

ASP.NET Core 8

Ali Ilci

B. A.

- Kursablauf & Organisatorisches
- Räumlichkeiten
- Vorstellungsrunde
 - Hintergrund, Motivation und Vorkenntnisse
- Bei Unklarheiten: Möglichst sofort fragen!

Über mich – Ali Ilci

- Freiberuflicher IT - Trainer (MCT)
- Fachinformatiker
- Erziehungswissenschaftler
- Themenschwerpunkte:
 - ASP.NET Core / MVC / WebApi / Blazor
 - C#
 - Html5 & Css3
 - JavaScript / TypeScript
- www.ilci.de



.NET Core

Was ist .NET Core

- Plattformübergreifendes Framework mit dem Ziel so universell wie möglich eingesetzt zu werden
- Open Source
- Bibliotheken werden durch NuGet verteilt
- Typische Einsatzszenarien sind zurzeit ASP.NET Webanwendungen, Konsolen-Apps sowie UWP-Apps
- Ist keine Untermenge des .NET Framework
- Besitzt ein Command Line Interface (CLI)

Full .NET Framework vs. .NET Core

| Full .NET Framework | .NET Core |
|--------------------------------------|-------------------------------------|
| Vollständiges „etablierte“ Framework | Modulare Version des .NET Framework |
| Nur Windows | Cross-Plattform |
| | Keine Untermenge vom .NET Framework |
| | Windows, Linux, Mac |
| | Implementation des .NET Standard |

Entscheidung zwischen Full .NET Framework vs. .NET Core

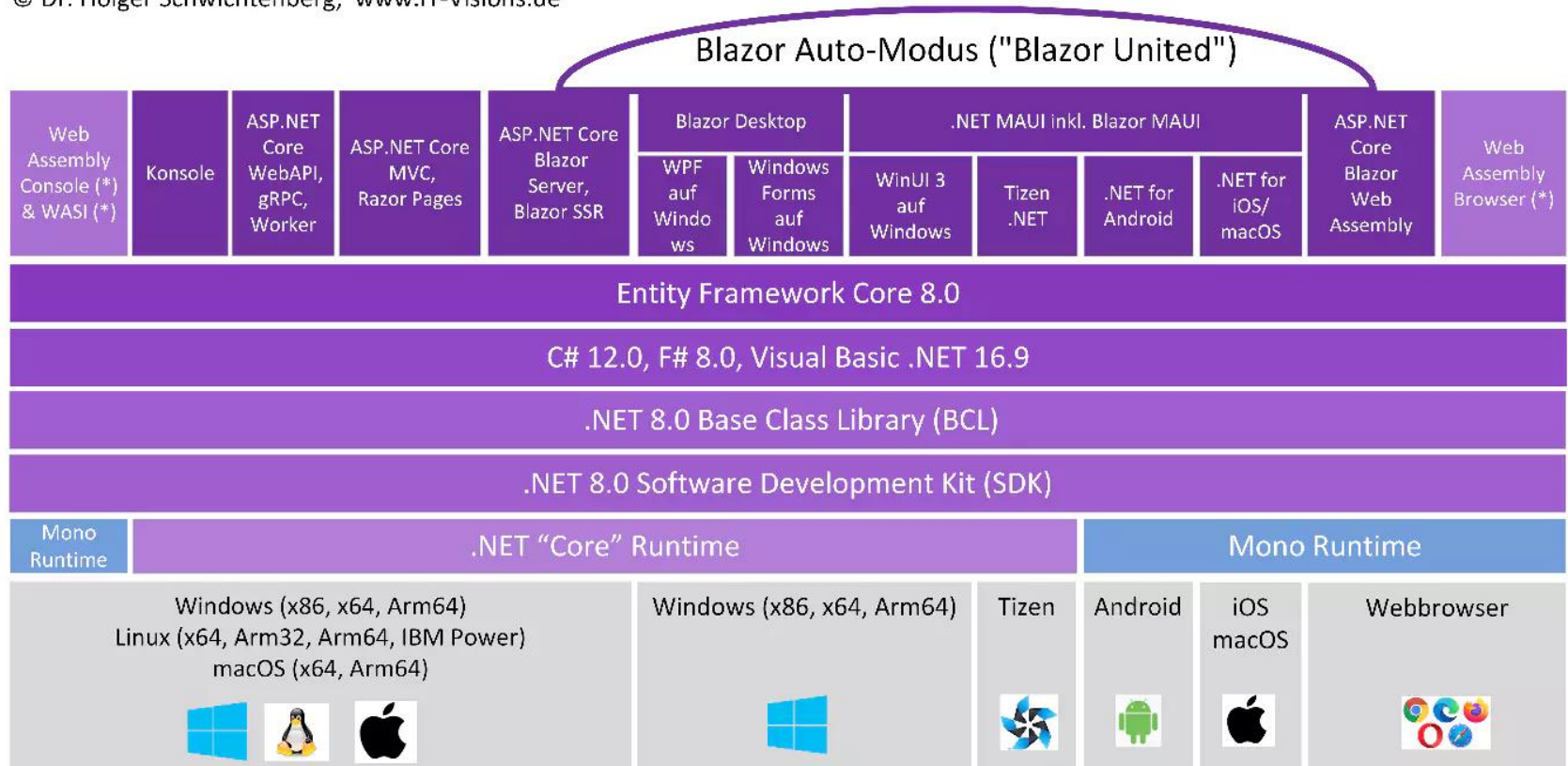
| Full .NET Framework | .NET Core |
|---|---|
| Aktuelle Projekt(e) nutzen .NET Framework | Cross-Plattform |
| Third-Party .NET Libraries | Docker Container |
| .NET Technologien, die nicht für .NET Core vorhanden sind | Side-by-Side .NET Versions in Applikationen |

<https://docs.microsoft.com/de-de/dotnet/articles/standard/choosing-core-framework-server>

.NET Familie 2023/2024

Aufbau von .NET 8.0

© Dr. Holger Schwichtenberg, www.IT-Visions.de

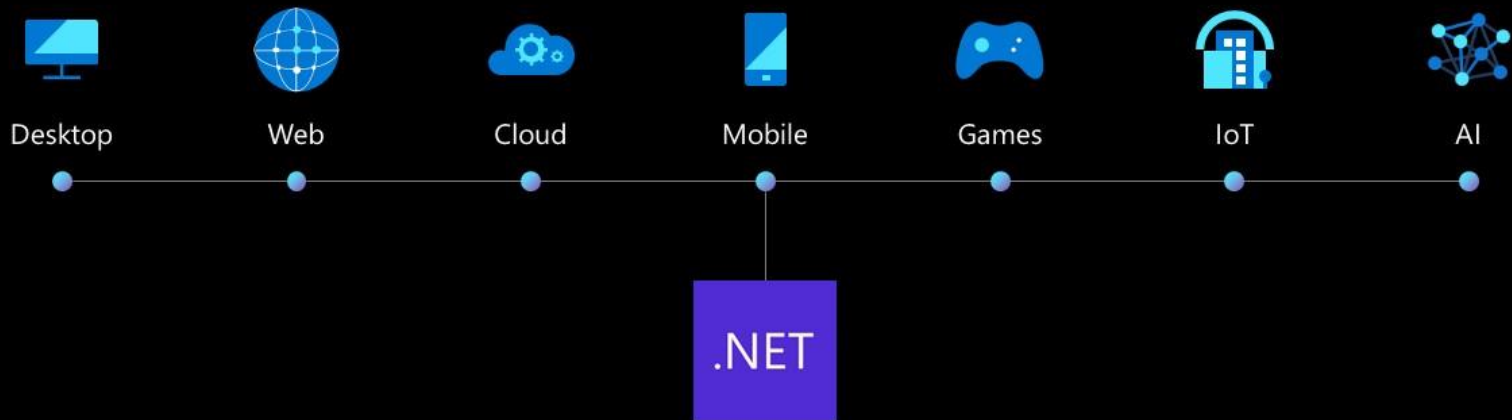


(*) Experimentelle Implementierungen ohne Support seit .NET 7.0 und auch noch in .NET 8.0

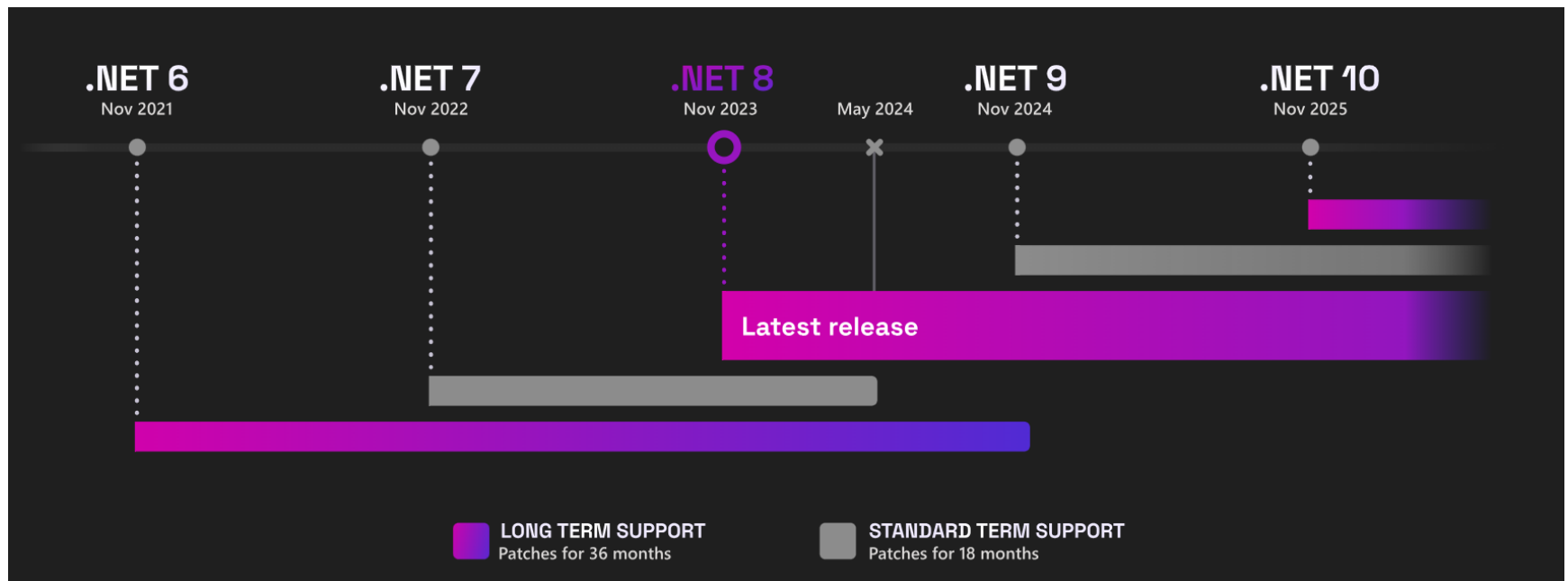
.NET 8 Platform

.NET

Your platform for building anything



.NET Schedule



<https://dotnet.microsoft.com/en-us/platform/support/policy>

- Command Line Application, die zum Entwickeln von Anwendungen auf allen Plattformen verwendet wird
- Führt die App aus und hosted die CLR
- OpenSource / SDK
- Erweiterbar (z.B. Entity Framework Befehle)
- Dokumentation
 - <https://docs.microsoft.com/de-de/dotnet/core/tools/>

.NET CLI Befehle „dotnet“ (Auszug)

| Befehl | Beschreibung |
|----------------|--|
| new | Erstellt ein neues Projekt, eine Konfigurationsdatei oder eine Lösung auf Grundlage der angegebenen Vorlage. |
| restore | Stellt die Abhängigkeiten und Tools eines Projekts wieder her |
| build | Erstellt ein Projekt und alle seine Abhängigkeiten |
| publish | Packt die Anwendung und ihre Abhängigkeiten in einen Ordner für die Bereitstellung auf einem Hostsystem |
| run | Führt Quellcode ohne explizite Kompilierungs- oder Startbefehle aus. |
| test | .NET-Testtreiber, der verwendet wird, um Komponententests auszuführen. |
| pack | Packt den Code in ein NuGet-Paket |
| clean | Löscht die Ausgabe eines Projekts. |
| sln | Ändert eine .NET Core-Projektmappendatei. |
| store | Speichert die angegebenen Assemblys im Laufzeitpaketspeicher |

Themen

Repository Tools
ViewComponents AutoMapper Bootstrap Model
Web Technologien Filter JQuery
TagHelper Komponenten Views Authentifizierung
AJAX Routing Mobile Views Middleware Validation
Jquery UI Partial Views
Fehlerbehandlung Actions Testing
Razor HTML Helpers
Action Filter Entity Framework Core Dependency Injection
Asynchronität Security Controller Data Annotations
Best Practices
Responsive Web Design C# 10/11/12

C# 10 / 11 / 12

Exkurs

Implicit & Global Using Statements

- Vermeidet wiederholte Auflistung von Usings
- Aktivierbar in Projektdatei (Default-Einstellung)
`<ImplicitUsings>enable</ImplicitUsings>`
- Globale Usings um Namespace projektweit bekannt zu machen

`//Vorher`

```
using System;  
using System.Collections.Generic;  
using Microsoft.AspNetCore.Http
```

`//Neu mit Implicit Using Statements`

`-`

`//Neu mit Global Using Statements in separater CS-Datei`
`global using Microsoft.AspNetCore.Http`

File Scoped Namespaces

- Einrückung der Namespaces
- Konfigurierbar für gesamte Solution mit .editorconfig

```
./.editorconfig  
csharp_style_namespace_declarations = file_scoped  
dotnet_diagnostic.IDE0161.severity = suggestion
```

```
namespace MeinNamespace.Services  
{  
    public class MeineKlasse  
    {  
    }  
}
```

```
//File Scoped Namespaces  
namespace MeinNamespace.Services;  
  
public class MeineKlasse  
{}
```


MaxBy/MinBy & DateOnly and TimeOnly

- **DateOnly** bzw. **TimeOnly** repräsentieren entweder das Datum **oder** die Uhrzeit im Gegensatz zu DateTime
- **MinBy/MaxBy** geben ganzes Objekt zurück

```
//DateOnly & TimeOnly
```

```
DateOnly date = DateOnly.MinValue; //01.01.0001 ohne Zeit
```

```
TimeOnly time = TimeOnly.MinValue; // 12:00
```

```
//MaxBy/MinBy
```

```
List<Person> people = new List<Person>
```

```
{
```

```
    new Person { Name = "John Doe", Alter = 25},
```

```
    new Person { Name = "Jane Doe", Alter = 23}
```

```
}
```

```
var person = people.MaxBy(c => c.Alter); // John Doe
```

Raw String Literal / Required Member

- Unformatierte Zeichenfolgen-Literale können beliebigen Text enthalten, ohne dass Escapezeichen erforderlich sind
- Required – Modifizierer gibt an, dass das Feld / Eigenschaft von allen Konstruktoren oder via Objektinitializer initialisiert werden muss

```
//String Literal vor C#11
```

```
var message = "Der folgende Text ist sehr \"wichtig\" ".
```

```
//ab C#11
```

```
var message = """Der folgende Text ist sehr "wichtig" """.
```

```
//Required Member
```

```
public required string FirstName {get; init;}
```

Primärkonstruktoren C# 12

- Verkürzte Schreibweise von Konstruktoren. Achtung parameterloser Konstruktor ist nicht mehr vorhanden und muss separat deklariert werden. Dadurch Aufruf des Primärkonstruktors

```
public class Kunde(Guid kundeId, string name, float preis)
{
    public Guid KundeId { get; set; } = kundeId;
    public string Name { get; set; } = name;
    public Kunde() : this(Guid.Empty, "") { }

    public override string ToString()
    {
        return $" : {Name} {preis}";
    }
}
```

Vereinfachte Initialisierung von Mengen, Spread, Opt. Parameter in Lambdas C# 12

- Syntax mit eckigen Klammern wie in Javascript
- Spread-Operator möglich (Array aus anderen Arrays)
- Optionale Parameter in Lambdas

// Vorher

```
string[] fruechte = new string[] {"Banane", "Orange"};
```

//Nachher

```
string[] fruechte = ["Banane", "Orange"];
```

//Spread-Operator

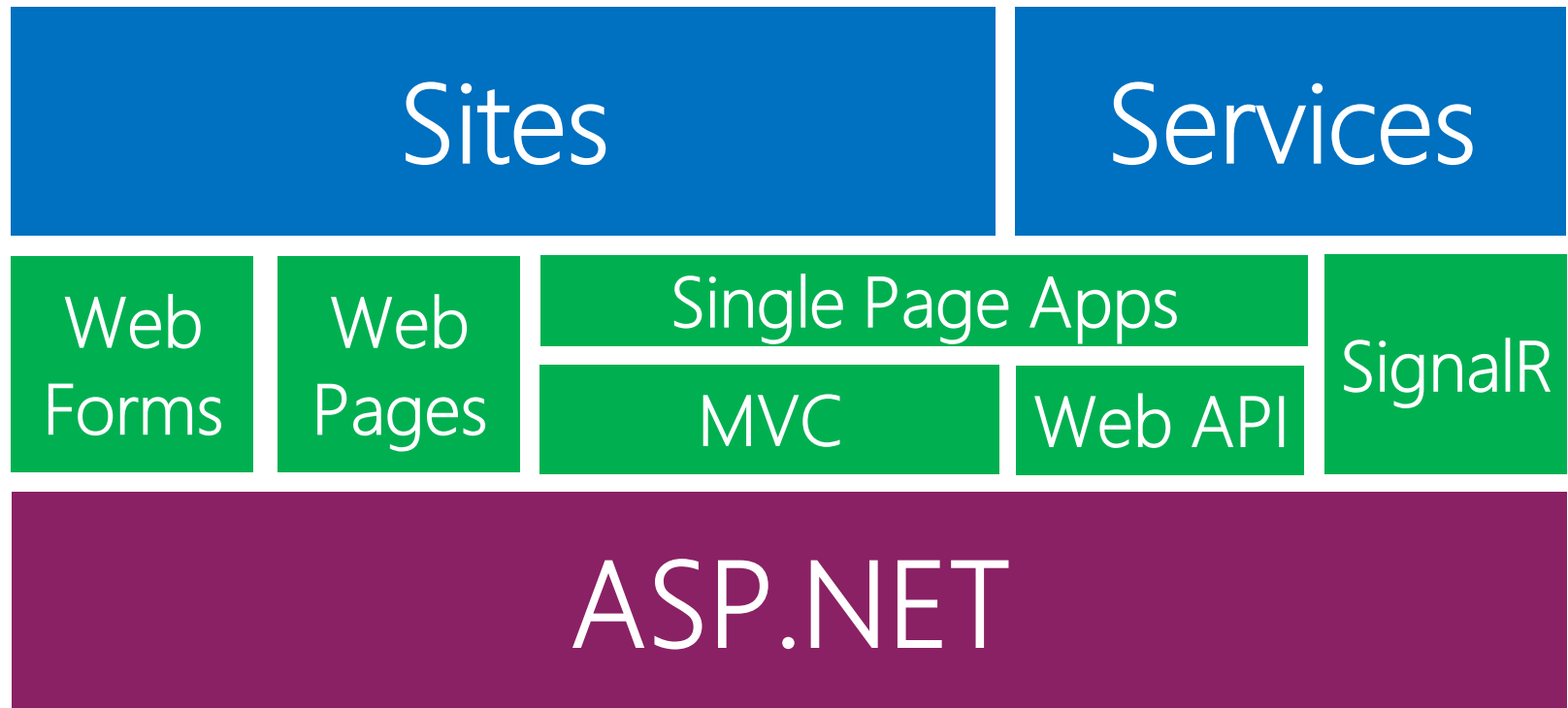
```
string[] alleFruechte = [.. fruechte, "Apfel"];
```

//Lambda mit opt. Parameter

```
var bru = (decimal x, decimal mw = 1.19) => (x * mw);
```

ASP.NET Core 8

Überblick ASP.NET 4.5



ASP.NET Core 8

(MVC + WebAPI + MinimalApi + Razor Pages + SignalR
+ gRPC + Blazor (SSR, Server, Blazor WebAssembly))

C# 12

.NET 8 Base Class library

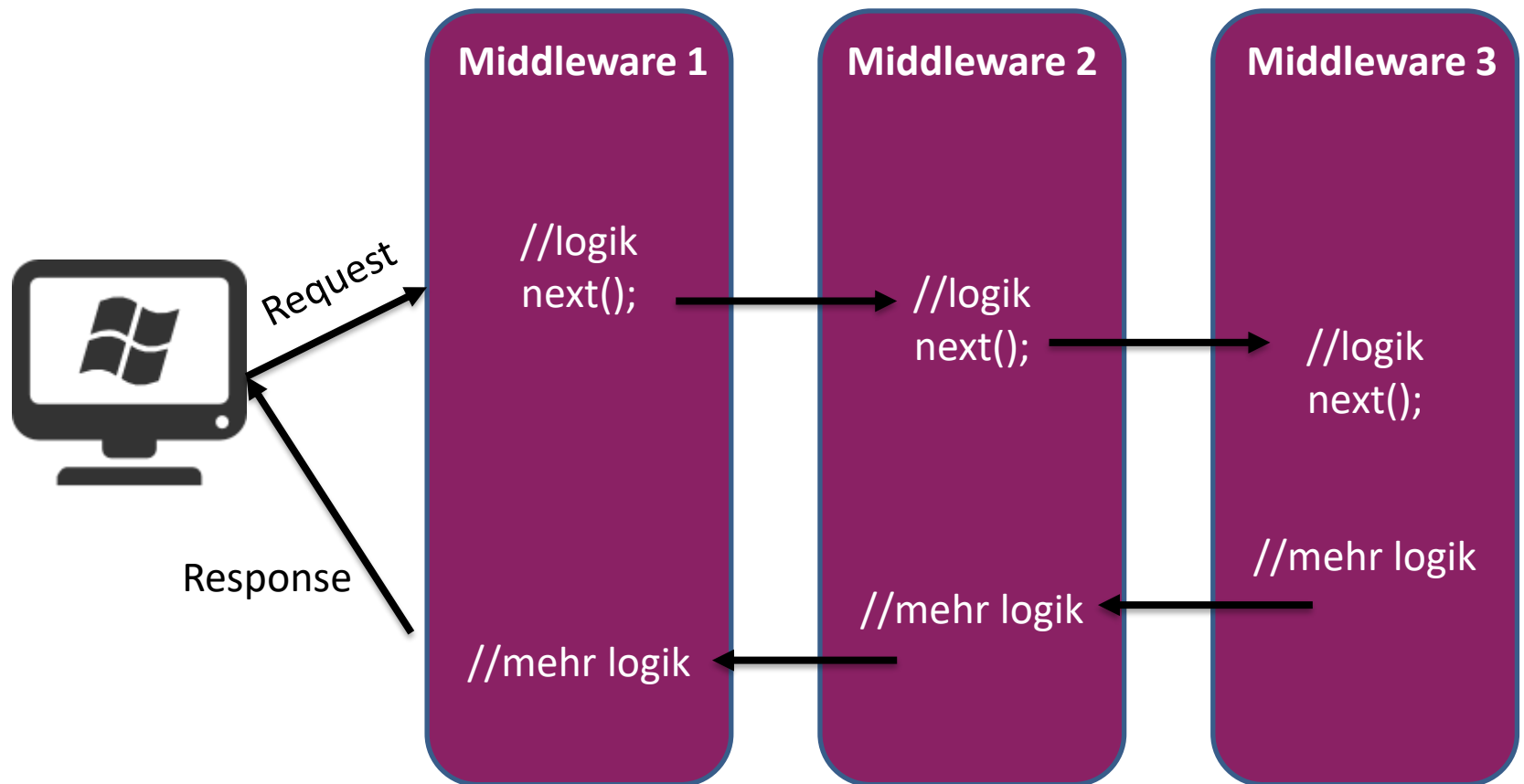
.NET Core

Windows & Linux & MacOS

Middleware

ASP.NET Core Request Pipeline & Middleware

- Querschnittsfunktion ähnlich HttpModulen & MessageHandler
- Erste Anlaufstelle bei der Bearbeitung der Anfragen.



