegende Konzepte
- Client - Server Architektur
- Webassembly
- Blazor

2 Razor Pages
- Razor Grundlagen
- Razor Ausdruck
- Single Page Application
- Component Lifecycle
- Navigation
- Formulare

ASP.NET Core

Razor Pages

Grundlagen

Blazor Anwendungen sind aufgebaut aus Razor Pages. Zur Programmierung von Razor Pages wird eine Kombination aus HTML und C verwendet.

ASP.NET Core

Razor Pages

html vs. Code

Blazor ermöglicht das direkte Zusammenspiel von HTML und Programmcode.

```
<html>
   <h1 class="text-info">Hallo @name!</h1>

    @code {
       string name = "Hugo"
    }
</html>
```


ASP.NET Core

Razor Pages

Element: Markup + CSS

HTML tags definieren die **Struktur** einer Razor Page.

```
<html>
    <h1 class="text-info">...</h1>
    <div>
        <span class="bg-light border p-2">...</span>
        ...
    </div>
</html>
```


ASP.NET Core

Razor Pages

Razor Pages: Elemente

Zur Implementierung der **Logik** von Razor Pages werden Razor Ausdrücke verwendet.

Es gibt 3 Formen von Razor Ausdrücken:

- Razor Ausdruck
- Razor Code Block
- Razor Steuerungsstruktur

ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Razor Pages

Razor Ausdrck

Razor Ausdrücke werden bei der Ausführung einer Razor Page zu einem einzelnen **Wert** evaluiert.

Für Razor Ausdrücke werden 2 Formen unterschieden:

- Implizite Razor Ausdrücke
- Explizite Razor Ausdrücke

ASP.NET Core

Razor Pages

Implizite Razor Ausdruck

Ein impliziter Razor Ausdruck beginnt mit einem @ gefolgt von einem **Ausdruck**.

Hinweis: Implizite Razor Ausdrücke ermöglichen den einfachen Zugriff auf **Variablenwerte**.

```
<html>
    ...
    <span>Zeitpunkt: @DateTime.Now </span>
    ...
</html>
```


ASP.NET Core

Razor Pages

Beispiel: Implizite Razor Ausdruck

```
<html>
    ...
    <span>Hallo @CustomToUpper(Name) </span>

    @code {
        public string Name { get; set; } = "Hugo";

        string CustomToUpper(string value) =>
            value.ToUpper();
    }
</html>
```


ASP.NET Core

Razor Pages

Explizite Razor Ausdruck

Ein Expliziter Razor Ausdruck beginnt mit einem @ gefolgt von einem **Ausdruck** eingebettet in ein Klammernpaar.

Hinweis: ERAs kommen immer dann zum Einsatz, wenn die Syntax eines Ausdrucks nicht vom HTML Parser verarbeitet werden könnte.

```
<html>
    ...
    <span>Ergebnis: 2 + 2 = @(2 + 2) </span>
    ...
</html>
```


ASP.NET Core

Razor Pages

Razor Code Block

Ein Razor Code Block beginnt mit einem @ gefolgt von **Programmdefinitionen** (*Klassen, Methoden, Variablen*) eingebettet in Mengenklammern.

Hinweis: Während Razor Ausdrücke stets zu einem Wert evaluiert werden, werden Code Blöcke **nicht gerendert**. Sie dienen zur Definition von Codeelementen.

ASP.NET Core

Razor Pages

Beispiel: Razor Code Block

```
<html>
    <!-- Razor Ausdruck -->
    <span>Hallo @CustomToUpper(Name) </span>

    <!-- Razor Code Block -->
    @code{
        public string Name { get; set; } = "Hugo";

        string CustomToUpper(string value) =>
            value.ToUpper();
    }
</html>
```


ASP.NET Core

Razor Pages

Razor Steuerungsausdruck

Razor Steuerungsstrukturen stellen eine **Erweiterung** von Razor Codeblöcken dar.

Steuerungsstrukturen dienen zur Steuerung des **Programmablaufs** in Razor Pages.

ASP.NET Core

Razor Pages

Steuerungsausdruck: if

```
<html>
    ...
    @if(names.Length > 0) {
        <span>We have friends: @names.Length<span>
    } else {
        <span>Let's buy a dog!<span>
    }

    @code {
        string[] names = {"Tobi", "Nikolei", "Franz"}
    }
</html>
```


ASP.NET Core

Razor Pages

Steuerungsausdruck: case

```
<html>
    ...
    @switch(index){
        case 200: <p>OK</p> break;
        case 404: <p>NOT FOUND<p> break;
        default :
           <p>code is wrong<p>
           break;
    }

    @code {
        int code = 404;
    }
</html>
```


ASP.NET Core

Razor Pages

Steuerungsausdruck: for

```
<html>
    ...
    @for(var i = 0; i < persons.Length; i++){
        var person = persons[i];

        <p>Name: @person.Name</p>
        <p>Age: @person.Age</p>
    }

    @code {
        Person[] persons = {
            new Person(){ name = "Huber", age = 21 },
            new Person(){ name = "Abdul", age = 23 }
        }
    }
</html>
```


ASP.NET Core

Razor Pages

Steuerungsausdruck: foreach

```
<html>
    ...
    @foreach(var person in persons){
        <p>Name: @person.Name</p>
        <p>Age:  @person.Age</p>
    }

    @code {
        Person[] persons = {
            new Person(){ name = "Huber", age = 21 },
            new Person(){ name = "Abdul", age = 23 }
        }
    }
</html>
```


ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Single Page Application

SPA Webanwendungen

Als SPA werden **Webanwendungen** bezeichnet, die aus einer einzigen HTML Seite bestehen.

Inhalte der Seite werden dabei bei Bedarf dynamisch nachgeladen.

ASP.NET Core

Single Page Application

Kommunikation

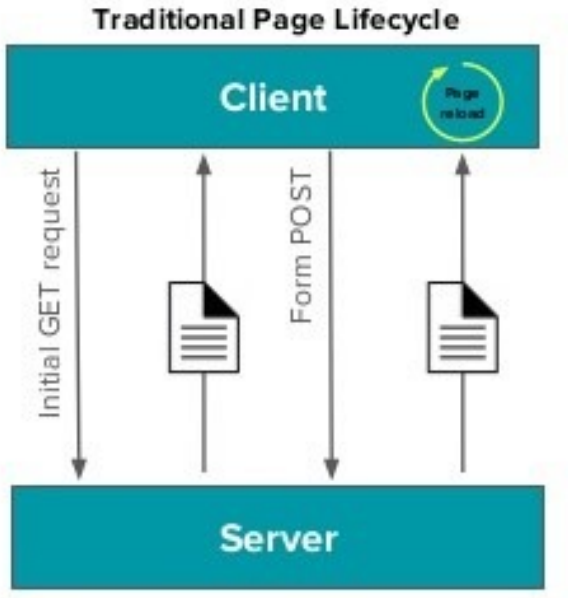
Intern verzichten SPAs auf die Navigation zwischen Webseiten.

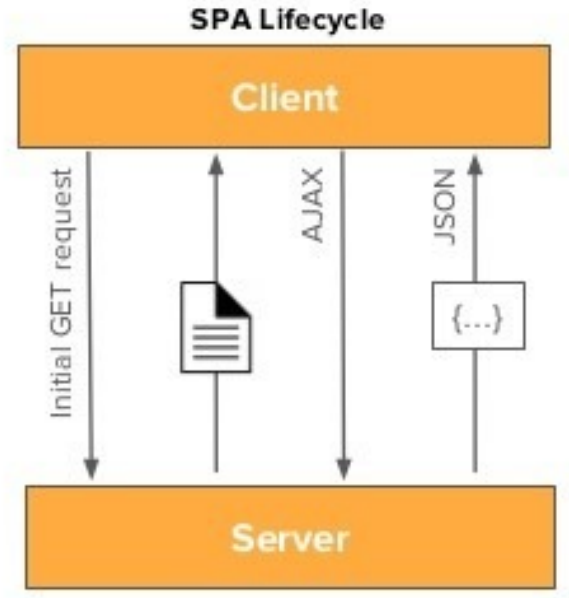
Der Präsentationsfluss wird nicht angehalten, da keine neuen Seiten, sondern nur **Bruchstücke** der ursprünglichen Seite neu geladen werden müssen.