- Software, die zu einer Anwendungspipeline zusammengesetzt wird (Delegate)
- Anwendung durch Extension-Methoden des **WebApplication** Objekts
- Die **WebApplication** stellt den Mechanismus bereit die Request-Pipeline zu konfigurieren
- Built-In-Middleware
 - Mvc, StaticFiles, CORS, Routing, DeveloperExceptionPage etc.
- Custom Middleware möglich
- Reihenfolge des Aufrufs relevant

- **Program-Klasse**
 - Verantwortlich für die Konfigurierung/Ausführen der Anwendung
 - Einstiegspunkt der Webapplikation
- **WebApplicationBuilder** verantwortlich für Configuration und Service Registration
- **IServiceCollection** wird genutzt um Services dem Container hinzuzufügen und zu konfigurieren
- Spezifizierung wie eine ASP.NET Applikation auf individuelle HTTP-Anfragen zu antworten
- Middlewares um HTTP-Anfrage-Pipeline zu konfigurieren



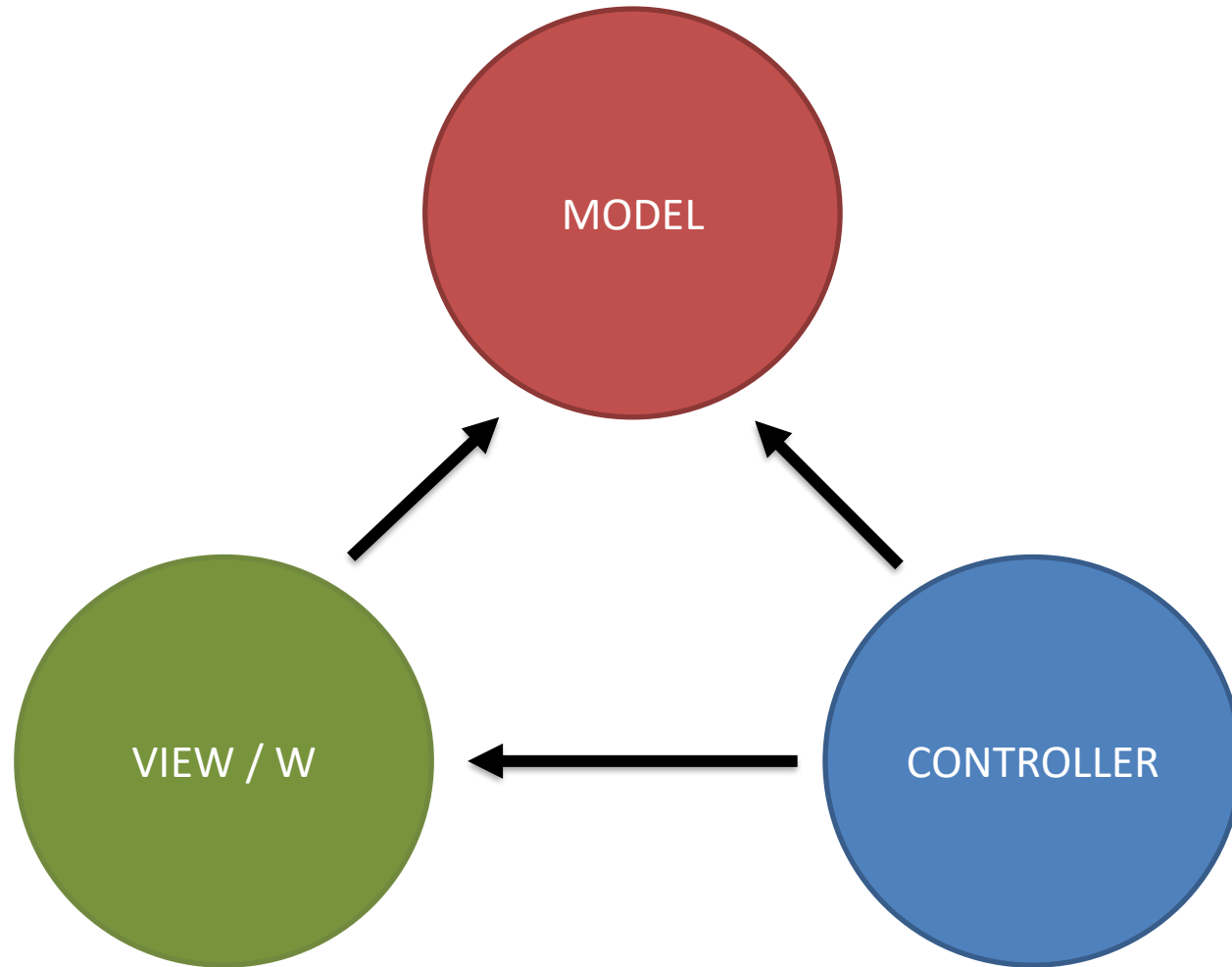
Projekt „Teilnehmer“ Middleware

ASP.NET Core MVC

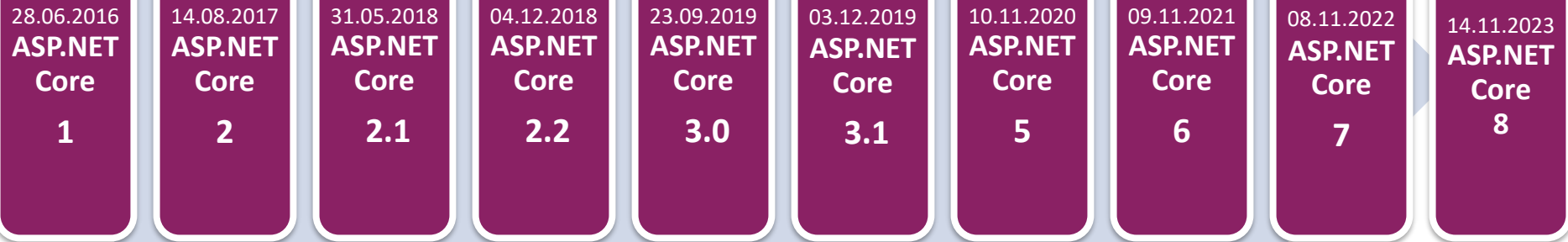
Was ist MVC?

- **Model-View-Controller**
- Entwurfs-/Architektur zur Strukturierung von Software-Projekten
- Erstmals 1979 in Smalltalk
- Ziel ist die Separation of Concerns
 - Datenmodell (Model)
 - Präsentation (View)
 - Steuerung (Controller)

Model-View-Controller



Versionshistorie ASP.NET Core MVC



Unterschiede ASP.NET Web Forms & MVC

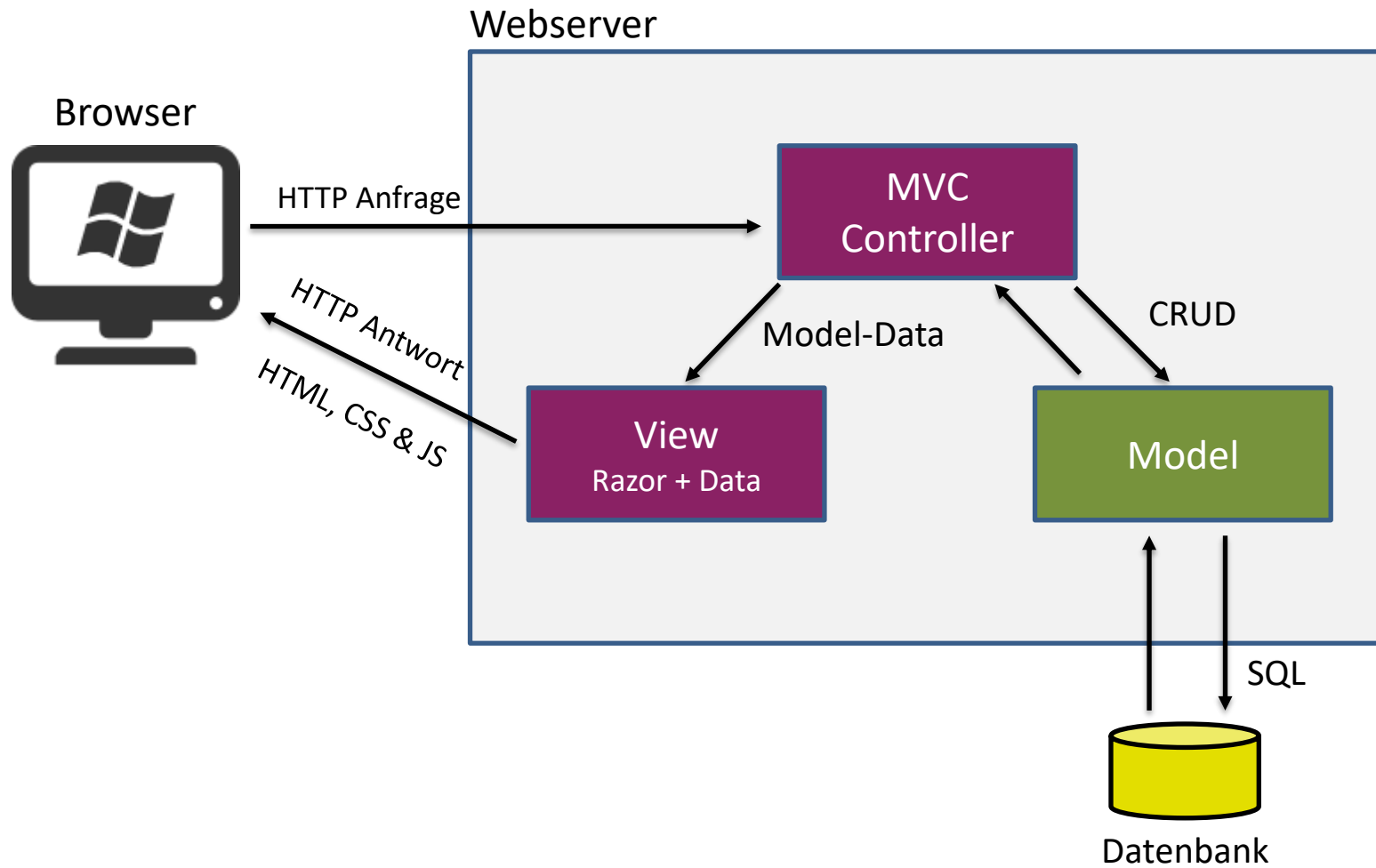
| Web Forms | MVC |
|-------------------------------|----------------------------------|
| Control und ereignisbasiert | Separation Of Concerns |
| Rich UI-Controls | Keine Toolbox |
| Adaptives Rendering | Volle Kontrolle über HTML / Urls |
| .aspx Dateien und Code-behind | .cshtml und .cs Dateien |
| RAD Development | Unit Testing / TDD |
| ViewState / Postbacks | Kein ViewState / Postback |



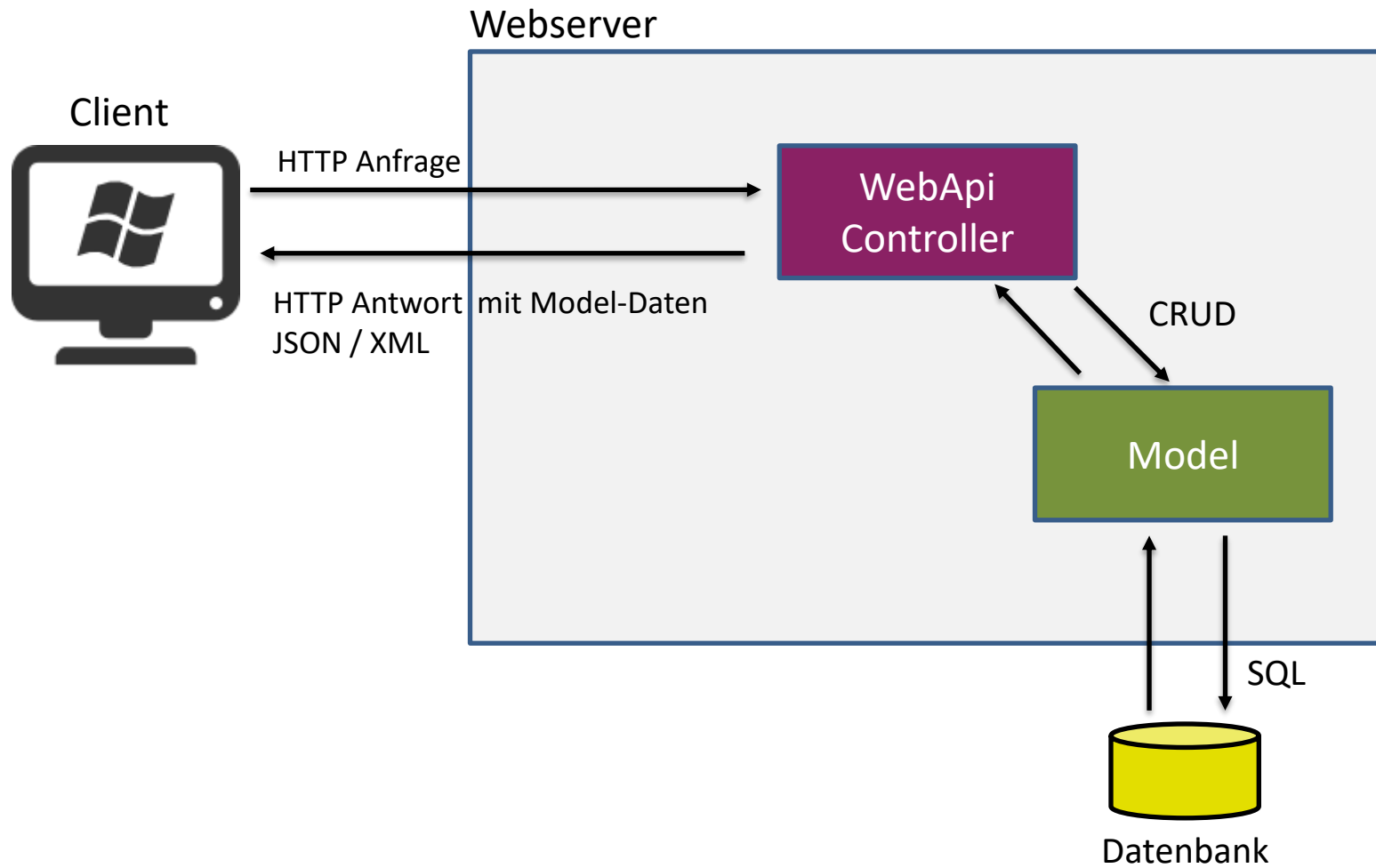
Projekt „Quote Of The Day“

Controller & Actions

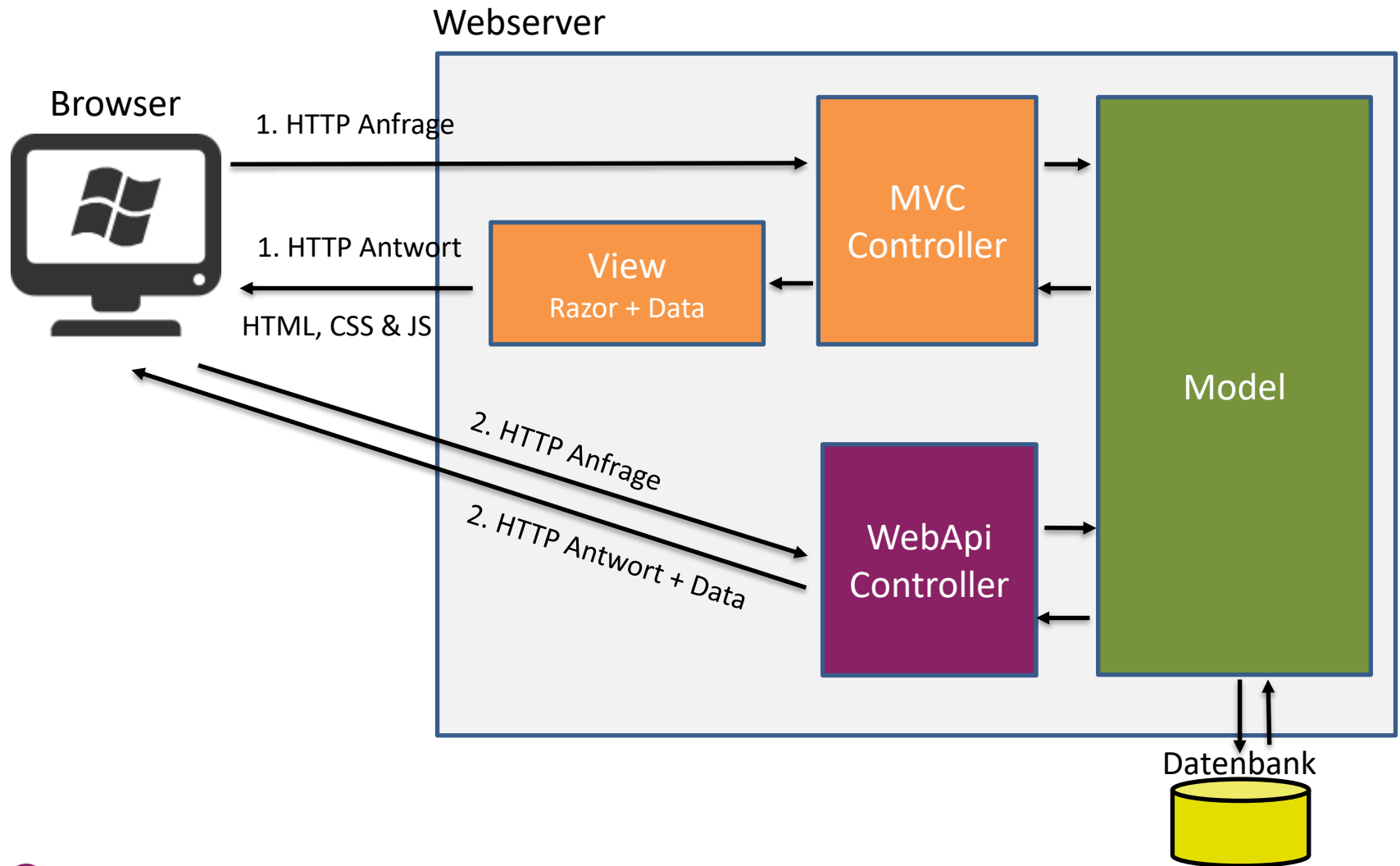
Request-Response Ablauf ASP.NET Core MVC



Request-Response-Ablauf ASP.NET Core WebApi

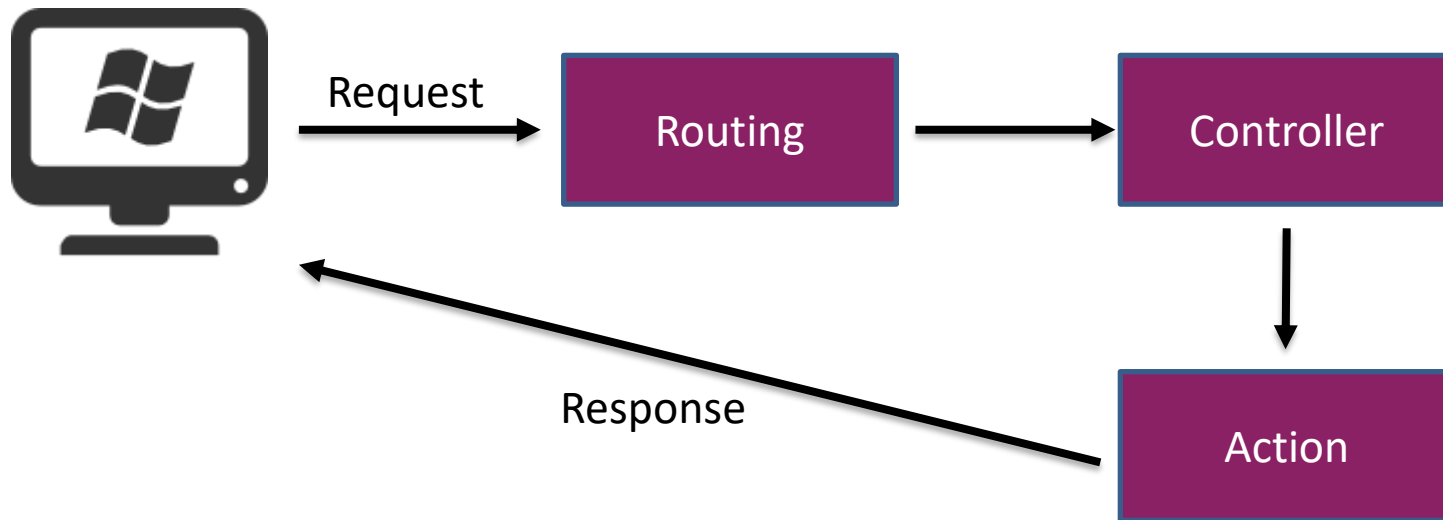


Request-Response Single Page Application



Was ist die Aufgabe eines Controller?

- Ausführung einer Aktionsmethode (meist CRUD)



- Ein **ActionResult** ist das Ergebnis einer **ActionMethode** welches die HTTP-Antwort zum Client darstellt
- Implementiert die Schnittstelle **IActionResult**
- Folgende Rückgabetypen können zurückgegeben werden
 - void (Keine Rückgabe)
 - IActionResult (und abgeleitete ActionResult)
 - anderen Typ

Action-Result Typen

| Rückgabetyp | Beschreibung |
|-------------------------|---|
| Content | Liefert den angegebenen String als Ergebnis der Anfrage zurück |
| File | Veranlasst einen Download als Ergebnis |
| JavaScript | Gibt den angegebenen String als JavaScriptCode zurück, der auf dem Client ausgeführt wird |
| Json | Liefert das angegebene Modell JSON-formatiert zurück |
| PartialView | Verwendet eine partielle View, um einen Teil einer Seite zu rendern |
| RedirectToAction | Leitet die aktuelle Anfrage an die angegebene Action-Methode um |
| Empty | Tut nichts! |
| RedirectToRoute | Leitet die aktuelle Anfrage zu dem URL um, der zur angegebenen Route passt |
| View | Ruft eine View auf |

Actions mit Parametern

- Der **DefaultModelBinder** ermittelt den Namen des Parameters und ordnet ihn der Action-Methode zu wegen Namensgleichheit
- <http://www.webseite.de/kunde/getkunde?name=schmidt>

DefaultModelBinder

```
public IActionResult GetKunde(string name)
{
    var kunde = (from k in db.Kunden
                  where k.Name == name
                  select p).FirstOrDefault();

    return View("Details",kunde);
}
```

- 2 Möglichkeiten um Daten an View zu übergeben
 - **View() Helper Methoden** bei Modeldaten
 - **ViewBag/ViewData** Eigenschaft im Falle von fehlenden Modeldaten

Controller

```
ViewBag.Uhrzeit = DateTime.Now;  
ViewBag.Message = "Dies ist ein schöner Tag";
```

View

```
<p>  
    Die Nachricht des Tages lautet: @ViewBag.Message  
    Heute ist der @ViewBag.Uhrzeit  
</p>
```

Eigene Action Results

- Eigene Klassen müssen von **IActionResult** oder **ActionResult** erben und **ExecuteResultAsync** bzw. **ExecuteResult**(synchron) implementieren
- Alternative erben von vorhandenen ActionResults

```
public class NoContentResult : IActionResult
{
    public Task ExecuteResultAsync(ActionContext context)
    {
        return Task.FromResult(context.HttpContext.Response.StatusCode =
                                StatusCodes.Status204NoContent);
    }
}
```

- Vereinfachte Implementierung ab .NET 4.5 mit `async/await`

```
public async Task<IActionResult> GetAutor(int id)
{
    var autor = await db.Autoren.FindAsync(id);

    if (autor == null)
    {
        return NotFound();
    }

    return View(autor);
}
```

Views

Razor, Layout, Html Helper,
Bootstrap, TagHelper,
ViewComponents

Was ist eine View?

- Ansichten / Views sind in ASP.NET Core MVC Vorlagen in HTML und einer speziellen CodeSyntax bzw. View-Engine namens RAZOR @
- Zwecks Übersichtlichkeit und Wiederverwendbarkeit kann ein Template weitere untergeordnete Templates beinhalten
- Dateien der Views haben die Endung cshtml
- Ansichten haben eine **Model** Eigenschaft, dessen Typ durch @model deklariert wird

Razor-Syntax

| Syntax | Beschreibung |
|--------------------------------|---|
| Code Block | <pre>@ { int x = 666; string y = "Dies ist ein Text"; }</pre> |
| Ausdruck (HTML Codiert) | <pre>@model.Message</pre> |
| Ausdruck (uncodiert) | <pre> @Html.Raw(model.Message) </pre> |
| Kombinieren Text und markup | <pre>@foreach(var item in items) { @item.Eigenschaft }</pre> |
| Code and Klartext mischen | <pre>@if (foo) { <text>Klartext</text> }</pre> |

Razor-Syntax II

| Syntax | Beschreibung |
|---|--|
| Using block | <pre>@using (Html.BeginForm()) { <input type="text" value="input hier"> }</pre> |
| Code and Klartext mischen (alternativ) | <pre>@if (foo) { @:Plain Text is @bar }</pre> |
| Escape @ | Hi philha@@example.com |
| Kommentar | <pre>@* This is a server side multiline comment *@</pre> |
| Generische Methode aufrufen | <pre>@(MyClass.MyMethod<AType>())</pre> |
| URL Auflösung mit ~ | <pre><img src="~/images/myImage.jpg" wird ersetzt zu <img src="/meineApp/images/myImage.jpg"</pre> |



Razor Syntax

ASP.NET MVC Models

- Was ist ein Model?
- Erstellung von Models
- Data Annotations
- Validation
- Entity Framework

Eine Klasse



Wer nutzt das Model?

- Benutzer
 - Mit welchen Daten wird interagiert?
- ASP.NET MVC
 - Eingabesteuerelemente
 - Darstellung
 - Validation

Model

| Author |
|------------------------|
| - Id : Guid |
| - Name : string |
| - Description : string |
| - BirthDate: DateTime |

```
public class Author
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description {get;set;}
    public DateTime? BirthDate {get;set;}
}
```

- Steuern die Anzeige und die Validation von Modelleigenschaften (Properties)
- System.ComponentModel.DataAnnotations
- Built-In-Data-Annotations für Anzeige

| Anzeige Attribute | Beschreibung |
|-------------------|---|
| DataType | Legt den Datentyp fest |
| Display | Legt den Anzeigenamen fest |
| DisplayFormat | Legt fest wie Datenfelder dargestellt und formatiert werden |
| UIHint | Bestimmt das Control für die Modelleigenschaft |

Data Annotations für Anzeige

| Data Type Enumeration | Beschreibung |
|-----------------------|---------------------------------------|
| CreditCard | Repräsentiert eine Kreditkartennummer |
| Currency | Repräsentiert eine Währung |
| Date | Repräsentiert ein Datum |
| DateTime | Repräsentiert ein Datum mit Uhrzeit |
| EmailAddress | Repräsentiert eine Email-Adresse |
| Html | Repräsentiert eine HTML-Datei |
| MultilineText | Repräsentiert mehrzeiligen Text |
| Password | Repräsentiert ein Passwort |
| Text | Repräsentiert Text |
| Time | Repräsentiert eine Uhrzeit |
| Url | Repräsentiert eine URL |

Model mit Data Annotations

| Author |
|----------------------|
| - FirstName : string |
| - Email : string |
| - Descript : string |
| |

```
public class Author
{
    [Display(Name="Ihr Name")]
    public string Name { get; set; }

    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }

    public string Descript { get; set; }
}
```



Data Annotation für Anzeige

Validierung

- Steuern die Anzeige und die Validation von Modelleigenschaften (Properties)
- System.ComponentModel.DataAnnotations
- Built-In-Data-Annotations für Anzeige

| Anzeige Attribute | Beschreibung |
|-------------------|---|
| DataType | Legt den Datentyp fest |
| Display | Legt den Anzeigenamen fest |
| DisplayFormat | Legt fest wie Datenfelder dargestellt und formatiert werden |
| UIHint | Bestimmt das Control für die Modelleigenschaft |

Model mit Data Annotations

| Author |
|------------------------|
| - AuthorId : Guid |
| - Name : string |
| - Description : string |
| - BirthDate: DateTime |

```
public class Author
{
    public Guid AuthorId { get; set; }

    [Display(Name="Name des Autors")]
    public string Name { get; set; }

    public string Description {get;set;}

    [DataType(DataType.Date)]
    public DateTime? BirthDate {get;set;}
}
```

- 5 Möglichkeiten der Model-Validierung
 - Built-In-Attribute
 - Benutzerdefinierte Attribute
 - Selbstvalidierendes Model
 - Clientseitig mit JQuery Validator Library
 - Remote

Built-In-Validierungsattribute

| Validation Attribute | Beschreibung |
|--------------------------|---|
| Compare | Prüft, ob zwei Eigenschaften denselben Wert haben. (z.B. Passwort) |
| CreditCard | Prüft, ob der zu validierende Wert eine Kreditkartennummer ist |
| EmailAddress | Prüft auf Email-Format |
| MaxLength | Prüft auf eine maximale Länge |
| MinLength | Prüft auf eine minimale Länge |
| Length | Prüft auf minimal- & maximale Länge |
| Range | Prüft, ob sich der zu validierende Wert in einem bestimmten Wertebereich befindet |
| DeniedValues | Verbotene Eingabewerte |
| RegularExpression | Validiert die Eigenschaft mit regulären Ausdrücken |
| Required | Markiert die Eigenschaft als Pflichtfeld |

Model mit Validierungsattribute

| Autor |
|--------------------------|
| - AutorId : int |
| - Name : string |
| - Beschreibung : string |
| - Geburtsdatum: DateTime |

```
public class Autor
{
    public int AutorId { get; set; }

    [Required(ErrorMessage="Bitte ...")]
    [StringLength(50)]
    [Display(Name="Name des Autors")]
    public string Name { get; set; }

    [StringLength(50)]
    public string Beschreibung {get;set;}

    [DataType(DataType.Date)]
    public DateTime? Geburtsdatum {get;set;}
}
```

- Ableitung von Basisklasse **ValidationAttribute** und überschreiben von "**IsValid**" Methode

```
public class NoAdminAttribute : ValidationAttribute
{
    protected override ValidationResult IsValid(object value,
                                                ValidationContext validationContext)
    {
        string name = (string) value;
        if(name.ToLower() != "admin" && name.ToLower() != "administrator")
        {
            return ValidationResult.Success;
        }
        var error = "Der Wert darf nicht Admin oder Administrator sein";
        return new ValidationResult(error);
    }
}
```

Benutzerdefinierte Validierungsattribute

| Autor |
|--------------------------|
| - AutorId : int |
| - Name : string |
| - Beschreibung : string |
| - Geburtsdatum: DateTime |

```
public class Autor
{
    public int AutorId { get; set; }

    [Required(ErrorMessage="Bitte ...")]
    [StringLength(50)]
    [NoAdmin]
    [Display(Name="Name des Autors")]
    public string Name { get; set; }

    [StringLength(50)]
    public string Beschreibung {get;set;}

    [DataType(DataType.Date)]
    public DateTime? Geburtsdatum {get;set;}
}
```

- Überprüfen, ob Model korrekt validiert wurde mit Eigenschaft **ModelState.IsValid** und optional **Fehlermeldung**

```
public IActionResult Post(Autor autor)
{
    if(!ModelState.IsValid)
    {
        ModelState.AddModelError("Name","So ein Pech auch");
        return BadRequest(ModelState);
    }
    return View(autor);
}
```

- Vorteil von vereinten Validierungen in einer Klasse
- Nachteil von nicht mehrfach verwendbaren Validierungen
- Schnittstelle **IValidateObject** muss implementiert werden

Selbstvalidierendes Model

```
public class Teilnehmer : IValidateObject
{
    ...
    [StringLength(50)]
    //[NoAdmin]
    public string Nachname { get; set; }
    ...
}

public IEnumerable<ValidationResult> Validate(ValidationContext
                                              validationContext)
{
    List<ValidationResult> errors = new List<ValidationResult>();
    if (Nachname.ToLower() == "admin" || Nachname.ToLower() ==
                                           "administrator")
    {
        errors.Add(new ValidationResult("Der Nachname darf nicht Admin
                                         oder Administrator sein"));
    }
    return errors;
}
```

- Notwendige JQuery-Bibliotheken
 - jquery-{version}.js
 - jquery.validate.js
 - jquery.validate.unobtrusive
- Normalerweise in Visual Studio Projektvorlage schon vorhanden

- Kombination aus clientseitiger Validierung mit serverseitiger Validierung
- Validierung via AJAX-Anfrage
- NuGet Paket **Microsoft jQuery Unobtrusive Ajax** benötigt
 - `jquery.unobtrusive-ajax.js`

- Erstellen einer Controller Action, die **JsonResult** zurückgibt

```
public JsonResult IsEmailUnique(string email)
{
    if(TeilnehmerAusDatenbank.Count(c => c.Email == email) == 0)
        return Json(true); //OK Email noch nicht vorhanden
    else
        return Json(false); //Fehler Email schon vergeben
}

public class Teilnehmer
{
    [Remote("IsEmailUnique","ControllerName", ErrorMessage="...")]
    public string Email { get; set;}
}
```

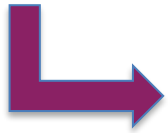


Validierung

- Unterstützen das Rendern von Html-Elementen in Views
- Diverse Built-in Helper
- Nutzen die Modelattribute für
 - Namensanzeige
 - Formatierung
 - Eingabeelemente
- Erstellen von eigenen HtmlHelper

■ `Html.ActionLink()`

```
@Html.ActionLink("Klicken Sie hier", "Anzeige", new { id = 1 })
```



```
<a href="/Home/Anzeige/1">Klicken Sie hier</a>
```

■ `Url.Action()`

```

```

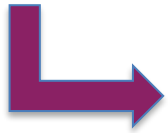


```

```

- **Html.DisplayNameFor()**

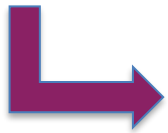
```
@Html.DisplayNameFor(model => model.FullName)
```



Der vollständige Name

- **Html.DisplayFor()**

```
@Html.DisplayFor(model => model.FullName)
```



Max Mustermann

■ Html.LabelFor()

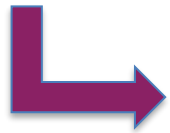
```
@Html.LabelFor(model => model.FullName)
```



```
<label for="FullName">Full Name </label>
```

■ Html.EditorFor()

```
@Html.EditorFor(model => model.IsAdministrator)
```



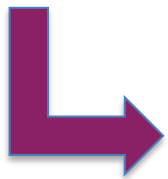
```
<input type="checkbox" name="IsAdministrator">
```

Speziellhelper EditorFor

| Control | Model Klasseneigenschaft | Gerendertes HTML von EditorFor |
|----------------------------|--|---|
| TextBox | <code>public string Titel { get; set; }</code> | <code><input type="text" name="Titel"></code> |
| Mehrzeilige Textbox | <code>[DataType(DataType.MultilineText)] public string Beschreibung { get; set; }</code> | <code><textarea name="Beschreibung" rows="5" cols="2"></code> |
| Checkbox | <code>public bool IstAdministrator { get; set; }</code> | <code><input type="checkbox" name="IstAdministrator"></code> |

■ Html.BeginForm()

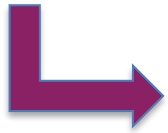
```
@using(Html.BeginForm("Erstelle", "Kunde", FormMethod.Post,  
new { enctype = "multipart/form-data"}))  
{  
    ... Eingabefelder ...  
}
```



```
<form action="/Kunde/Erstelle" method="post"  
        enctype="multipart/form-data">  
  
</form>
```


■ `Html.Validation()`

`@Html.ValidationSummary()`



``

`Bitte geben Sie einen Nachname ein`

`Bitte geben Sie eine gültige Emailadresse ein`

``

■ `Html.ValidationMessageFor()`

`@Html.ValidationMessageFor(model => model.Email)`



Bitte geben Sie ein gültige Emailadresse ein

- Helper sind Methoden um wiederkehrende Aufgaben auszulagern

```
@helper RenderFirmenLogo(string logoName)
{
    
}

<div class="logo">
    @RenderFirmenLogo("ilci.png")
</div>
```

- Erweiterung der HtmlHelper-Klassen durch selbstdefinierte Helper-Klasse

```
public static class HtmlHelperExtensions
{
    public static IHtmlContent Truncate(this IHtmlHelper helper,
        string text, int length = 20)
    {
        if (text.Length > length)
            text = text.Substring(0, length) + "...";

        return new StringHtmlContent(text);
    }
}
```



Html Helper

TagHelper

- Was sind TagHelper?
- Built-In TagHelper

Was sind TagHelper?

- TagHelper sind Klassen, die Html-Elemente verändern, um sie z.B. mit zusätzlichen Inhalten zu ergänzen oder sie mit neuem Inhalt zu ersetzen (ähnlich Angular-Direktiven)
- In der *_ViewImports.cshtml* müssen sie registriert werden (Assembly Name)
 - `@addTagHelper *`, Microsoft.AspNetCore.Mvc.TagHelpers
- Zwei Methoden zum Überschreiben **Process** & **ProcessAsync**
- Deaktivierung für individuelle Elemente
 - `<!span asp-validation-for..></!span>`
- `@tagHelperPrefix` möglich

HtmlHelpers

```
@Html.Label(„Vorname“, new { @class = „caption“})
```

WebControls

```
<asp:Label runat=„server“ ID=lblVorname“></asp:Label>
```

TagHelpers

```
<label class=„caption“ asp-for=„Vorname“></label>
```


Wie werden TagHelper aktiviert?

- `AttributeName`
- `Html Element Tag`
- `Eltern Element`

TagHelper Process Methode

- Razor View Engine sucht nach validen Tag Helfern
 - `<my-custom info=„Bill“></my-custom>`
- Führt die Process Methode aus

```
public class MyCustomTagHelper : TagHelper
{
    public string Info { get; set; }

    public override void Process(TagHelperContext context, TagHelperOutput output)
    {}
}
```

Verbindung von Html Element Attributen zu C# Properties

- Html Attribute

```
<auto model=„911“ marke=„porsche“></auto>
```

- C# Property

```
public class AutoTagHelper : TagHelper  
{
```

```
    public string Model { get; set; }
```

```
    public string Marke { get; set; }
```

//alt. Syntax mit Namen des Html-Attributes

```
[HtmlAttributeName(„model-name“)]
```

```
public string Model { get; set; }
```

```
}
```



TagHelper

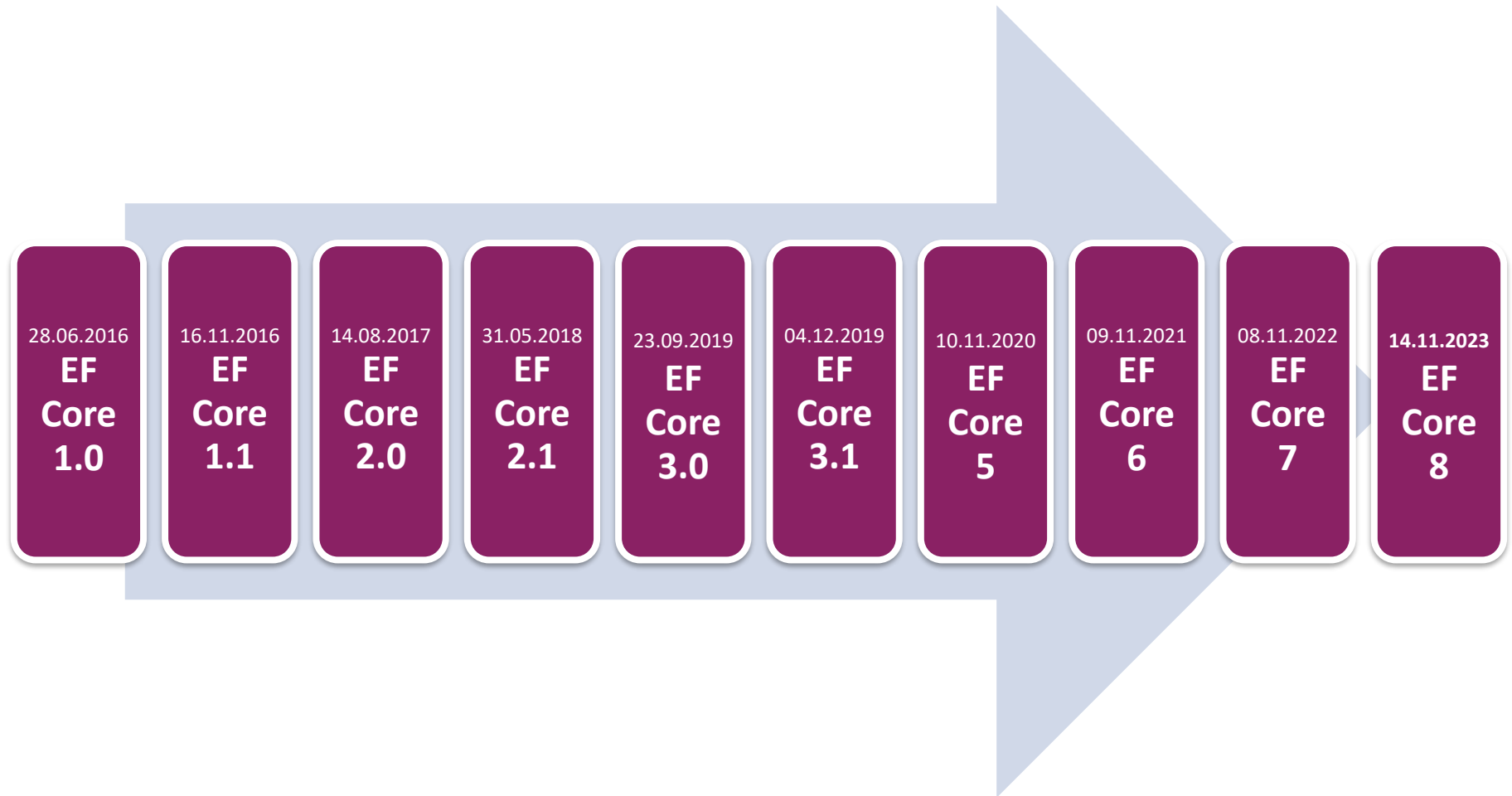
Exkurs: Entity Framework Core 8

■ ORM-Framework

- Bildet Objekte auf eine relationale DB ab
- Web Forms, MVC, WPF, WCF, Web API
- Unabhängig von verwendeter Datenbank



Versionshistorie Entity Framework



Ohne Entity Framework

- Fehleranfällig, Unsicher, Kompliziert

```
string con = "Data Source=.;Initial Catalog=AutorDB;IntegratedSecurity=True";
string cmd= "INSERT INTO Author (LastName,FirstName) VALUES ('Twain', 'Mark')";

using (SqlConnection connection = new SqlConnection(connString))
{
    using(SqlCommand com= new SqlCommand(cmdString))
    {
        com.Connection= con;
        connection.Open();
        com.ExecuteNonQuery();
    }
}
```


Mit Entity Framework

- Objekte <> Tabellen Abbildung
- Schutz vor SQL-Injections
- Vermeidung von Syntaxfehlern
- ConnectionString in Konfigurationsdatei

```
using (var db = new AuthorDBContext())  
{  
    db.Authors.Add(new Author { LastName = "Twain", FirstName = "Mark" });  
    db.SaveChanges();  
}
```

- Domain-Model
 - Konzeptionelles Model, welches ein Geschäftsproblem darstellt, aber noch keine technische Lösung
- Objekt – Modell
 - Eine Klasse, die die Eigenschaften eines realen oder abstrakten Objektes beschreibt
- ER – Modell
 - Visualisiertes Datenmodell
- DB – Modell
 - Datenbankdiagramm einer physischen Datenbank

- Entity
 - = Tabelle in Datenbank
 - Eigenschaften = Spalten der Tabellen
- Relationship
 - Fremdschlüsselbeziehungen in der Datenbank
- Context
 - Repräsentiert den Zugriff auf die Datenbank
 - CRUD

Model Workflows EF Core

| Vorgehensweise | Datenbank vorhanden | Beschreibung |
|--------------------------------|---------------------|--|
| Code First | Nein | Objekte und deren Beziehungen werden in Klassen beschrieben. Daraus wird dann mit dem EF die DB erstellt |
| Reverse Engineering Code First | Ja | Generierung von Entity Framework Klassen aus der DB |

- NuGet-Paket-Manager
 - Ab VS 2012 standardmäßig installiert
 - Als Erweiterung für VS 2010 verfügbar
- NuGet-Konsole
 - Install-Package EntityFrameworkCore

Code First Entitäten

- Für jede Entity eine Klasse erstellen

```
public class Author
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public virtual List<Quote> Quotes { get; set; }
}
```

```
public class Quote
{
    public Guid Id { get; set; }
    public string QuoteText { get; set; }
    public int AuthorId { get; set; }
    public virtual Author Author { get; set; }
}
```

Beziehung 1 : n

Code First Data Context

- Erstellen eines EF Data Context
- Data Context => DB
- Jede Entity **T** eine Eigenschaft **DbSet<T>** hinzufügen

```
public class QuoteContext : DbContext
{
    public QuoteContext(DbContextOptions<QuoteContext> options) :
        base(options)
    {}

    public DbSet<Author> Authors => Set<Author>();

    public DbSet<Quote> Quotes => Set<Quote>();
}
```

Verwenden EF in Controller

- DataContext instanziiieren
- Gewünschte Entitäten abfragen (Filtern möglich)
- Als Model an View übergeben

```
public class AuthorsController
{
    private QuoteContext db = new QuoteContext();

    public IActionResult GetAuthors()
    {
        var authors = db.Authors.ToList();
        return View(authors);
    }
}
```


- Löst das Problem Code und Datenbank synchron zu halten
- Möglichkeit von Snapshots
 - Generierung von Scripten um Code/Datenbank zu synchronisieren
 - Migrierung & Scripting SQL diverser Snapshots
- NuGet-Konsole
 - **Add-Migration** fügt neue Migration ein
 - **Update-Database** synchronisiert Modell mit Datenbank
 - **Script-Migration** erstellt SQL-Script

EF Core Commands

Installation des Tools

dotnet tool install --global dotnet-ef

<https://docs.microsoft.com/de-de/ef/core/miscellaneous/cli/dotnet>

| Package Manager Console | dotnet CLI | Beschreibung |
|---|--|---|
| add-migration <migrationsname> | dotnet ef migrations add <migrationsname> | Erstellt eine Migration mit Migrations Snapshot |
| Remove-migration | dotnet ef migrations remove | Entfernt die letzte hinzugefügte migration |
| Update-database | dotnet ef database update | Aktualisiert die |
| Script-migration | Dotnet ef migrations script | Generiert ein SQL Script mit allen Migrationen |

Layout

- Vorlage für mehrere Webseiten (Masterlayout)
- Webseite kann mehrere Masterlayouts besitzen
- Standardmasterlayout für die Views wird in der **ViewStart.cshtml** festgelegt oder individuell in der View
- Standardlayout => Views/Shared/_Layout.cshtml
- Twitter Bootstrap Standard

_Layout.cshtml

```
<!DOCTYPE html>
<html>
  <head>
    <meta name= "viewport" content= "width=device-width" />
    <title>@ViewData["Title"]</title>
  </head>
  <body>
    <div>
      <h3>Views / Shared / Layout.cshtml</h3>
      @RenderBody()
    </div>
    @RenderSection("scripts", false)
  </body>
</html>
```

_Layout.cshtml - Elemente

| Element | Beschreibung |
|---|---|
| @ViewBag.Title bzw. @ViewData[„Title“] | Titel der individuellen Content-Views |
| @RenderBody | Rendering der Content-View (darf nur einmal vorhanden sein) |
| @Render.Section | Aufteilung von inhaltlich zusammenhängenden Layout-Regionen |

- Templates aus HTML und Razor-Code
- Können Bestandteile von Layout-Views, Standard-Views und partiellen Views sein
- Zweck ist die Extrahierung von Funktionalitäten wegen Übersicht, Austauschbarkeit und Wiederverwendung
- Möglichkeit der Controllerzuordnung (im Controller-Ordner) oder Anwendungszuordnung (Shared-Ordner)
- Konvention: vorangehender Underscore im Namen

- Mit HTML Helpers werden partielle Views aufgerufen
 - **@Html.Partial** für partielle Views mit gleichem Eltern Model
 - **@Html.Action** für partielle Views mit unterschiedlichem Eltern Model
 - `@{await Html.RenderPartialAsync("_AuthorsList");}`
 - `<partial name="_AuthorsList" model="Model" />`
- ViewBag/ViewData um Daten zwischen Controller Aktion, Eltern-View und Partielle-View zu teilen

- Verschiedene Möglichkeiten Mobile Views zu erstellen
 - CSS MediaQueries (manuell)
 - Bootstrap (Framework)
 - JQuery Mobile

ViewComponents

- ViewComponents sind Klassen ähnlich PartialViews die Fragmente von Html-Ausgaben darstellen. Es sind wiederverwendbare Webbausteine
- Sie besitzen eine Elternseite bzw. werden von ihr aufgerufen
- Sie besteht aus einer Klasse abgeleitet von *ViewComponent* und normalerweise einer View
- Typische Anwendungsfälle
 - Tag Cloud, Einkaufswagen, letzter Blogartikel

ViewComponents

- Suffix ViewComponent
- Unterstützt Constructor DI
- Kein Bestandteil des Controllerlebenszyklus und somit keine Filterverwendung möglich
- Invoke bzw. InvokeAsync – Methode, die ein `IViewComponentResult` zurückgibt
- Keine Modellbindung
- Pfade
 - `/Pages/Components/<viewname>`
 - `Views/<controllername>/Components/<ansichtskomponentenname>/<ansichtsname>`
 - `Views/Shared/Components/<ansichtskomponentenname>/<ansichtsname>`

- Aufruf durch

`@await Component.InvokeAsync(„VIEW
NAME“, anonymous type)`

- Alternative

`<vc:[view-component-name]></vc:[view-component-name]>`



ViewComponents



Die „K“-Frage für Webentwickler



Künstler



Klempner

Was ist Bootstrap?