ASP.NET Core

Single Page Application

WA vs. SPA

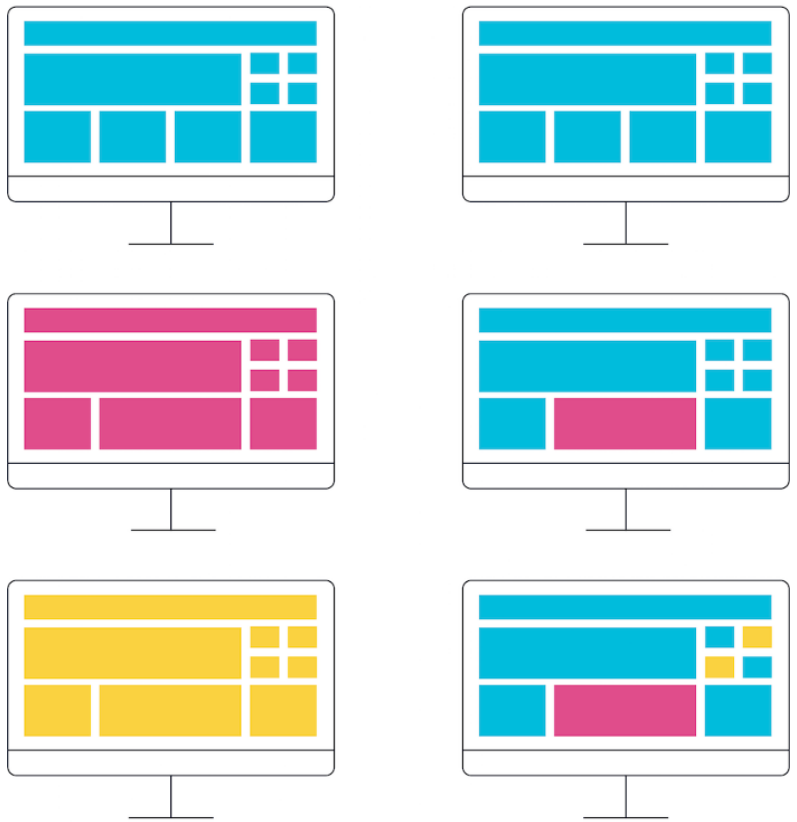




ASP.NET Core

Single Page Application

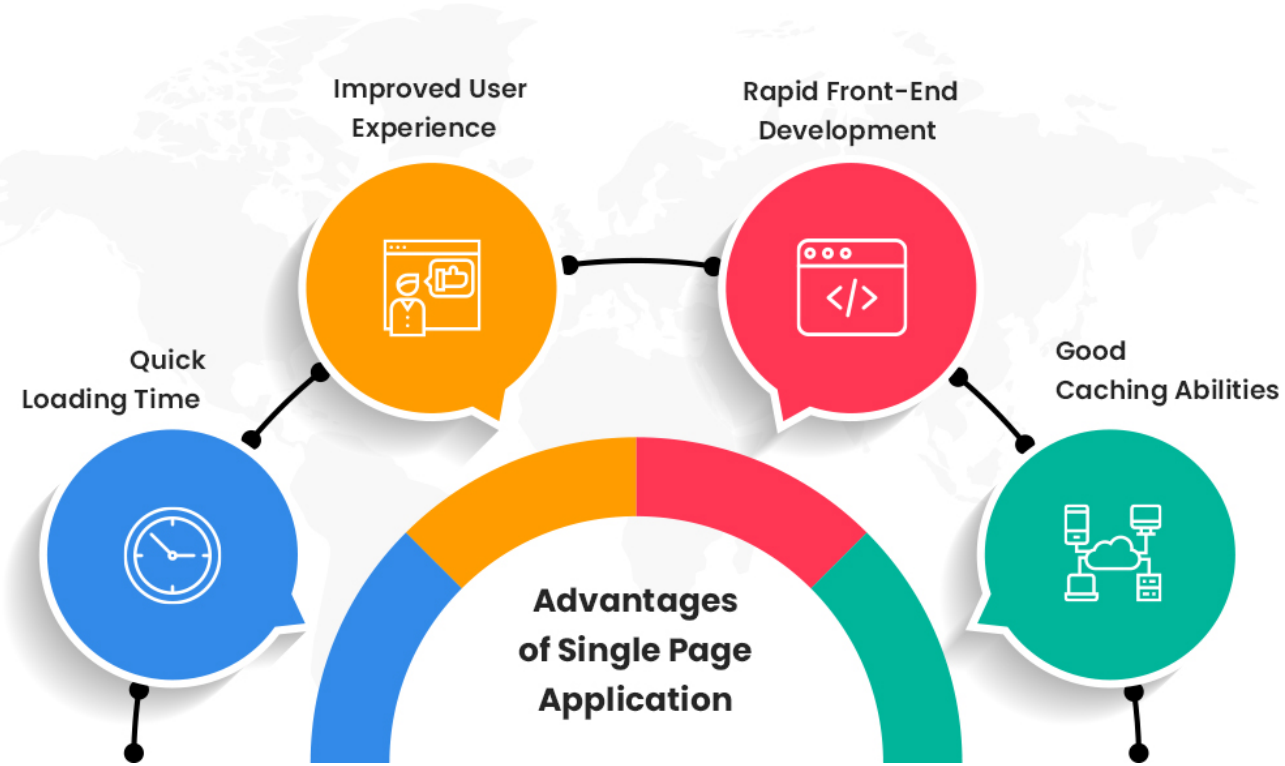
WA vs. SPA



ASP.NET Core

Single Page Application

Vorteile



ASP.NET Core

Single Page Application

Komponenten

Komponenten sind die **Bausteine** einer SPA Anwendung.