- Open Source Front-End Framework von Twitter
- Mobile First Responsive Web Design
- Modular
- Besteht aus CSS und JavaScript
- Vielfalt an Designkomponenten
- Standard-Designvorlage seit MVC 5

Mobile First Responsive Design

Graceful Degradation



Progressive Enhancement

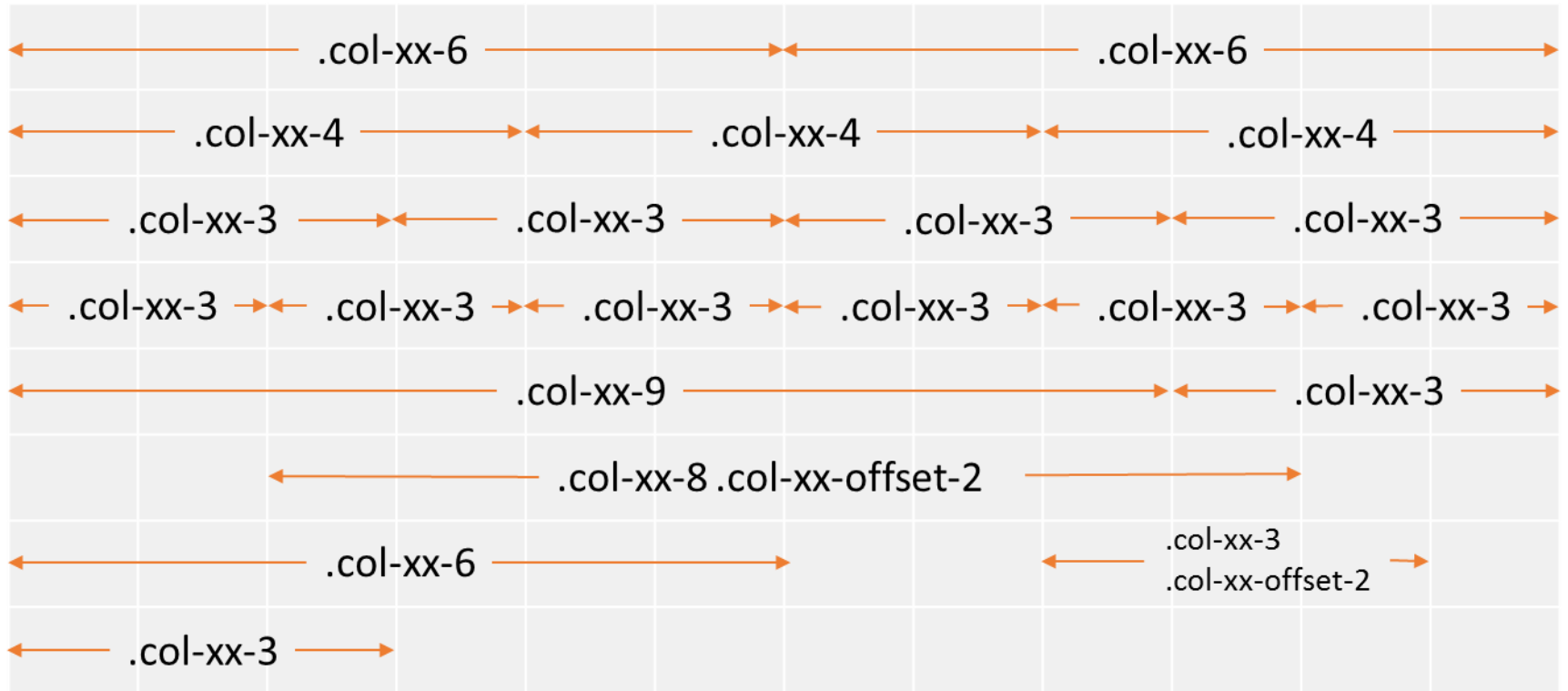


Quelle: <http://www.deepblue.com>

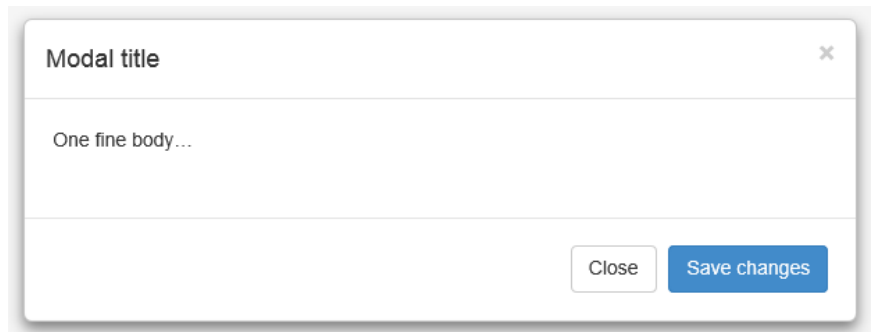
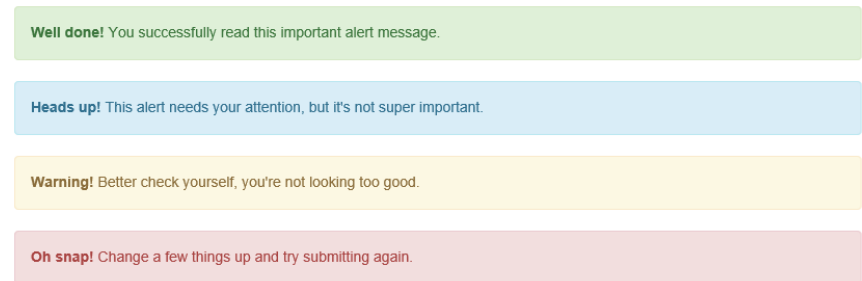
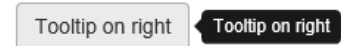
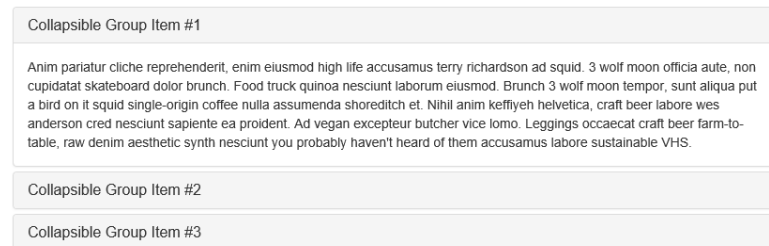
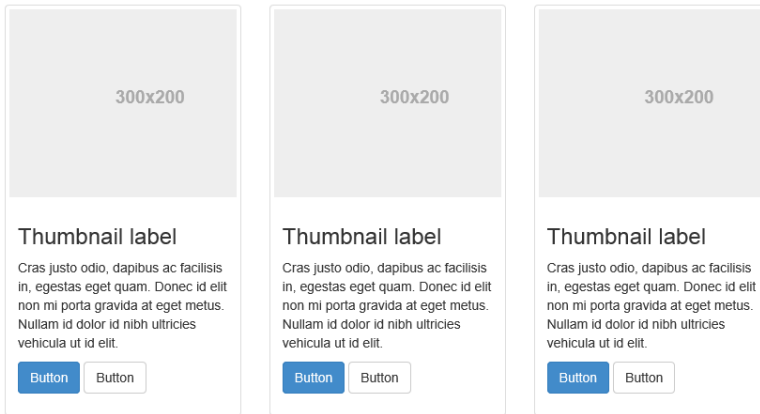
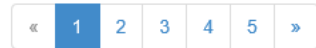
Grid System

| | xs <576px | sm ≥576px | md ≥768px | lg ≥992px | xl ≥1200px | xxl ≥ 1400 |
|---------------------|-----------------------------------|-----------------|-----------------|-----------------|-----------------|------------------|
| Max container width | None (auto) | 540px | 720px | 960px | 1140px | 1320px |
| Class prefix | .col- | .col-sm- | .col-md- | .col-lg- | .col-xl- | .col-xxl- |
| # of columns | 12 | | | | | |
| Gutter width | 1.5rem (.75rem on left and right) | | | | | |
| Custom gutters | Yes | | | | | |
| Nestable | Yes | | | | | |
| Column ordering | Yes | | | | | |

12 Column Grid System



Bootstrap Komponenten Auszug

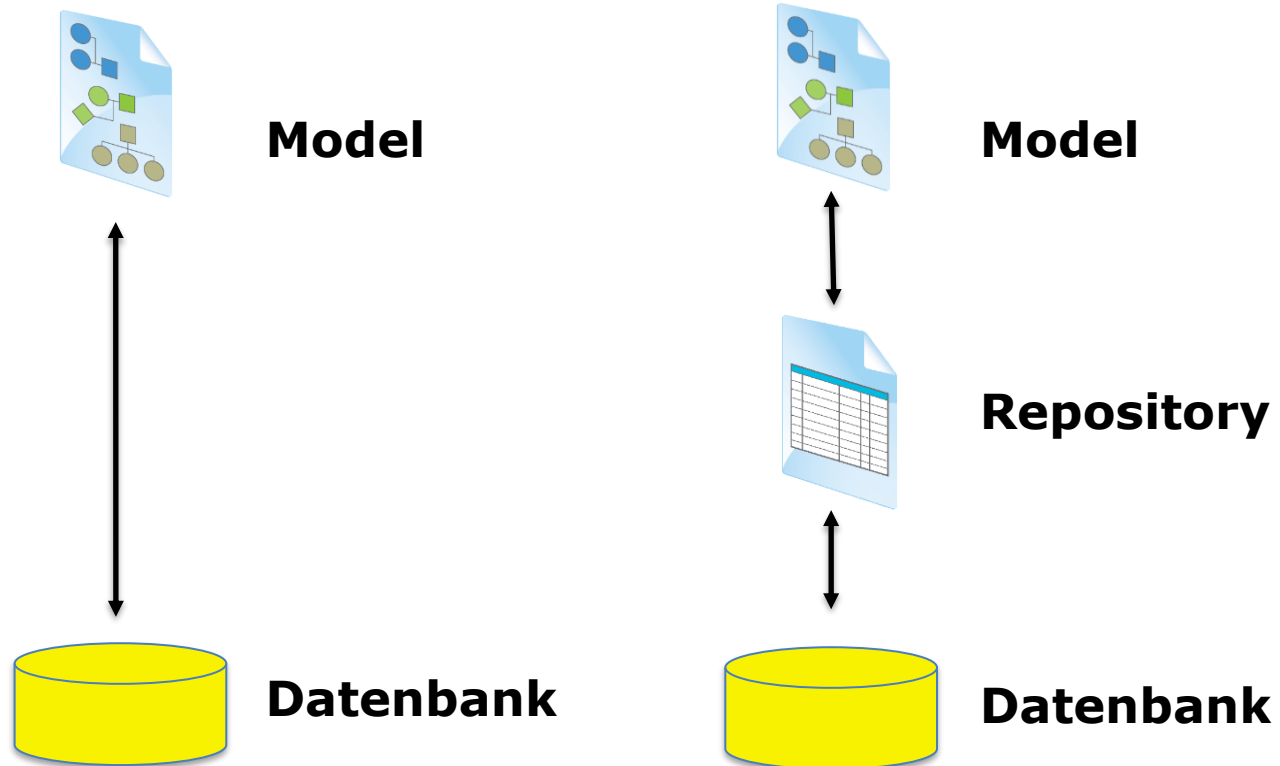


- Bootstrap 5
 - <http://getbootstrap.com/>
- Themes
 - <http://bootswatch.com/> (Free)
 - <https://wrapbootstrap.com/> (Kommerziell)
- Online Editor
 - <http://bootply.com/>
- Fuel UX
 - <http://earmbrust.github.io/dieselui/#>

Repository

Repository

- Separation of Concerns
- Lose Kopplung

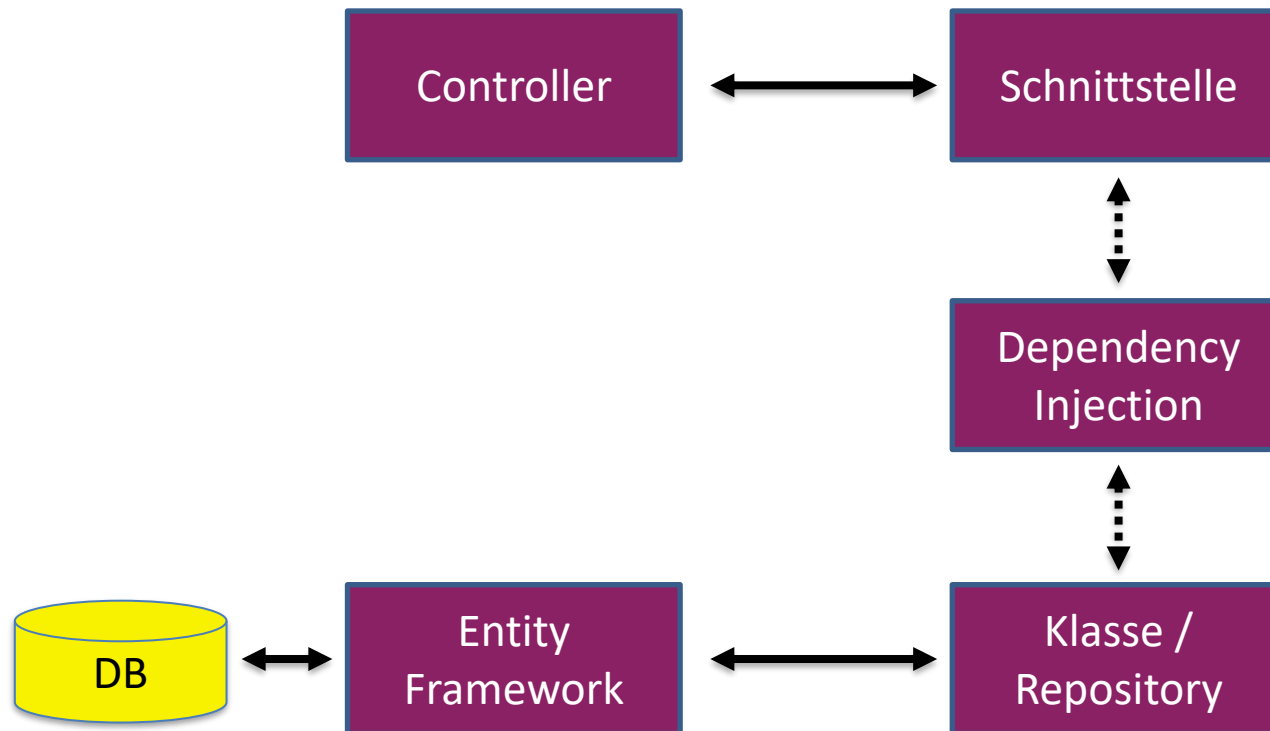


Dependency Injection

IoC

- *Inversion of Control* delegates the function of selecting a concrete implementation type for a class's dependencies to an external component
- *Dependency Injection* is a specialization of the IoC Pattern. The Dependency Injection pattern uses an object – the container – to initialize objects and provide the required dependencies to the object

Zugriff mit Dependency Injection



Dependency Injection Beispiel

```
public class AuthorsController : Controller
{
    private readonly IAuthorRepository _authorRepository;

    protected AuthorController(IAuthorRepository authorRepository)
    {
        _authorRepository = authorRepository;
    }
}
```

Interface, *keine* konkrete Implementierung

Konstruktor Injection

- Ziel ist die Flexibilität und Testbarkeit zu steigern
- Erleichtert das Austauschen & Testen von Abhängigkeiten
- Built-In-ASP.NET Core
- Populäre DI-Frameworks
 - Microsoft Unity
 - Ninject
 - StructureMap

- BuilderServices in Program verantwortlich für die Bereitstellung der Services

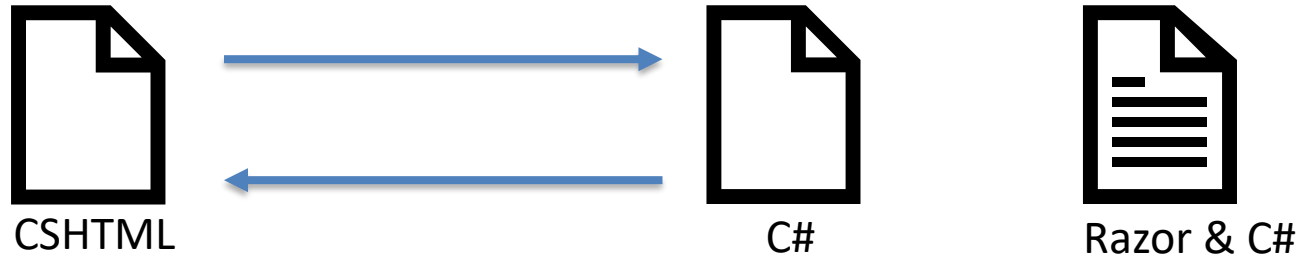
| Scope | Bedeutung | Erläuterung |
|------------------|------------------------|--|
| Transient | vorübergehend | Eine neue Instanz des Typs wird bei jeder Anforderung des Typs verwendet. |
| Scoped | bereichsbezogen | Eine neue Instanz des Typs wird bei seiner erstmaligen Anforderung in einer bestimmten HTTP-Anforderung erstellt und anschließend für alle nachfolgenden Typen verwendet, die während dieser HTTP-Anforderung aufgelöst werden |
| Singleton | übergreifend | Eine Instanz des Typs wird ein Mal erstellt und von allen nachfolgenden Anforderungen für diesen Typ verwendet. |



DI

Razor Pages

- Bieten eine vereinfachte Seitenstruktur ohne MVC-Pattern mit der Razor Syntax
- Design Pattern
 - CSHTML = Razor und HTML Markup
 - C# = Code und Logik
- Standardordner **Pages**
- Html, CSS und C# Kenntnisse erforderlich



Unterschiede ASP.NET Core Razor Pages & MVC

| Razor Pages | MVC |
|----------------------------------|---------------------|
| Page zentriert | Action zentriert |
| Einfachere Struktur | Komplexere Struktur |
| Viele Gemeinsamkeiten mit Blazor | |

Razor Page & Model

```
@page
```

← Page Direktive erforderlich

```
@model IndexModel
```

← Model Direktive verweist auf Klasse

```
<div>  
  <p>@Model.Email<</p>  
</div>
```

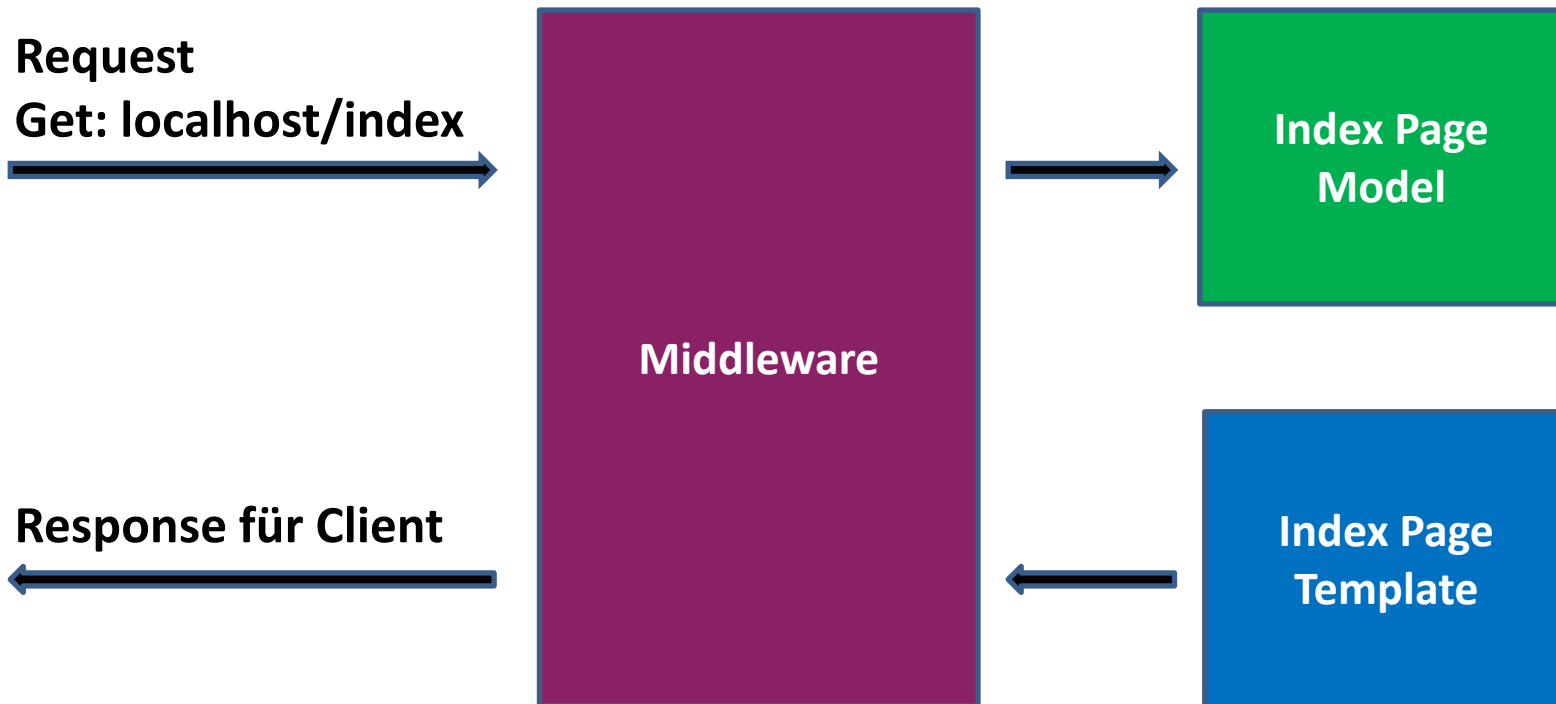
← Model Eigenschaften Zugriff

```
class IndexModel : PageModel  
{  
    public Email {get; set;}  
}
```

← Erbt von PageModel

← Eigenschaft zugreifbar von der Razor Seite

Razor Page Request Pipeline



Page Model Request Handling

| HTTP Request Type | Page Model Convention | Beschreibung |
|-------------------|-----------------------|-----------------------|
| GET | OnGet() | Seite laden |
| POST | OnPost() | Erstellung neue Daten |
| PUT | OnPut() | Änderung Daten |
| DELETE | OnDelete() | Löschen Daten |

Model Binding



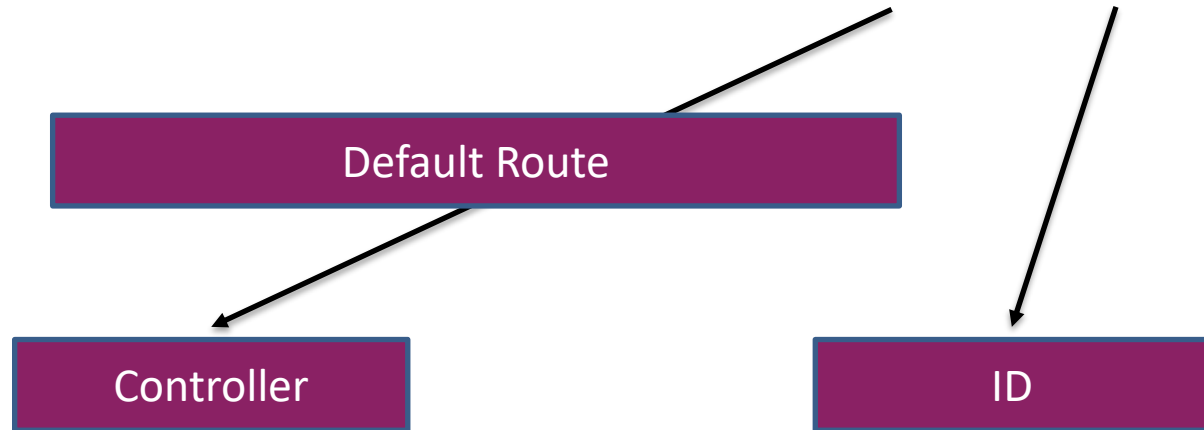


RazorPages

Routing

- ASP.NET Routing Engine
- Hinzufügen & Konfigurieren von Routen
- Parameterübergabe mit Routen

<http://www.webseite.de/api/Autor/3>



- Benutzerdefinierte Routen
 - Einfache URLs / Muss nicht Projekt-Struktur entsprechen
 - Nützlich für SEO

- Routing wird in **Startup.cs** -> **Configure** festgelegt
- Reihenfolge der Abarbeitung wichtig! Wer zuerst kommt, malt zuerst
- Beim Suchen der passenden Route wird die Anzahl der Abschnitte in der URL gezählt und mit der in den Routen verglichen
- Route beinhaltet Name, URL, Constraints and Defaults

Konventionell Routing-Beispiel

Standardroute

```
endpoint.MapControllerRoute(  
    name: „default“,  
    template: \"{controller=Home}/{action=Index}/{id?}\",  
);
```

Benutzdefinierte Route

```
routes.MapRoute(  
    name: "Authors",  
    url: "/authors/authorname/{name}",  
    defaults: new { controller = "Authors" },  
    constraints: new { name = @"[a-zA-Z ]+$" });
```

- Vereinfacht das Routing
- Empfohlen für ASP.NET Core WebAPI
- Features:
 - Route Constraints
 - Custom Route Constraints
 - Route Names

Attribut Routing Beispiel

```
[Route("authors")]
public class AuthorsController : Controller
{
    [Route] // z.B ./authors
    public IActionResult Get() { return Ok(authors); }

    [Route("{id}")] // Opt. Para. z.B /authors/5
    public IActionResult Get(int id) { return Ok(author);}

    [Route("authorname/{name}", Name = "GetAuthorByName")]
    // z.B /authors/authorname/Albert%20Einstein
    public IActionResult GetAuthorByName(string name)
    { return Ok(); }
}
```

Attribut Routing Constraints

| Constraint | Beschreibung | Beispiel |
|------------|--|------------------------------------|
| alpha | Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z) | {x:alpha} |
| bool | Matches a Boolean value. | {x:bool} |
| datetime | Matches a DateTime value. | {x:datetime} |
| decimal | Matches a decimal value. | {x:decimal} |
| guid | Matches a GUID value. | {x:guid} |
| int | Matches a 32-bit integer value. | {x:int} |
| length | Matches a string with the specified length or within a specified range of lengths. | {x:length(6)} {x:length(1,20)} |
| max | Matches an integer with a maximum value. | {x:max(10)} |
| maxlength | Matches a string with a maximum length. | {x:maxlength(10)} |
| min | Matches an integer with a minimum value. | {x:min(10)} |
| minlength | Matches a string with a minimum length. | {x:minlength(10)} |
| range | Matches an integer within a range of values. | {x:range(10,50)} |
| regex | Matches a regular expression. | {x:regex(^\\d{3}-\\d{3}-\\d{4}\$)} |

- Möglichkeit eigene Routing Constraints zu definieren für komplexere Logik
 1. Implementierung von **IRouteConstraint**
 2. Registrieren in **ConfigureServices**
 3. Action mit benutzerdefiniertem Attribut markieren

Custom Routing Constraints Beispiel

1. Implementierung

```
public class AuthorDescriptionConstraint : IRouteConstraint
{
    private readonly string[] _validAuthorDescription;
    public AuthorDescriptionConstraint(string options)
    {
        _validAuthorDescription = options.Split(',');
    }

    public bool Match(HttpContext httpContext, IRouter route, string routeKey,
        RouteValueDictionary values, RouteDirection routeDirection)
    {
        object value;
        if (values.TryGetValue(routeKey, out value) && value != null)
        {
            return _validAuthorDescription.Contains(value.ToString(), StringComparer.OrdinalIgnoreCase);
        }
        return false;
    }
}
```

Custom Routing Constraints Registrierung

2. Registrieren

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    //Custom Route Constraints
    services.AddRouting(options =>
    {
        options.ConstraintMap.Add("authordescription",typeof(AuthorDescriptionConstraint));
        ....
    }
}
```

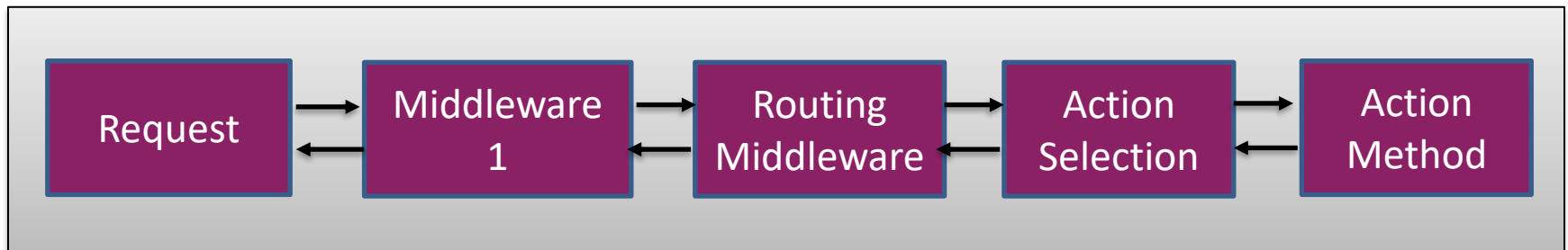
3. Action Attribute

```
[Route(„authordescription/{description:authorDescription(Dozent,IT-Trainer)}")]
public IActionResult ShowOnlyDozenten(string description)
{
    var authors = db.Authors.Where(c => c.Description.Contains(description));
    return Ok(authors);
}
```

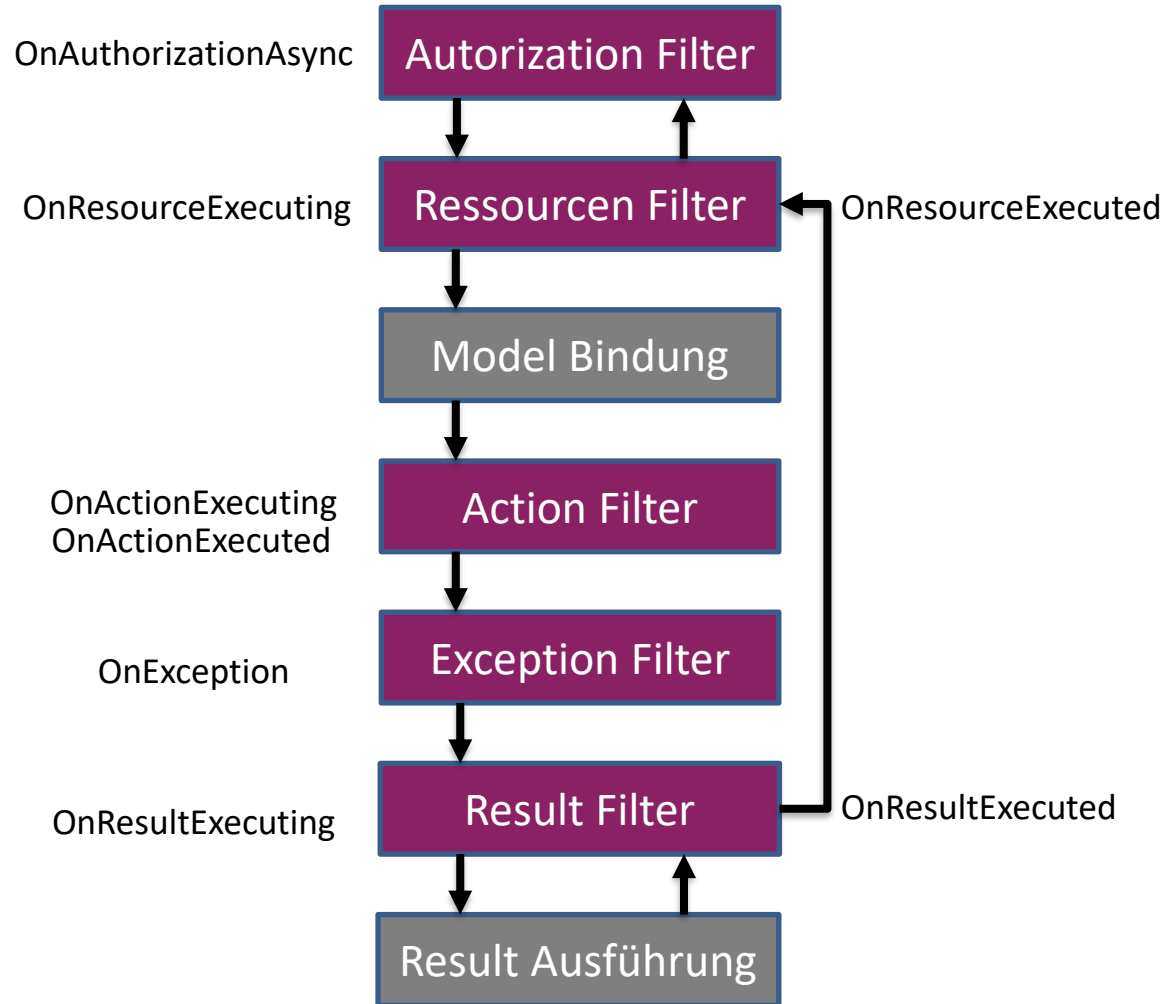
Filter

Filter

- Filter bieten eine Möglichkeit **vor** und **nach** dem Ausführen von Aktionen **vordefinierte** oder **benutzerdefinierte** Logiken/Methoden aufzurufen
- Filter könne auf Aktions-, Controller- und Anwendungs- Ebene sein
- 5 Built-in Filterarten



Reihenfolge Filterausführung



Built-In-Filterarten

| Filtertyp | Schnittstelle | Methode | Beschreibung |
|----------------------|--|---|---|
| Authorization | IAsyncAuthorizationFilter | OnAuthorizationAsync | Ausführung vor dem Beginn einer Aktion zwecks Autorisierung |
| Resource | IResourceFilter/ IAsyncResourceFilter | OnResourceExecuting OnResourceExecuted | Werden am Anfang/Ende einer Anforderung aktiv, zum Bearbeiten einer Anforderung |
| Action | IActionFilter | OnActionExecuting OnActionExecuted | Ausführung vor/nach einer Aktion |
| Exception | IExceptionHandler/ IAsyncExceptionHandler | OnException | Ausführung, wenn eine Ausnahme im Controller auftritt |
| Result | IResultFilter / IAsyncResultFilter | OnResultExecuting OnResultExecuted | Ausführung vor/nach dem Aktionsergebnis |

Action Filter Beispiel

```
public class SampleActionFilter : IActionFilter
{
    ...
    public void OnActionExecuting(ActionExecutingContext context)
    {
        //before action
        _logger.LogInformation($"Routing-Informationen: Controller ->
                                {context.RouteData.Values["controller"]} " +
                                $"## Action -> {context.RouteData.Values["action"]}");
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        //after action
        _logger.LogInformation($"Erfolgreich ausgeführt");
    }
}
```

- Authorize
 - Kontrolliert Wer den Controller/Action zugreifen darf via Eigenschaften Users / Roles
- ValidateAntiForgeryToken
 - Schützt vor cross-site request forgery (csrf)
 - Verlangt das Hinzufügen von Anti-Forgery Token zur View
- RequireHttps
 - Verlangt SSL



Filter

Tracing / Logging

- Built-In-Tracing/Logging durch ILoggerFactory der Microsoft.Extensions.Logging bzw. ILogger<T> mit DI
 - Aktivierung in Configure
 - `loggerFactory.AddConsole();`
 - Feature Dependency Injection
 - Benutzerdefiniertes Logging möglich durch 3rd Party-LoggingProvider
- Loggingmöglichkeiten
 - NLog
 - Serilog



Logging mit NLOG

Fehlerbehandlung

- Verschiedene Fehlerbehandlungsmöglichkeiten
 - Action-Ebene (**ExceptionHandlerAttribute**)
 - Controller-Ebene (**ExceptionHandlerAttribute**)
 - Global durch **ExceptionHandler** oder **ExceptionHandlerAttribute**
 - Spezielle Entwickler-Exception-Middleware
 - `Configure -> app.UseDeveloperExceptionPage();`

ExceptionFilterAttribute

```
public class ResourceRemovedAttribute : ExceptionFilterAttribute
{
    public override void OnException(ExceptionContext context)
    {
        if (context.Exception is ResourceRemovedException)
        {
            context.Result = new StatusCodeResult(StatusCode.Status410Gone);
        }
    }
}
```

[ResourceRemoved]

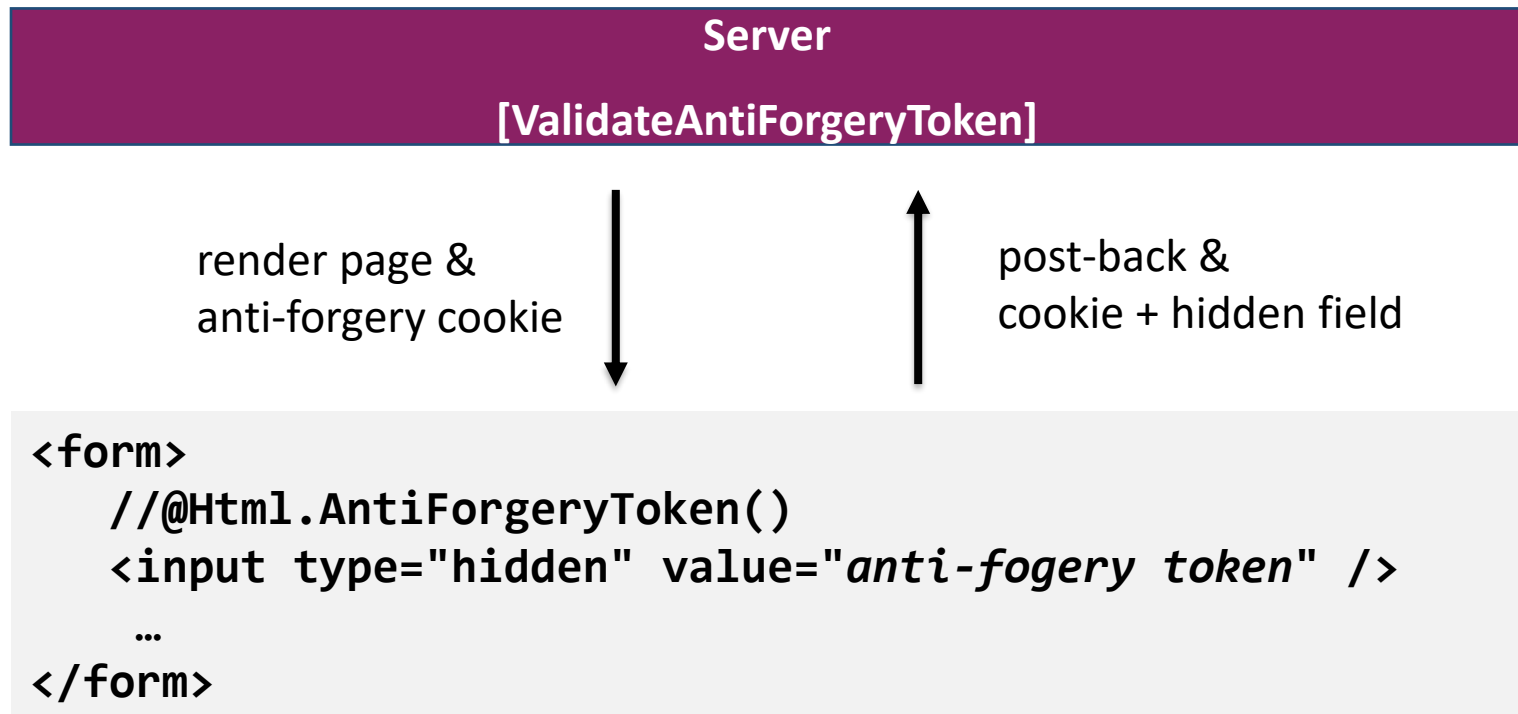
```
public IActionResult Delete(int id)
{
    ... auslösen der ResourceRemovedException
}
```

- Beruht auf dem Einschleusen von JavaScript Code in einer dynamischen Webanwendung
- Die Ursache ist das Zusammensetzen von Ausgaben aus Platzhalter und Benutzereingaben
- Wird verhindert indem die Eingabe/Ausgabe enkodiert (Standard in .NET 4.5.+)
- Für ältere Versionen AntiXSS-Library

- Beruht auf dem Einschleusen von JavaScript Code in einer dynamischen Webanwendung
- Die Ursache ist das Zusammensetzen von Ausgaben aus Platzhalter und Benutzereingaben
- Wird verhindert indem die Eingabe/Ausgabe enkodiert (Standard in .NET 4.5.+)
- Für ältere Versionen AntiXSS-Library

CSRF - Abwehr

- [ValidateAntiForgeryToken] für Action
- @Html.AntiForgeryToken() für View

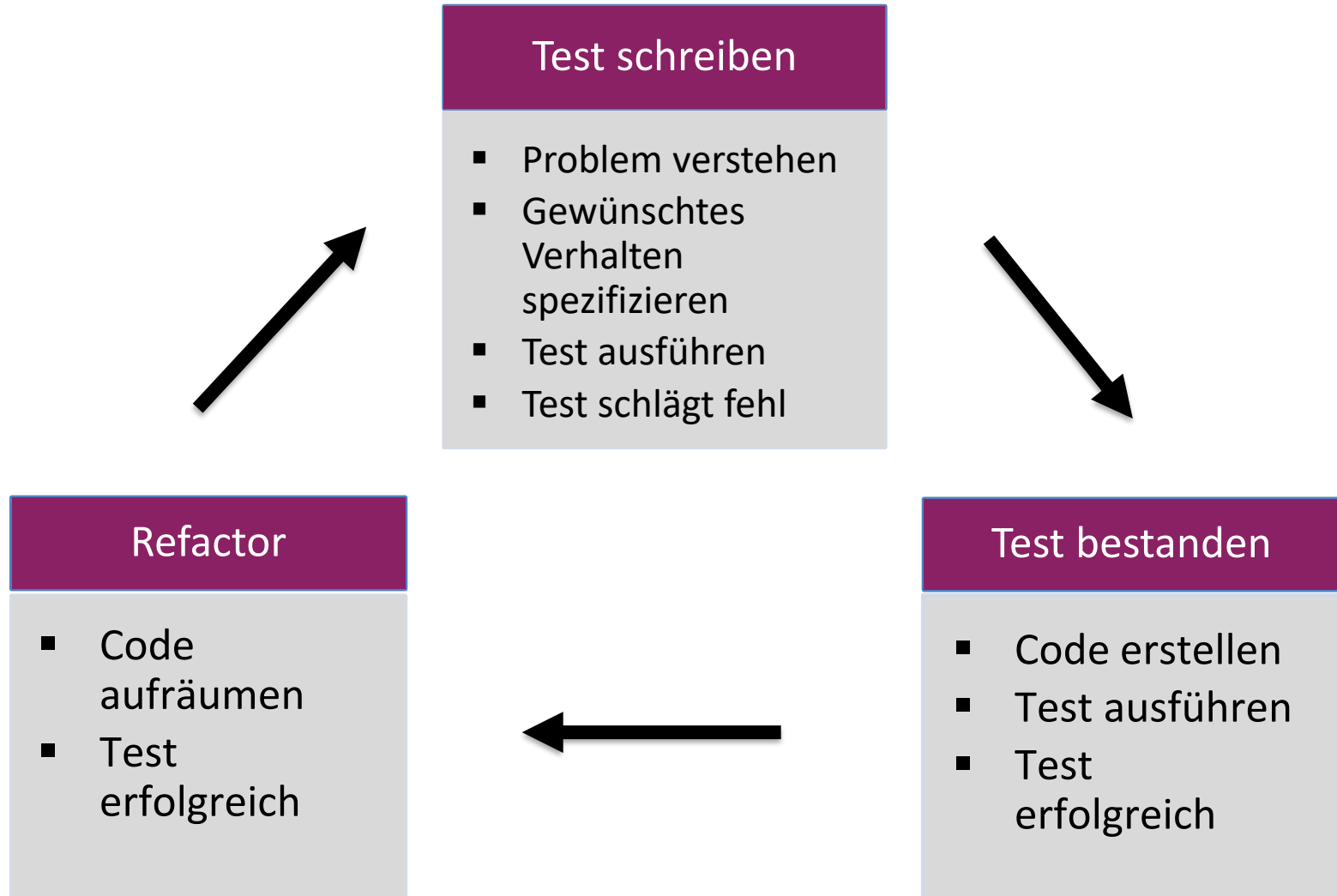


- Ein **Modultest** (auch **Komponententest** oder oft vom engl. unit test als **Unittest** bezeichnet) wird in der Softwareentwicklung angewendet, um die funktionalen Einzelteile (Module) von Computerprogrammen zu testen, d. h. sie auf korrekte Funktionalität zu prüfen.

(Quelle: Wikipedia / <https://de.wikipedia.org/wiki/Modultest>)

- Verschiedene Phasen
 - Arrange (Vorbereitung)
 - Arrangiert Daten auf denen die Test später laufen
 - Act (Ausführung)
 - Ruft die Methoden auf, die getestet werden sollen
 - Assert (Ergebnis)
 - Überprüft, ob das Ergebnis den Erwartungen entspricht
- Visual Studio integriert Testmöglichkeit

Test Driven Development



- Neues Test-Projekt zur Solution hinzufügen
- Controller testen:
 1. Erstellen eines Repository Interface
 2. Implementieren des Repository Interface
 3. Implementierung eines Fake Repositories
 4. Benutzen des Fake Repositories um Controller zu testen

- Ein Mocking Framework automatisiert die Erstellung von Mock Objekten während der Test
- Hauptvorteil der Zeitersparnis beim Schreiben von Unit-Tests
- Verschiedene Frameworks
 - Moq
 - JustMock

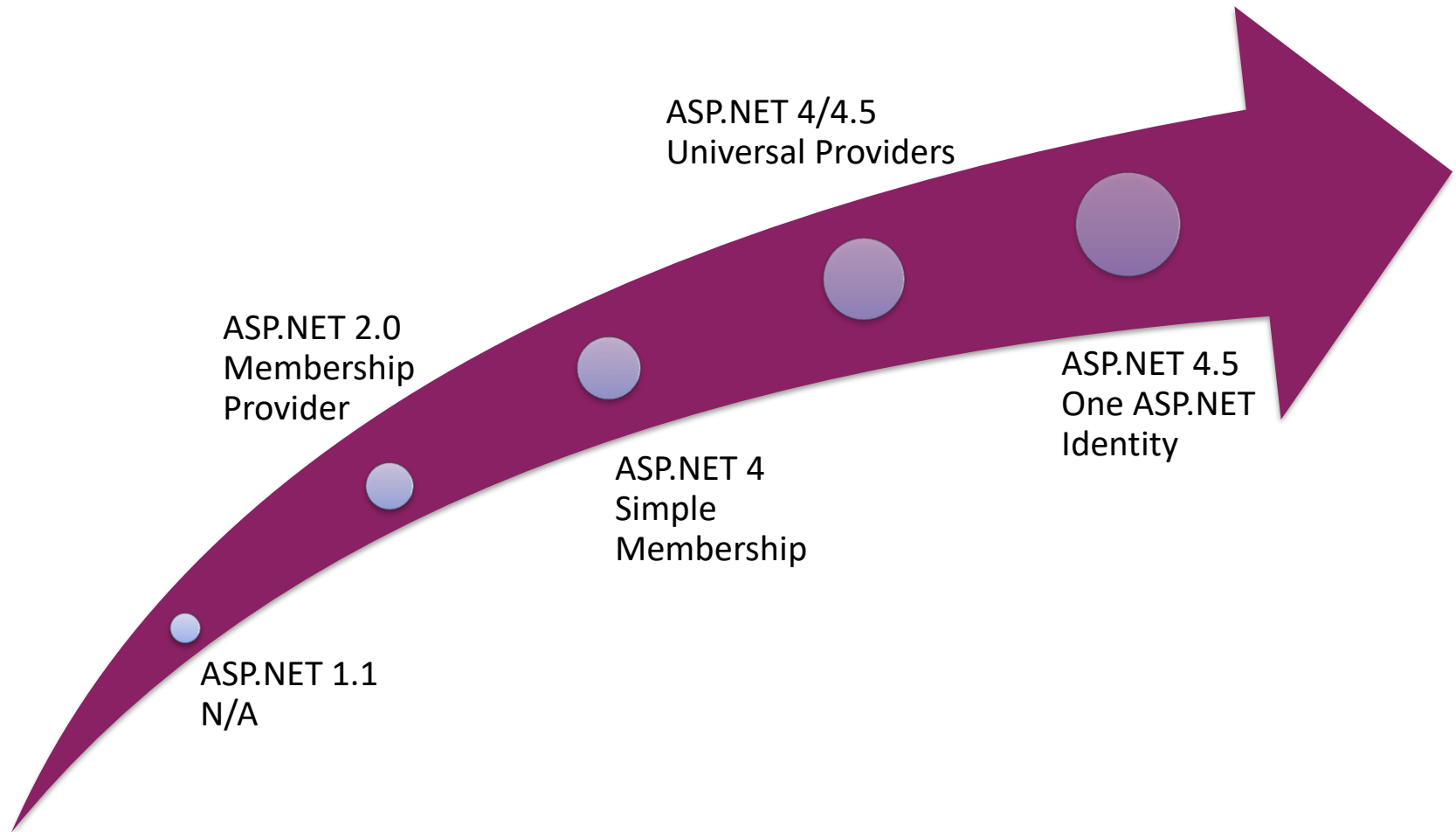


Unit Test / Mocking

Authentifizierung / Autorisierung

- Authentifikation klärt die **WER**-Frage
 - Ist der Benutzer jener Benutzer, der er vorgibt zu sein?
- Autorisierung klärt die **WAS**-Frage
 - Was darf der Benutzer tun?
- ASP.NET Core Identity ist die Basis-Komponente für Authentifizierung/Autorisierung mittels Middleware
 - User-Verwaltung (Registrierung, Login etc...)
 - Persistierung via Entity Framework
 - Umfangreiche Konfigurierungsmöglichkeiten (Regeln, Rollen, Claims, Passwörter, External Providers)

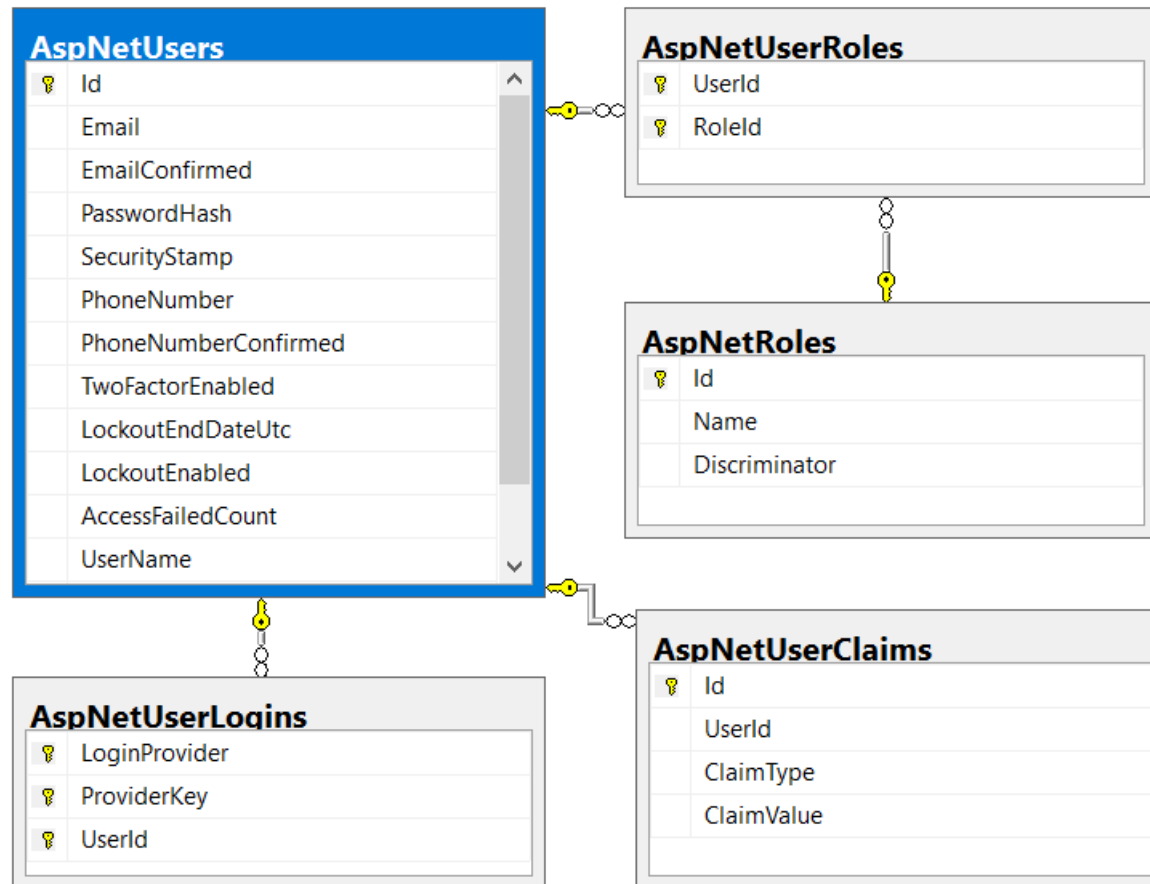
Historie



- One ASP.NET Identity System
 - MVC, Web Forms, WebAPI
- Role Provider
- Claims basierend
- Social Login Providers
- Windows Azure Active Directory
- OWIN Integration
- NuGet package

- Zwei-Factor-Authentifizierung
- Account Lockout
- Account Confirmation
- Password reset
- Erweiterter Password Validator

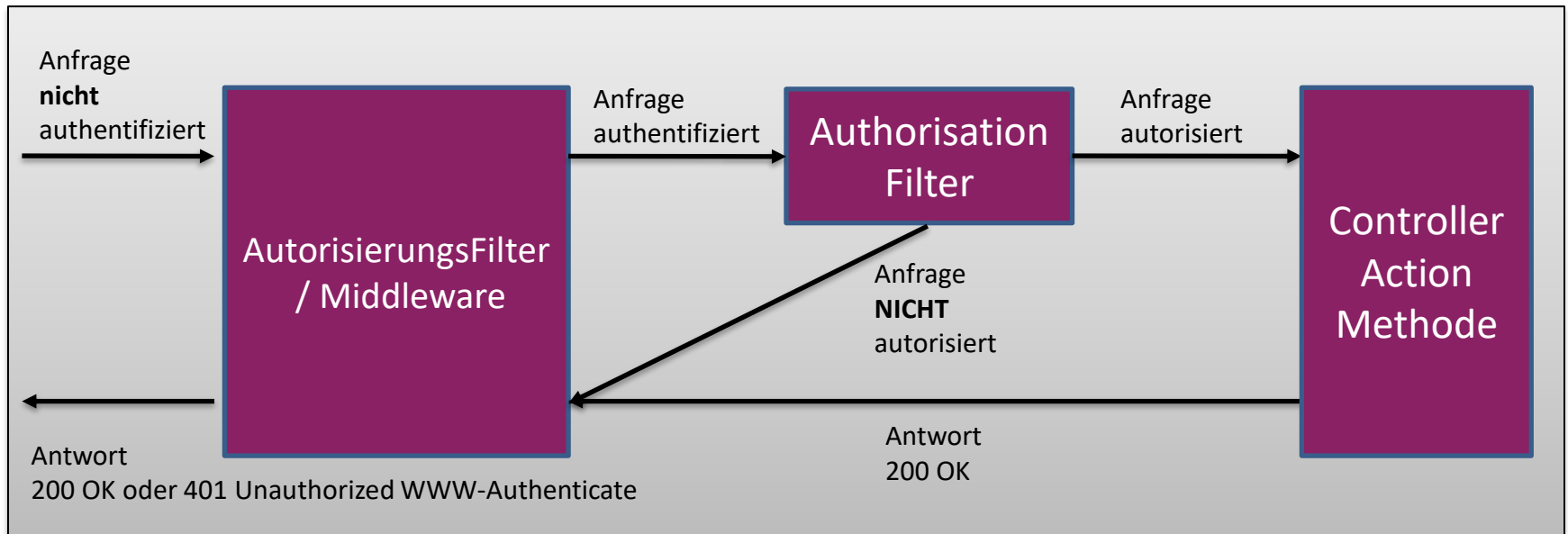
ASP.NET Identity Datenbank Schema



Authentifizierungsmöglichkeiten

| Identity | Technologie | Szenario |
|--|------------------------------------|--------------------------------|
| Individual User Accounts | ASP.NET Identity, Social Providers | Internet |
| Active Directory / Windows Authentifizierung | IIS + Windows Authentifizierung | Intranet |
| Azure Active Directory | OpenID Connect | Cloud basierte Enterprise Apps |
| Keine Authentifizierung | - | Anonymer Zugriff |

Authentifizierungs-/Autorisierungsablauf



- Authorize
 - Kontrolliert Wer den Controller/Action zugreifen darf via Eigenschaften Users / Roles / Claims



Authentifizierung

JavaScript & JQuery

- Bietet Interaktivität für die Webanwendung (Client/Server)
- Grundlage von AJAX
- MVC nutzt JS-Dateien in Views via
 - Inline JavaScript
 - JavaScript-Dateien
- Weitverbreiteste JS-Bibliothek
 - jQuery

- Umfangreiche JavaScript-Bibliothek
- Funktionen zur DOM-Manipulation und – Navigation
- Visuelle Effekte
- Cross-Browser-Kompatibilität (jQuery 1.x)
- April 2013: jQuery 2.0 (aber kein IE 6-8 Support!)
- Juni 2016 JQuery 3.0

- Folgende CSS3 Selektoren stehen zur Auswahl von Elementen
 - Per Name => \$("tr")
 - Per ID => \$("#ElementID")
 - Per CSS-Klasse => \$(".ueberschrift")
- Ereignisbehandlung