Zur Laufzeit einer Anwendung werden Komponenten fortlaufend **geladen** und wieder verworfen. Die ursprüngliche Seite bleibt dieselbe.

ASP.NET Core

Single Page Application

Razor Komponente

Blazor Anwendungen bestehen aus **Razor Komponenten**.

ASP.NET Core

Single Page Application

Razor Komponente

Razor Komponenten sind eine **Mischung** von HTML und Programmcode. Razor Komponenten werden intern zu c# Klassen kompiliert.

ASP.NET Core

Single Page Application

Beispiel: Razor Komponente

```
<!-- ------------------------------------------ -->
<!-- Komponente: Card
<!-- ------------------------------------------ -->
<div class="card">
    <div class="card-body">
        <h3 class="card-title">@Movie.Title</h3>
        <p class="small">@Movie.Description</p>
     </div>
</div>

@code {
    [Parameter]
    Movie Movie { get; set; }
}
```


ASP.NET Core

Single Page Application

Beispiel: Razor Komponente

```
<!-- ------------------------------------------ -->
<!--  Einbinden von Card
<!-- ------------------------------------------ -->
   ...
   <div class="col">
      <div class="caed-deck">
          <Card Movie="@MovieCard"/>
      </div>
   </div>
   ...

@code{
    Movie MovieCard { get; set; }
}
```


ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Component Lifecycle

Lifecycle Methoden

Beim **Laden** einer neuen Komponente, werden vor dem Rendern der Komponente in der Webanwendung, durch das Framework eine Zahl von Methoden aufgerufen.

Diese Methoden werden auch als Lifecycle Methoden bezeichnet.

ASP.NET Core

Component Lifecycle

Lifecycle Methoden - Frameworkhook

Die Lifecycle Methoden dienen dem Framework dabei als Frameworkhooks. Programmierer haben damit die Möglichkeit mit der Komponente **programmtechnisch** zu interagieren.

Hinweis: Razor Komponenten erben von `ComponentBase`. `ComponentBase` gibt die Lifecycle Methoden an alle Komponenten weiter.

ASP.NET Core

Component Lifecycle

Lifecycle Methoden

```
<div> ... </div>
@code {
    protected override Task SetParametersAsync(
            ParameterView parameters
    ) {...}
    protected override void OnInitialized() {...}
    protected override async Task OnInitialized() {...}
    protected override void OnParametersSet() {...}
    protected override async Task OnParametersSet() {...}
    protected override void OnAfterRender(
         bool firstRender
    ) {...}
}
```


ASP.NET Core

Component Lifecycle

Lifecycle Methoden: Reihenfolge

```
<div> ... </div>
@code {
    (1) protected o. Task SetParametersAsync(
            ParameterView parameters
        ) {...}
    (2) protected o. async Task OnInitialized() {...}
    (3) protected o. async Task OnParametersSet() {...}
    (4) protected o. async Task OnAfterRender(
            bool firstRender
        ) {...}
}
```


ASP.NET Core

Component Lifecycle

Lifecycle Methoden: SetParametersAsync

```
<div> ... </div>
@code {
    /*
     * Die Methode wird aufgerufen um die [Parameter]
     * der Komponente zu laden
     */
    protected override Task SetParametersAsync(
        ParameterView parameters
    ) {...}
}
```


ASP.NET Core

Component Lifecycle

Lifecycle Methoden: `OnInitializedAsync`

```
<div> ... </div>
@code {
    /*
     * Die Methode wird aufgerufen, um die internen
     * Variablen der Komponente zu initialisieren
     */
    protected override Task OnInitialisedAsync() {...}
}
```


ASP.NET Core

Component Lifecycle

Lifecycle Methoden: `OnParametersSetAsync`

```
<div> ... </div>
@code {
    /*
     * Die Methode wird aufgerufen um auf die
     *  nderung  von Parametern zu reagieren
     */
    protected override Task OnParametersSetAsync() {...}
}
```


ASP.NET Core

Component Lifecycle

Lifecycle Methoden: `OnAfterRenderAsync`

```
<div> ... </div>
@code {
    /*
     * Die Methode wird aufgerufen, nachdem die Komponente
     * auf der Seite gerendert wurde
     */
    protected override Task OnAfterRenderAsync() {...}
}
```


ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Navigation

Page vs. Component

Blazor unterscheidet 2 Arten von Komponenten:

- Razor Page
- Razor Component

ASP.NET Core

Navigation

Page Direktive

Pages sind Komponenten die explizit über eine **URL** geladen werden können. Programmtechnisch wird dazu die `@page` Direktive verwendet.