```
$(document).ready(function () {  
    $("#HalloButton").click(function (event) {  
        alert("Hallo Welt");  
    });  
})
```

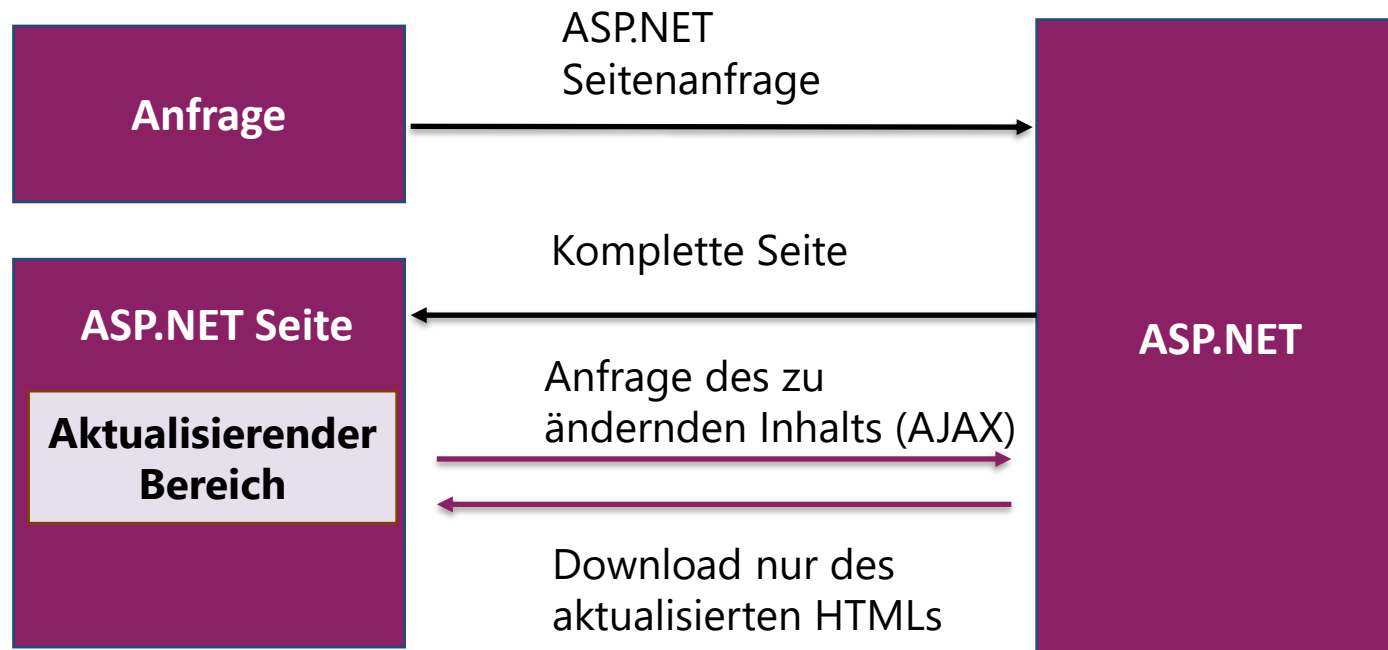
AJAX-Anfragen mit JQuery

```
$.ajax({
    type: "GET",
    dataType: "json",
    url: "Home/GetAutorInfo",
    data: "{ ID: ,123456' }",
    contentType: "application/json; charset=utf-8",
    success: function(data) {
        $("#autorName").html(data.autor.name);
    },
    failure: function(msg) {
        alert(msg);
    }
});
```

- Alternativ Anfrage mit JSON-Rückgabe => **\$.getJSON(...)**

AJAX

- **Asynchronous JavaScript and XML** / Stichwort Web 2.0
- Erlaubt das Aktualisieren nur von gewissen Bereichen einer Webseite



WebAPI

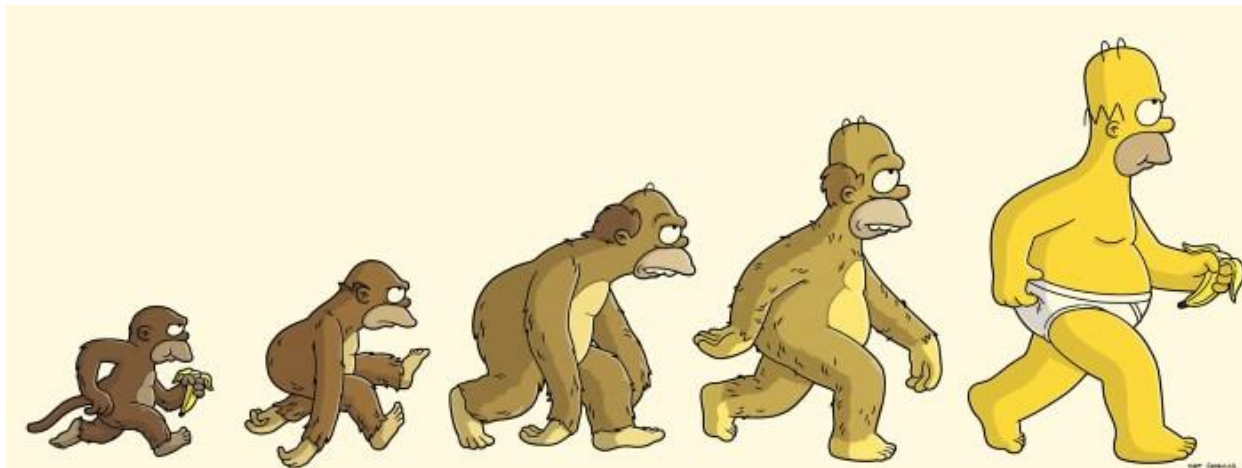
Was ist ein Webservice?

Ein **Webservice** oder **Webdienst** ist eine Softwareanwendung, die über ein Netzwerk für die direkte Maschine-zu-Maschine-Interaktion bereitgestellt wird. Jeder Webservice besitzt einen Uniform Resource Identifier (URI), über den er eindeutig identifizierbar ist, sowie eine Schnittstellenbeschreibung in maschinenlesbarem Format (als XML-Artefakt, meist WSDL), die definiert, wie mit dem Webservice zu interagieren ist. Die Kommunikation kann (muss aber nicht) über Protokolle aus dem Internetkontext wie HTTP laufen und XML-basiert sein

Quelle: <https://de.wikipedia.org/wiki/Webservice> bzw.
<http://www.w3.org/TR/ws-gloss/#webservice>

Wie kann man Nachrichten über das Netzwerk senden?

- **Applikation Protokoll** – komplex und nicht interoperabel
- **Remote Procedure Call (RPC)** - nicht interoperabel 90er
- **XML-RPC** – XML über HTTP `98
- **SOAP** – fügt einen Envelope (Umschlag) zur XML Nachricht `00



SOAP - Beispiel

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"  
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
<soap:Body xmlns:m="http://www.example.org/stock">
```

```
<m:GetStockPrice>
```

```
<m:StockName>IBM</m:StockName>
```

```
</m:GetStockPrice>
```

```
</soap:Body>
```

```
</soap:Envelope>
```

Was ist REST?

- **RE**presentational **S**tate **T**ransfer
- Paradigma der Softwareentwicklung
- Dissertation von Roy Fielding „*Architectural Styles and the Design of Network-based Software Architectures*“ (2000, University of California)
- Ziel ist ein Softwarearchitekturstil für verteilte Hypermedia Systeme (wie das WWW)
- Fielding generalisierte die Webarchitektur-Prinzipien und präsentierte es als ein Framework von Bedingungen (RESTful)

6 Bedingungen der REST-Architektur

(1) Client-Server-Architektur

- Trennung von Client-Server (Separation of Concerns)

(2) Stateless (zustandslose) Interaktionen

- Jede Nachricht enthält alle nötigen Informationen für Client bzw. Server (Isolation)

(3) Cacheability

- Nachrichten vom Server kennzeichnen sich als cacheable oder nicht cacheable

(4) Layered System (Schichtenmodell)

- Client hat keine Infos, ob mit Endserver oder dazwischenliegenden Server verbunden ist

(5) Code on Demand*

- Client-Flexibilität erweitert bzw. passt die Funktionalität des Clients temporär via ausführbaren Code

(6) Uniform Interface

- Jeder Dienst, Methode oder Status besitzt eine eindeutige Adresse

■ Ressource

- Eine Ressource ist jegliche Information, die benannt ist mit einer eindeutigen URI

■ Ressource Repräsentation

- Eine Ressource-Serialization in einem vorgegebenen Format/Media Type wie XML oder JSON

■ Selbstbeschreibende Nachricht

- Jede Nachricht beinhaltet genügend Informationen um zu beschreiben wie die Nachricht prozessiert werden soll

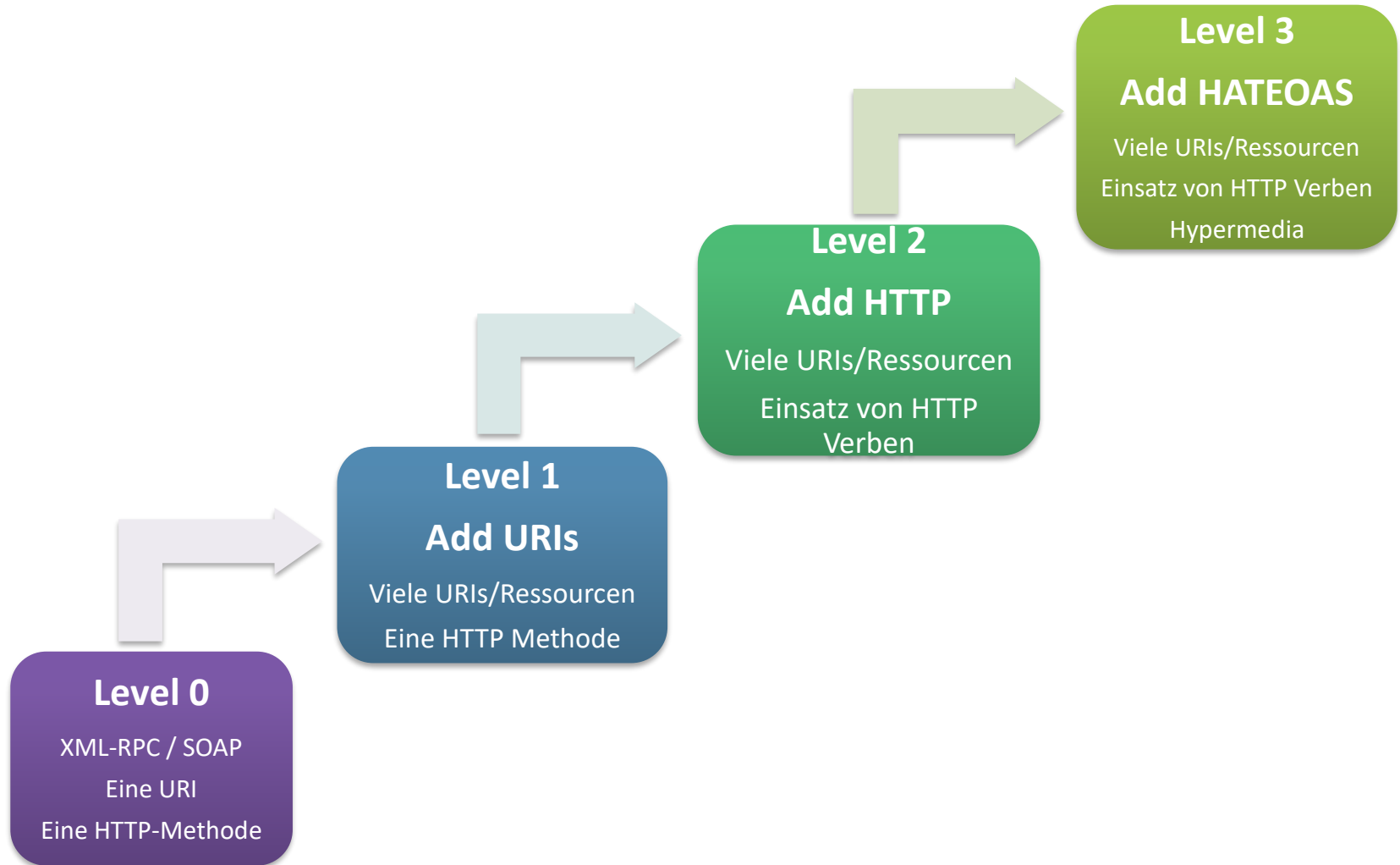
■ Hypermedia

- Verlinkung von Ressourcen zwecks Navigation

HTTP Verben / REST Operationen

| HTTP Verb | Beschreibung |
|---------------|--|
| GET | Gibt einen einzelnen oder eine Sammlung von Einträgen zurück |
| POST | Erzeugt einen neuen Eintrag |
| PUT | Aktualisiert einen bestehenden Eintrag (alle Eigenschaften) |
| PATCH | Aktualisiert einzelne Eigenschaften eines existierenden Eintrags |
| DELETE | Löscht einen Eintrag |

Richardson REST Maturity Model



RMM Level 0 – Beispiel Aufgabe

- Eine URI und eine HTTP-Methode

| Methode | URI | HTTP Verb |
|---------------------|-------------------------|-----------|
| CreateAufgabe | /api/aufgabeService.svc | POST |
| GetAufgabe | /api/aufgabeService.svc | POST |
| GetAufgabeZuordnung | /api/aufgabeService.svc | POST |
| SucheAufgabe | /api/aufgabeService.svc | POST |
| UpdateAufgabe | /api/aufgabeService.svc | POST |

RMM Level 1 – Beispiel Aufgabe

- Viele URIs/Ressourcen und eine HTTP-Methode

| Methode | URI | HTTP Verb |
|---------------------|-------------------|-----------|
| CreateAufgabe | /api/aufgaben | POST |
| GetAufgabe | /api/aufgaben/123 | POST |
| GetAufgabeZuordnung | /api/aufgaben/123 | POST |
| SucheAufgabe | /api/aufgaben | POST |
| UpdateAufgabe | /api/aufgaben/123 | POST |

RMM Level 2 – Beispiel Aufgabe

- Viele URIs/Ressourcen und Einsatz HTTP-Verben

| Methode | URI | HTTP Verb |
|---------------------|-------------------|-----------|
| CreateAufgabe | /api/aufgaben | POST |
| GetAufgabe | /api/aufgaben/123 | GET |
| GetAufgabeZuordnung | /api/aufgaben/123 | GET |
| SucheAufgabe | /api/aufgaben | GET |
| UpdateAufgabe | /api/aufgaben/123 | UPDATE |

RMM Level 3 – Beispiel Aufgabe

- Viele URIs/Ressourcen und Einsatz HTTP-Verben & **Hypermedia As The Engine Of Application State**

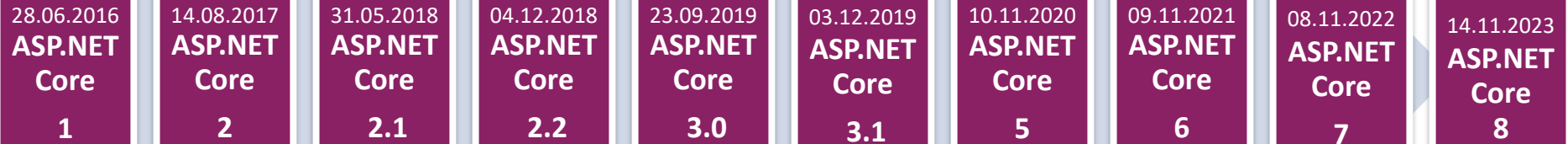
```
<?xml version="1.0" encoding="utf-8"?>
<Aufgaben>
  <Aufgabe Id="123">
    <link rel="self" href="/api/aufgaben/123" method="GET" />
    <link rel="update" href="/api/aufgaben/123" method="PUT" />
    <link rel="users" href="/api/aufgaben/123/users" method="GET" />
  </Aufgabe>
</Aufgaben>
```

RMM Level 3 – Beispiel Aufgabe JSON

```
{
  "AufgabeId":123,
  "links": [
    {
      "rel":"self",
      "href":"/api/aufgaben/123",
      "method":"GET"
    },
    {
      "rel":"update",
      "href":"/api/aufgaben/123",
      "method":"PUT"
    },
    {
      "rel":"users",
      "href":"/api/aufgaben/123/users",
      "method":"GET"
    }
  ]
}
```

ASP.NET Core 8 Api

Versionshistorie ASP.NET Core Web Api



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 28.06.2016 ASP.NET Core 1 | 14.08.2017 ASP.NET Core 2 | 31.05.2018 ASP.NET Core 2.1 | 04.12.2018 ASP.NET Core 2.2 | 23.09.2019 ASP.NET Core 3.0 | 03.12.2019 ASP.NET Core 3.1 | 10.11.2020 ASP.NET Core 5 | 09.11.2021 ASP.NET Core 6 | 08.11.2022 ASP.NET Core 7 | 14.11.2023 ASP.NET Core 8 |
|---|---|---|---|---|---|---|---|---|---|

- WCF Team arbeitete an der Unterstützung von REST.
 - WCF WebHTTP, WCF REST Starter Kit und WCF Web Api
- Parallel dazu hat das MVC-Team die Erstellung von Basis WebApi mit JSON als Rückgabe veröffentlicht
- WCF und ASP.NET Teams wurden zusammengelegt mit ASP.NET Web API als Ergebnis
- Im Zuge des neuen modularen .NET Core wurde **ASP.NET MVC + WebApi = ASP.NET Core**

Was ist ASP.NET Core WebApi?