Hinweis: Die **@page** Direktive wird intern als **Route** Attribut aufgelöst.

```
@page "/hallo"
<div>...</div>

@code {
    ...
}
```


ASP.NET Core

Navigation

Page Direktive - Path Variables

Blazor ermöglicht es Komponenten über URL **Pfadvariablen** zu initialisieren. Standardmäßig wird der Typ einer Pfadvariable als String angenommen.

```
@page "/people/{Message}"
<div>
    ...
    <h2>@Message</h2>
</div>

@code {
    [Parameter]
    public string Message { get; set; }
}
```


ASP.NET Core

Navigation

Page Direktive - Path Variables

Für Pfadvariablen kann explizit ein **Datentyp** definiert werden.

```
@page "/people/{Id:int}"
<div>...</div>

@code {
    [Parameter]
    public int Id { get; set; }
}
```


ASP.NET Core

Navigation

Page Direktive - Path Variables

Für Razor Komponenten können mehere **Pfade** definiert werden.

```
@page "/people/{Id:int}"
@page "/people/{Message}"
@page "/people"
<div>...</div>

@code {
    [Parameter]
    public int? Id { get; set; }
    [Parameter]
    public string? Message { get; set; }
}
```


ASP.NET Core

Navigation

Anchor

Zur Navigation zwischen Komponenten kann das **Anchor Element** verwendet werden.

```
@page "/people"
<div>
    ...
    <a href="/authors/1" class="...">Autoren</a>
</div>

@code {
    ...
}
```


ASP.NET Core

Navigation

NavigationManager

Programmtechnisch kann ein `NavigationManager` zur **Navigation** zwischen Komponenten verwendet werden.

```
@page "/people"
@inject NavigationManager _navManager
<div>...</div>

@code {
    public void Authorize(){
        ...
        _navManager.NavigateTo("/authors/2", true);
    }
}
```


ASP.NET Core

Navigation

NavigationManager

```
@page "/people"
@inject NavigationManager _navManager
<div>
    <button @onclick="@(
          () => _navigationManager.NavigateTo(
              "/authors/2", true
          )
    )">
        Load Author
    </button>
</div>

@code {
    ...
}
```


ASP.NET Core

1. Blazor - grundlegende Konzepte
   - Client - Server Architektur
   - Webassembly
   - Blazor
2. Razor Pages
   - Razor Grundlagen
   - Razor Ausdruck
   - Single Page Application
   - Component Lifecycle
   - Navigation
   - Formulare

ASP.NET Core

Formulare

Zur **Verarbeitung** von Daten werden in Blazor Formulare verwendet.

Mit Razorausdrücken können Objekte direkt an die Elemente eines Formulars gebunden werden. Dazu wird das `bind-value` Attribut verwendet.

ASP.NET Core

Formulare

Klasse: Movie

```
public class Movie {
    [StringLength(100)]
    [Required]
    public string Title { get; set;}

    [StringLength(400)]
    [Required]
    public string Description { get; set; }
    ...
    public int Duration { get; set; }
    ...
    public EGenre Genre { get; set; }
}
```


ASP.NET Core

Formulare

EditForm - Initialisierung

```
<EditForm Model="Movie" OnValidSubmit="Create"
    class="form">
    <DataAnnotationsValidator/>
    ...
</EditForm>

@code{
    public Movie Movie { get; set; } = new ();

    public void Create() {
        ...
    }
}
```


ASP.NET Core

Formulare

EditForm - Databinding

```
<EditForm ... class="form">
    <DataAnnotationsValidator/>
    <div class="form-row">
        <div class="form-group col-md-5">
            <label for="movieTitle" class="small
                text-info">title:</label>
            <InputText class="form-control
                form-control-sm" id="movieTitle"
                placeholder="input title"
                @bind-Value="@Movie.Title"/>
            <ValidationMessage For="()=> Movie.Title"/>
        </div>
    </div>
    ...
</EditForm>
```


ASP.NET Core

Formulare

EditForm - Send Request

```
<EditForm ... class="form">
    ...
    <div class="form-row mt-3">
        <NavLink href="movies" class="btn btn-danger
            btn-sm">Cancel</NavLink>
        <button class="btn btn-info ml-2
            btn-sm">@ButtonLabel</button>
    </div>
</EditForm>
```


ASP.NET Core