- Leichtgewichtiges Microsoft-Web-Framework um moderne Internet-Anwendungen zu erstellen
- Ziel ist die Erstellung von HTTP-Diensten für .NET
- Erstellung von REST basierten Services mit HTTP und XML/JSON
- Komplette Neuentwicklung / Modularisiert
- Open Source
 - <https://github.com/dotnet/aspnetcore>
- ASP.NET Core 8 erfordert .NET Core 8

- Basiert auf HTTP somit keine Unterstützung von TCP, Named Pipes, UDP etc.
- Konvention über Konfiguration
- Unterstützung von Content Negotiation
- Plattformunabhängig
- Built-In- Dependency Injection
- Typische Einsatzszenarien
 - Backend für Single Page Applications, Mobile Clients, Desktop Anwendungen

Unterschiede SOAP (WCF) / REST (Web API)

| | SOAP (WCF) | REST (Web API) |
|----------------------------------|----------------------|-----------------------------|
| Schnittstellenbeschreibung | WSDL | Keine |
| Adressmodell | URI | URI |
| Schnittstelle | Anwendungsspezifisch | Generisch (GET,POST,PUT..) |
| Discovery | UDDI | Generische Schnittstellen |
| Status | Server / Client | Client |
| Transport | HTTP, SMTP, UDP... | HTTP |
| Standard | W3C | Ist Architekturstil |
| Formatting / Content Negotiation | Nein | Ja |
| Hypermedia | Nein | Ja |
| Performance | Hoher Overhead | Niedriger Overhead |



Projekt Quote of the Day

Action Results von IActionResult (Auszug)

| Name | Beschreibung |
|--|--|
| BadRequest() | Erstellt eine BadRequest Objekt mit Status Code 400 |
| BadRequest(message) | Erstellt eine BadRequestObjectResult Objekt mit Status Code 400 + Nachricht im Response Body |
| BadRequest(modelstate) | Erstellt ein BadRequestObjectResult mit Status Code 400 und Validationinfos im Response Body |
| Content(content,contentType) | Erstellt ein ContentResult mit beliebigem ContentType |
| Created(uri, object) | Erstellt einen CreatedResult mit Statuscode 201 und URL des neu erstellten Objekts |
| CreatedAtRoute(routeName, routeValues, value) | CreatedAtRouteResult mit 201 Statuscode mit URL via benannter Route und Daten |
| File(byte[],string) | Erstellt ein FileContentResult mit Binär-Datei als Antwort |
| NoContentResult | Erstellt eine NoContentResult mit Statuscode 204 |
| NotFound() | Erstellt ein NotFoundResult mit Statuscode 404 |

Action Results von IActionResult (Auszug)

| Name | Beschreibung |
|------------------------------------|---|
| Json(data) | Erstellt ein JsonResult mit übergebenen Daten als JSON |
| Ok() | Erstellt ein OkResult mit Status Code 200 |
| Ok(data) | Erstellt ein OkObjectResult mit Status Code 200 mit Rückgabedaten |
| Redirect(target) | Erstellt einen RedirectResult mit Statuscode 302 mit der URL für den Client |
| RedirectToRoute(name,props) | Erstellt ein RedirectToRouteResult, das eine URL von der Routingkonfiguration benutzt (302) |
| StatusCode(code) | Erstellt ein StatusCodeResult, dass den spezifischen Statuscode benutzt |
| Unauthorized() | Erstellt ein UnauthorizedResult mit StatusCode 401 |

Eigene Action Results

- Eigene Klassen müssen von **IActionResult** oder **ActionResult** erben und **ExecuteResultAsync** bzw. **ExecuteResult**(synchron) implementieren
- Alternative erben von vorhandenen ActionResults

```
public class NoContentResult : IActionResult
{
    public Task ExecuteResultAsync(ActionContext context)
    {
        return Task.FromResult(context.HttpContext.Response.StatusCode =
                                StatusCodes.Status204NoContent);
    }
}
```

HTTP Status Codes (Auszug)

■ 2xx Erfolgreiche Verarbeitung

| Code | Bedeutung | Erläuterung |
|------------|-------------------|--|
| 200 | OK | Die Anfrage wurde erfolgreich verarbeitet, die Antwort enthält weitere Informationen |
| 201 | Created | Die Anfrage wurde erfolgreich verarbeitet und als Ergebnis wurde eine neue Ressource angelegt, deren URI sich in einem Location-Header befindet. |
| 204 | No Content | Der Server liefert nur Metadaten (in Form von Header-Informationen), keine Daten. |

HTTP Status Codes (Auszug)

■ 3xx Umleitung

| Code | Bedeutung | Erläuterung |
|------|-------------------|---|
| 301 | Moved Permanently | Die Ressource ist unter einer neuen URI erreichbar, die im Location-Header benannt wird. Clients sind aufgefordert, evtl. bestehende Bookmarks (oder allgemeiner: gespeicherte Referenzen) zu aktualisieren. |
| 302 | Found | Die Ressource hat aktuell eine andere URI (im Location-Header aufgelistet), ein Client soll jedoch weiterhin die ursprüngliche URI verwenden. Browser interpretieren einen 302-Statuscode leider nicht konsistent: Einige wiederholen den Request mit der gleichen Methode, andere senden ein HTTP GET. |
| 304 | Not Modified | Als Antwort auf ein Conditional GET (mit einem If-None-Match-oder If-Modified-Since-Header) signalisiert dieser Statuscode, dass sich die Ressource nicht geändert hat. |

HTTP Status Codes (Auszug)

■ 4xx Clientfehler

| Code | Bedeutung | Erläuterung |
|------|------------------------|--|
| 400 | Bad Request | Die Anfrage ist vom Server nicht verarbeitbar |
| 401 | Unauthorized | Ohne Authentifizierungsinformationen verarbeitet der Server den Request nicht |
| 403 | Forbidden | Der Server hat den Request zwar interpretieren können, verweigert jedoch die Ausführung. Auch Authentifizierung ändert daran nichts. |
| 404 | Not Found | Der Server kennt keine Ressource mit dieser URI. |
| 405 | Method Not Allowed | Die HTTP-Methode wird von der Ressource nicht unterstützt, der Allow-Header enthält die Liste der Methoden, die verwendet werden dürfen |
| 406 | Not Acceptable | Der Server kann keine Repräsentation zurückliefern, deren Format einem der vom Client im Accept-Header aufgelisteten Medientypen entspricht. |
| 415 | Unsupported Media Type | Gegenstück zu 406: Das Format, in dem der Client den Inhalt sendet, kann vom Server nicht akzeptiert werden. |

HTTP Status Codes (Auszug)

■ 5xx Serverfehler

| Code | Bedeutung | Erläuterung |
|------|-----------------------|--|
| 500 | Internal Server Error | Ein nicht näher spezifizierter interner Fehler ist bei der Verarbeitung im Server aufgetreten. Dies ist die »weichste« aller Fehlermeldungen – sie erlaubt dem Client keinerlei Rückschluss auf die Art des Fehlers. |
| 501 | Not Implemented | Die HTTP-Methode, die der Client verwendet, wird von keiner Ressource des Servers unterstützt. |
| 502 | Bad Gateway | Ein Gateway-Server (wie zum Beispiel ein Reverse Proxy Cache oder ein Load Balancer) hat von einem nachgelagerten Server eine ungültige Antwort erhalten. |
| 503 | Service Unavailable | Der Server ist aktuell nicht in der Lage, die Anfrage zu beantworten. In einem Retry-After-Header kann er dem Client mitteilen, nach welchem Zeitraum sich ein erneuter Versuch lohnt. |
| 504 | Gateway Timeout | Ein nachgelagerter Server hat dem Gateway nicht rechtzeitig eine Antwort geliefert. |

HttpClient

- Die Klasse `HttpClient` bietet die Möglichkeit der Konsumierung von Web-API-Services für .NET-basierte Clients
- Asynchrones Programmiermodell

HttpClient - Beispiel

```
static void Main(string[] args)
{
    var url = "https://localhost/api/sprueche/spruchdestages";
    var client = new HttpClient();
    var response = await client.GetAsync(url);

    response.EnsureSuccessStatusCode();

    var content = await response.Content.ReadAsStringAsync();
    var spruch = JsonSerializer.Deserialize<Sdt>(content);

    System.Console.WriteLine($"Spruch Rückgabe (Auszug) ->
    Name: {spruch.AutorName} ## Beschreibung:
    {spruch.AutorBeschreibung} ## Spruch: {spruch.SpruchText}");

    Console.ReadLine()
}
```


DEMO

„Http-Client“

Media Type Formatter

Media Type Formatter

- Primär unterstützt ASP.NET Core JSON
- Möglichkeit eigene Formate bereitzustellen
- Content-Negotiation entscheidet über Serialisierer
 - Http-Accept-Header bestimmt das Rückgabeformat (wenn möglich!)
 - **RespectBrowserAcceptHeader** in ApiOption muss **true** sein
 - Rückgabe-Format kann erzwungen werden mit **[Produces(contentType)]**-Attribut
- Input/Output-Formatter können separat hinzugefügt bzw. entfernt werden

| Name | Beschreibung |
|---|------------------------------------|
| JsonFormatter | Json (nutzt JSON.NET) |
| XmlSerializerFormatter/ XmlDataContractSerializerFormatter | Xml (nutzt DataContractSerializer) |

- Implementierung von OutputFormatter
- Registrierung des benutzerdefinierten Formatters in den API-Optionen ConfigureServices

```
services.AddControllers(options => {  
    options.OutputFormatters.Add(new CsvQuoteOfTheDayFormatter());  
});
```

DEMO

„CSV Formatter für SpruchDesTages“

CORS

Cross Origin Resource Sharing

- Same-Origin-Policy verhindert, dass eine Browseranwendung via Javascript auf fremde Websites zugreift
- Kein Zugriff wenn Protokoll, Domäne oder Port abweichen
- CORS erlaubt den Zugriff auf den Service
- Vorgehensweise
 - NuGet-Paket Microsoft.AspNetCore.Cors
 - `services.AddCors()`
 - `app.UseCors();`
 - Annotierung der gewünschten Methoden/Controller mit `[EnableCors]` oder `Global`

DEMO

CORS

- Möglichkeit via API-Key Zugriff zu gewähren
- Client übermittelt „x-api-key“ im Header eines Requests, welche auf Serverseite überprüft wird
- Via Filter bzw. Middleware möglich

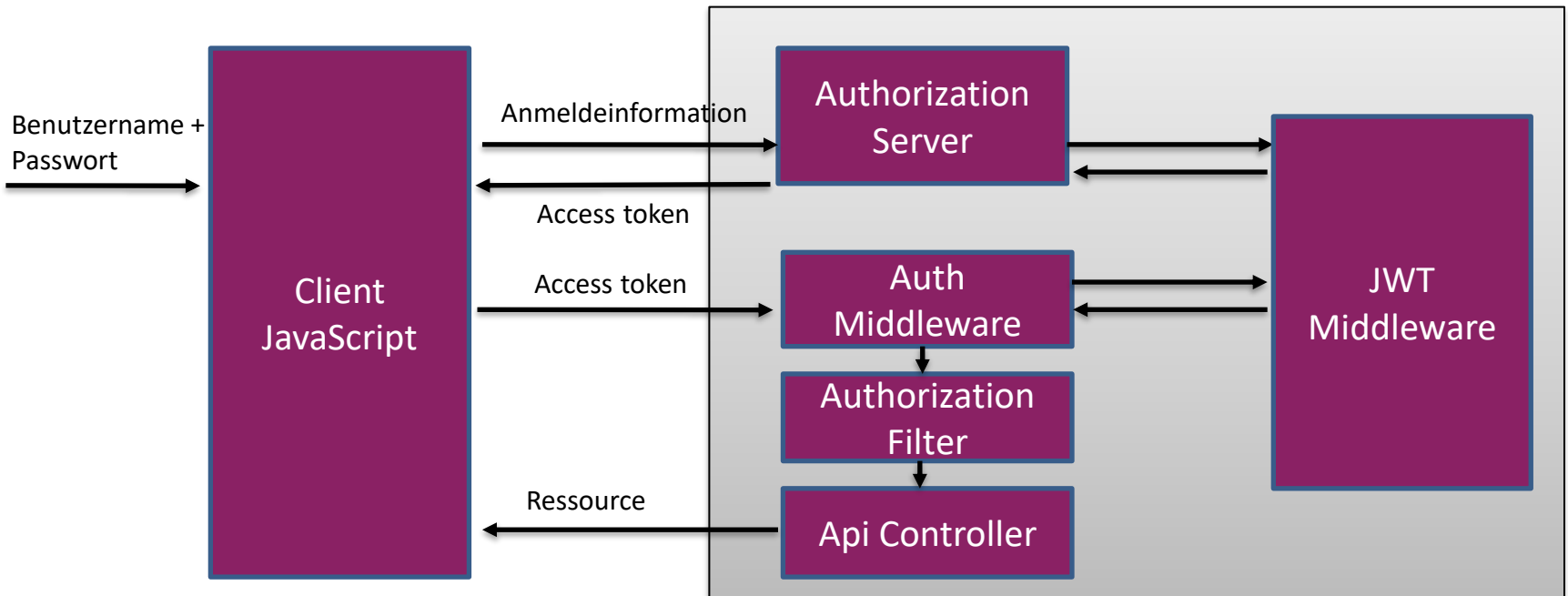
JWT - Bearer-Authentifizierung

- Definition
 - *A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession) (Quelle: <https://tools.ietf.org/html/rfc6750#section-1.2>)*
- OAuth 2.0 mit dem Ziel Benutzern die Möglichkeit zu geben, einen Teil ihrer Rechte an Dritte weiterzugeben, ohne das eigene Passwort mitzuteilen
- Begrifflichkeiten
 - **Resource** = jegliche schützbares Daten
 - **Resource server** = Server, der die Resource hostet
 - **Resource owner** = Die Entität, die den Zugriff auf die Resource erlauben kann (User)
 - **Client** = Die Anwendung, die auf die Resource zugreifen möchte (Webbrowser)
 - **Access token** = Ein Token, das Zugriff auf die Resource gewährt
 - **Bearer token** = Spezieller access token den jeder benutzen kann
 - **Authorization server** = Server, der access tokens vergibt
- ASP.NET Core durch `JwBearerAuthenticationMiddleware`
 - HTTP Header Authorization : Bearer BEARER_TOKEN

- JSON Web Token (JWT) ist ein Webstandard, der eine Methode zum Übertragen von Claims als JSON-Objekt so definiert, dass sie kryptografisch signiert oder verschlüsselt werden können
- JWT besteht aus 3 Teilen:
 1. **Header** = Ein JSON-Objekt, das den Typ des Tokens (JWT) und den Algorithmus anzeigt, der für die Signierung verwendet wird
 2. **Payload** = Ein JSON-Objekt mit den geltend gemachten Claims der Entität
 3. **Signature** = Eine Zeichenfolge, die mit einem Geheimnis und dem kombinierten Header und Payload erstellt wird. Wird verwendet, um zu überprüfen, ob das Token nicht manipuliert wurde.
- Infos auf <https://jwt.io/>

OAuth2 Individual Account Web API - Ablauf

1. Benutzer trägt Benutzername und Passwort im Client ein
2. Client sendet die Anmeldeinformationen (Credentials) an den Authorization Server
3. Authorization Server authentifiziert den Benutzer und gibt ein Access Token zurück
4. Client setzt im Authorizations-Header der HTTP-Anfrage den Access Token um auf die geschützte Ressource zuzugreifen (nach Filtern)



- Authorize
 - Kontrolliert Wer den Controller/Action zugreifen darf via Eigenschaften Users / Roles / Claims

ASP.NET Blazor

Was ist Blazor?

ASP.NET Core Blazor ist Microsofts Webentwicklungsframework zur Entwicklung von interaktiven Single-Page-Applications (SPA) mit C# und HTML

Zurzeit drei Hauptarten von Blazor

- *Blazor WebAssembly* (aka Client-Side)
 - *Blazor Server* (aka Server-Side)
- Blazor Static Server-side-Rendering (aka SSR seit .NET 8 als MPA)

- Entwickeln mit C# statt JavaScript
- Nutzung des .NET-Ökosystems von .NET Bibliotheken
- Seit Version 6 Native Dependencies

WebAssembly (wasm) ist ein offener Standard, der vom W3C festgelegt wurde. Er definiert einen Bytecode zur Ausführung von Programmen innerhalb von Webbrowsern, kann aber auch außerhalb von diesen genutzt werden. Ziel der Entwicklung war es, leistungsstärkere Webanwendungen als bisher zu ermöglichen, sowohl was die Ladezeiten als auch die Ausführung betrifft

Quelle: <https://de.wikipedia.org/wiki/WebAssembly>

Webseite: <https://webassembly.org/>

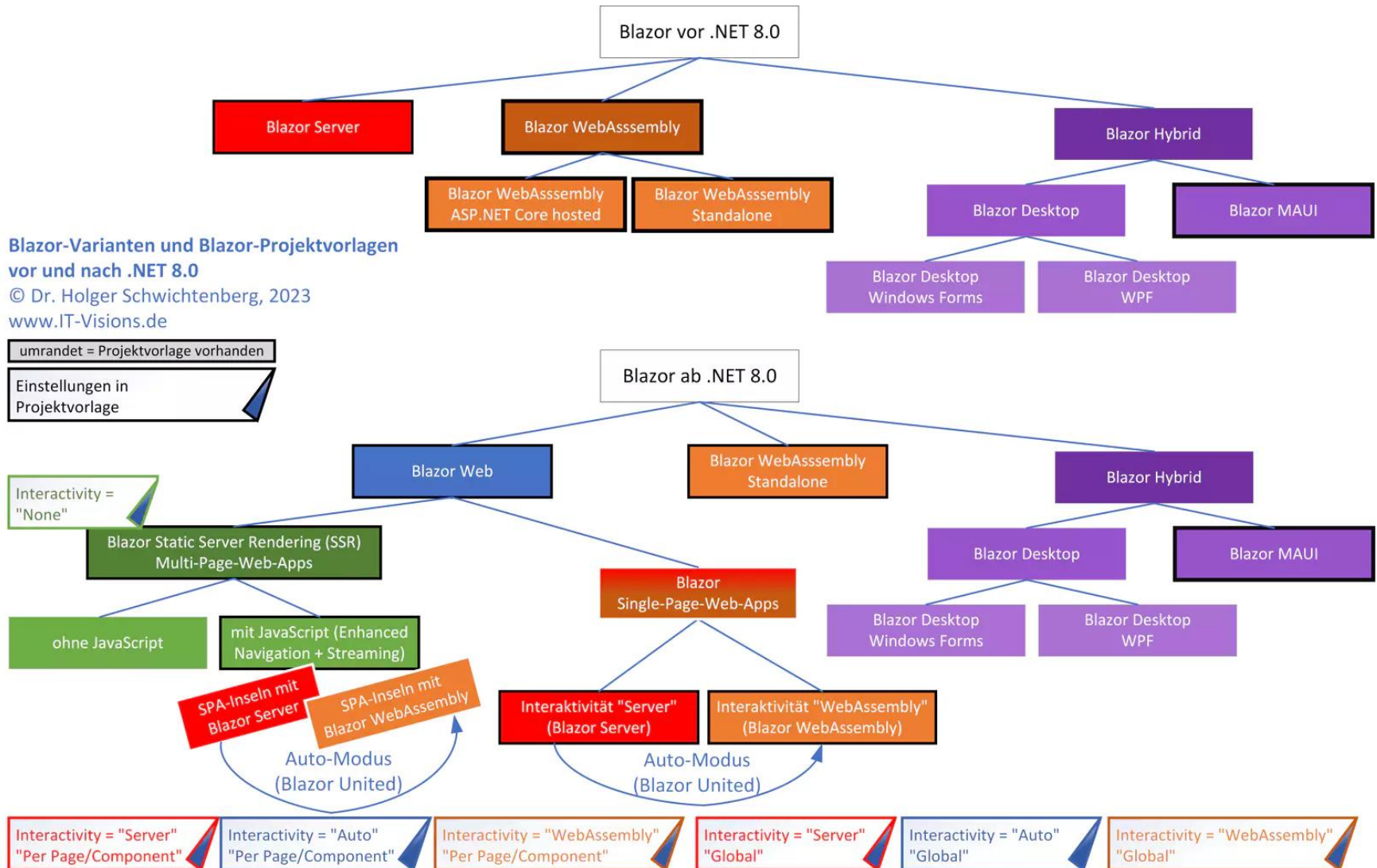
Überblick ASP.NET Core Blazor 8

Blazor-Varianten und Blazor-Projektvorlagen vor und nach .NET 8.0

© Dr. Holger Schwichtenberg, 2023

www.IT-Visions.de

umrandet = Projektvorlage vorhanden
Einstellungen in Projektvorlage

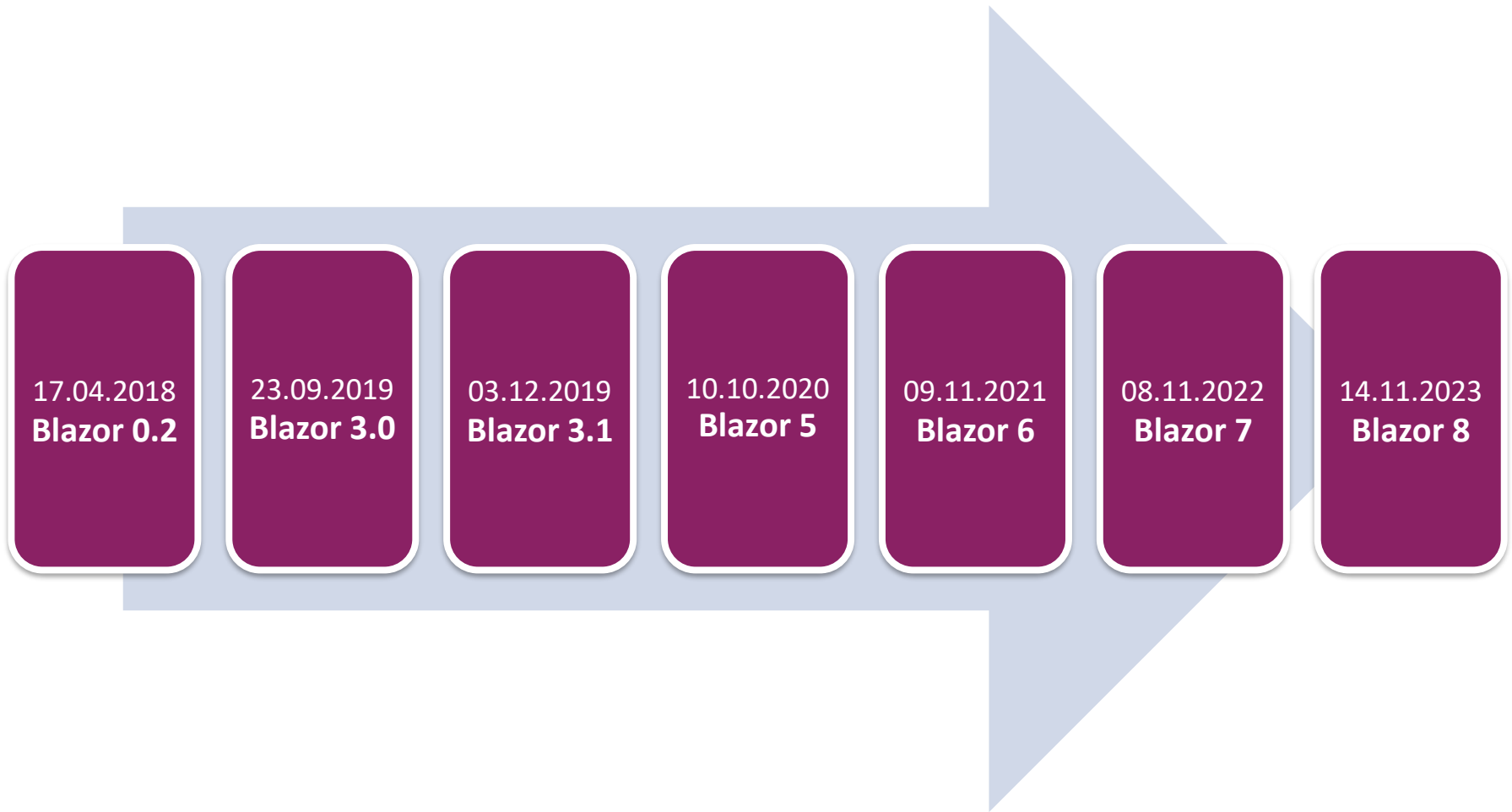


NEU in ASP.NET Core Blazor 8

- Unterstützung C# 12
- Auto-Rendering-Modus
- Static-Server-Rendering
- Full-Stack WebUI
- Hot Reload Verbesserungen
- Unterstützung Fluent UI
- Keyed Services
- Sektionen
- QuickGrid
- AOT
- Blazor Identity UI
- HTTP/3

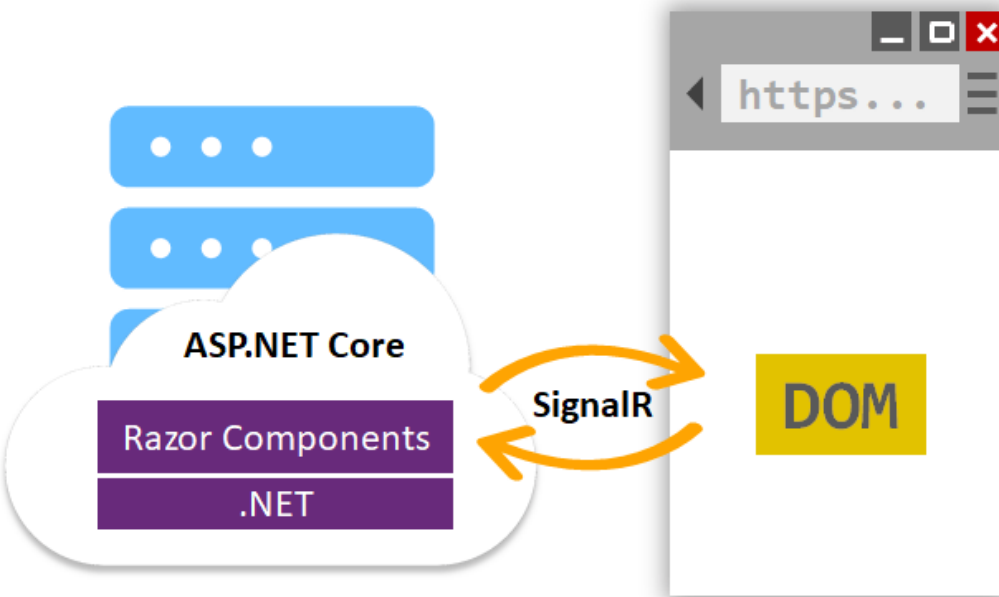
<https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0?view=aspnetcore-8.0#blazor>

Versionshistorie Blazor

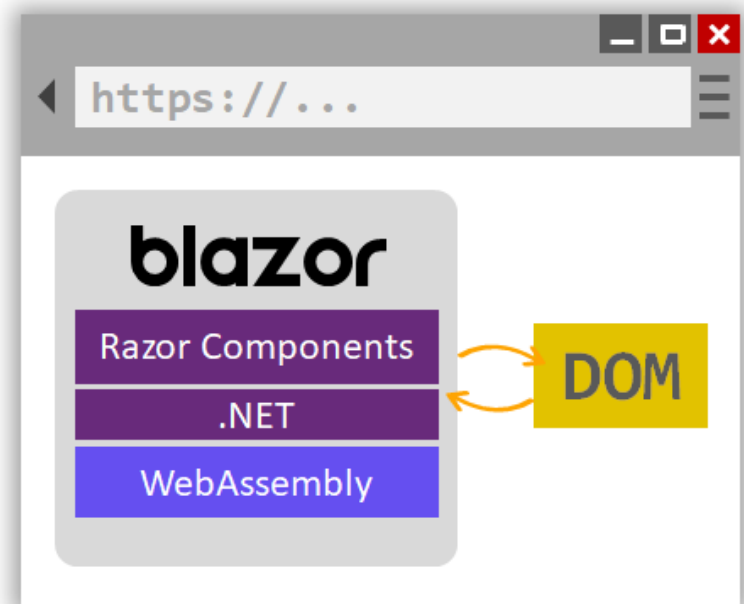


Blazor Server vs. Blazor WebAssembly

Blazor Server

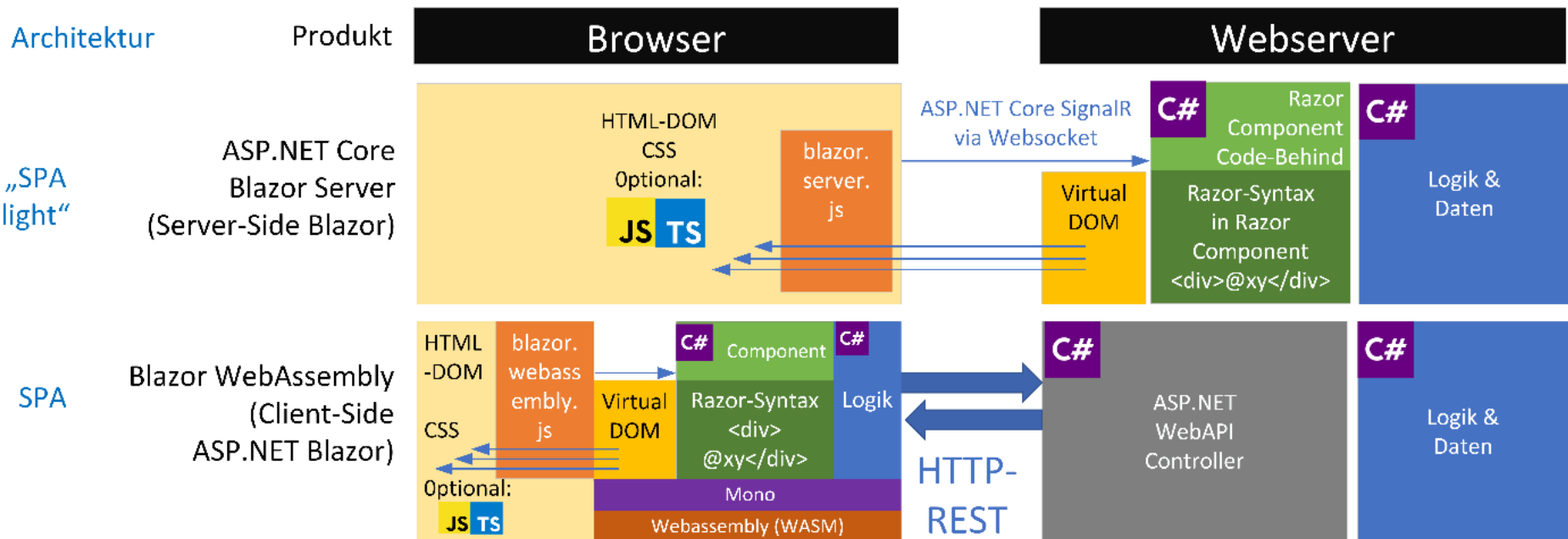


Blazor WebAssembly



ASP.NET Core Blazor: Architekturalternativen

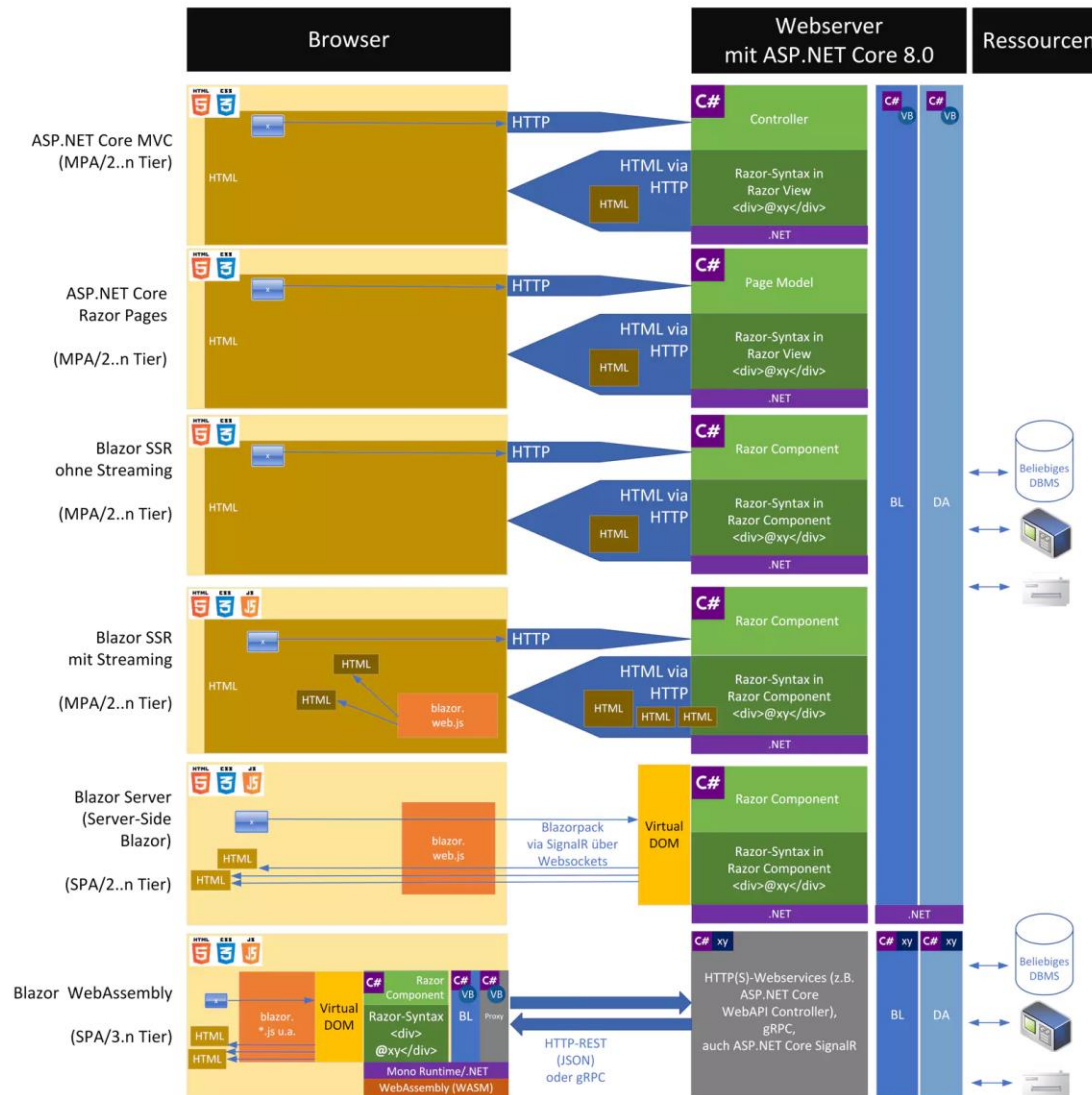
© Dr. Holger Schwichtenberg, www.IT-Visions.de 2018-2020



Architektur .NET 8

ASP.NET Core-/Blazor-Browseranwendungen Anwendungsarchitekturen im Vergleich

© Dr. Holger Schwichtenberg, www.IT-Visions.de 2023



Unterschiede Blazor Server & Blazor WASM

| Merkmal | Blazor WebAssembly | Blazor Server / SSR |
|---------------------------------|---|---------------------|
| Entwickeln mit C# | ✓ | ✓ |
| Kleine Downloadgröße | .NET Runtime & Abhängigkeiten müssen heruntergeladen werden | ✓ |
| Hohe Ausführungsgeschwindigkeit | ✓ | Latenz |
| Serverless | ✓ | Server nötig |
| Browserunabhängig | Neuere Browser Kein IE | ✓ |
| Static File Deployment CDN | ✓ | Server nötig |
| Offline-Fähig | ✓ | Nein |

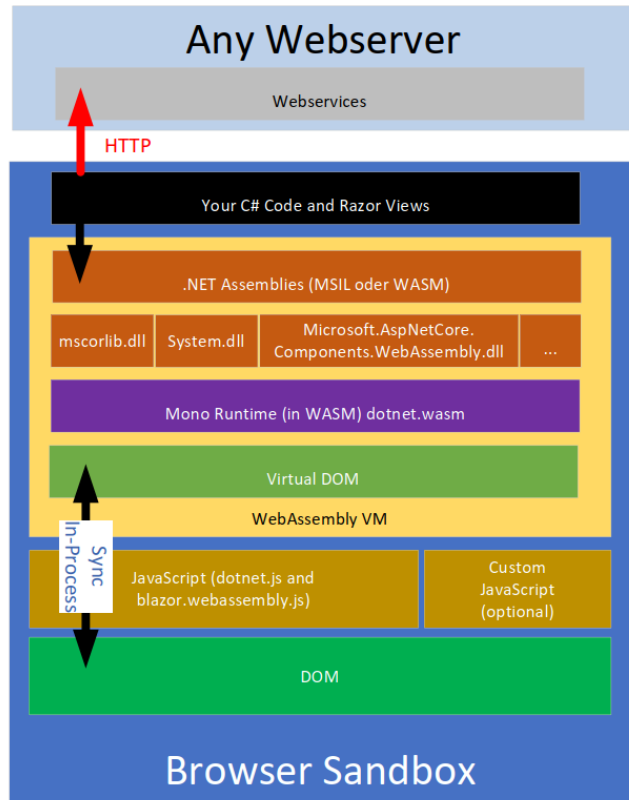
Unterschiede Blazor Server & Blazor WASM Fort.

| Merkmal | Blazor WebAssembly | Blazor Server / SSR |
|-------------------------------------|---------------------|-----------------------|
| Kapselung der Geschäftslogik nötig | ✓ | Nein aber möglich |
| Restriktionen durch Browser-Sandbox | ✓ | Zugriff auf .NET APIs |
| Netzwerkprotokolle | Http, Https, gRPC | alle |
| Gute Skalierbarkeit | ✓ | |
| Multi-Threading | Nur UI-Thread | ✓ |
| Schutz des eigenen Codes | In Client einsehbar | ✓ |

Browser Sandbox

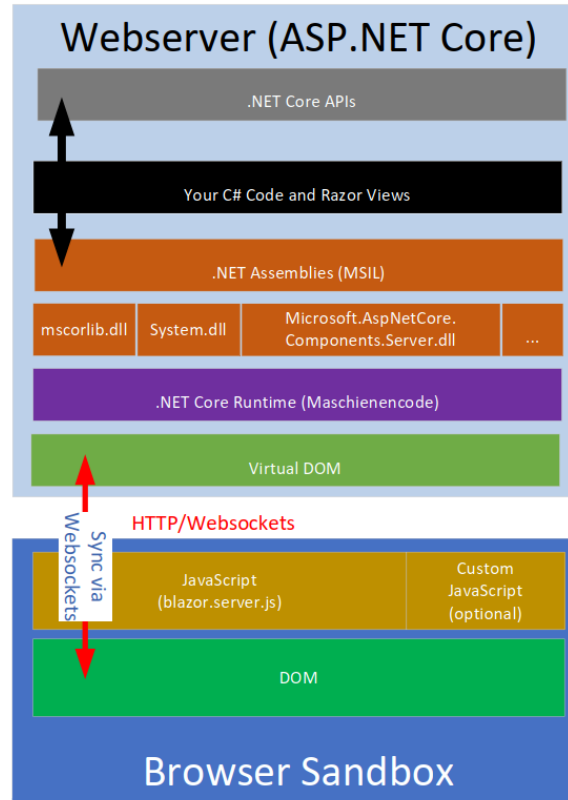
Blazor WebAssembly

Author: Dr. Holger Schwichtenberg, www.IT-Visions.de



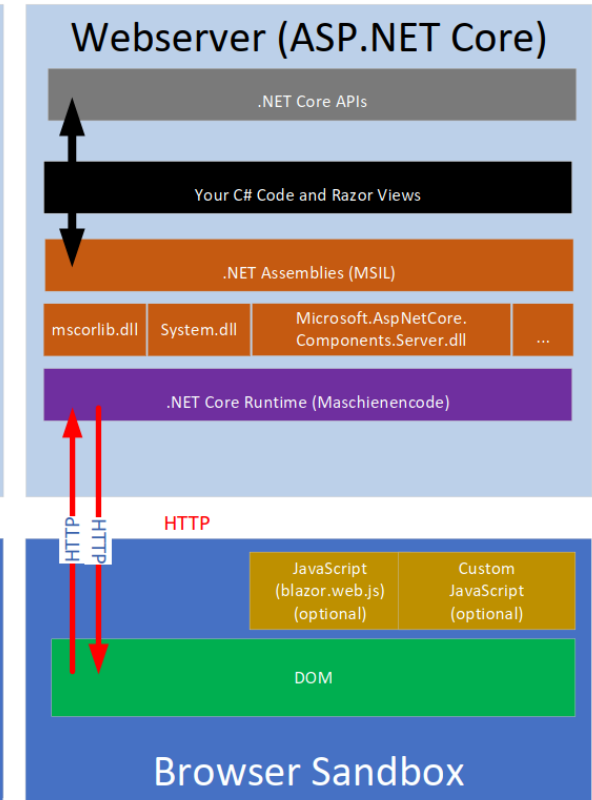
Blazor Server

Author: Dr. Holger Schwichtenberg, www.IT-Visions.de



Blazor SSR

Author: Dr. Holger Schwichtenberg, www.IT-Visions.de



- **Program**
 - Verantwortlich für die Konfigurierung/Ausführen der Anwendung
 - Einstiegspunkt der Webapplikation
- **WebApplicationBuilder** verantwortlich für Configuration und Service Registration
- **IServiceCollection** wird genutzt um Services dem Container hinzuzufügen und zu konfigurieren
- **Middlewares** um Pipeline zu konfigurieren
- **App.razor** ist Startkomponente der Anwendung, die auf die **Routes.razor** verweist, die wiederum die **MainLayout.razor** lädt
- `<script src="_framework/blazor.web.js"></script>` in App.razor stellt SignalR-Verbindung zum WebServer her

Razor-Komponenten

Razor

Was ist eine Razor-Komponente?

- Zentrale UI-Elemente einer Webanwendung
- Templatesprache HTML, CSS und einer speziellen CodeSyntax bzw. View-Engine namens RAZOR @ sowie C#-Code
- Dateien haben die Endung „razor“ und erben implizit von ComponentBase
- Name muss mit Großbuchstaben starten
- Zwei Möglichkeiten Code zu erstellen
 - Mixed Ansatz mit @code
 - Partielle C# Klasse
- Können mit der @page Direktive direkt aufgerufen werden
- CSS und JavaScript-Isolation möglich
- Einbinden als Tag => <NameKomponente />

Komponenten-Beispiel

```
@page "/counter"
<p>Current count: @currentCount</p>
<button @onclick="IncrementCount">Click me</button>

@code {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

Adresse

Variable

Click-Ereignis

Mixed Ansatz
Counter.razor

```
public partial class Counter
{
    ...
}
```

C# Klasse
Counter.razor.cs

Parameter

Parameter, Arbitrary, Cascading

Parameter

- Parameter können von Eltern-Komponente übergeben werden via Attribute
- Parameter **muss** *Property* und *public* sein
- Zusätzlich als Pflichtparameter [**EditorRequired**] dem Child-Parameter hinzugefügt werden, damit Entwickler den Parameter setzt

Index.razor

```
@page "/"

<Home Title="@title"></Home>

@code {
    private string title = "Homepage";
}
```

Home.razor

```
<h1>@Title</h1>

@code {

    [Parameter, EditorRequired]
    public string Title { get; set; }
}
```


Arbitrary Parameter

- Zusätzliche Parameter können gesammelt übergeben werden

```
@page "/"
```

```
<Home Title="@title" alt="Bild des Autors" width="150"></Home>
```

```
@code {  
    private string title = "Homepage";  
}
```

```
<h1>@Title</h1>  

```

```
@code {  
    [Parameter] public string Title { get; set; }  
  
    [Parameter(CaptureUnmatchedValues = true)]  
    public Dictionary<string, object> AdditionalAttributes { get; set; }  
}
```

Cascading Parameters

- Parameter werden von Eltern- zu allen Kindkomponenten übergeben

```
@page "/"
<CascadingValue Value="@_color">
    <Home></Home>
</CascadingValue>
```

```
@code {
    private string _color = "red";
}
```

```
<h1 style=„color: @Color">@Title</h1>
```

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

    [CascadingParameter] //optional mit Name
    public string Color { get; set; } // Setter vorhanden & public & Type muss gleich sein
}
```

Render Fragment Parameters

- Inhalt an Kind-Komponenten senden

```
@page "/"
```

```
<Home>
```

Dieser Inhalt wird in der Kind-Komponente angezeigt

```
<AlternativContent>
```

Alternativer Inhalt wird angezeigt

```
</AlternativContent>
```

```
</Home>
```

```
<div>@ChildContent @AlternativContent</div>
```

```
@code {
```

```
    [Parameter]
```

```
    public RenderFragment ChildContent { get; set; } //ChildContent od NamedTag in Eltern-Komp
```

```
    [Parameter]
```

```
    public RenderFragment AlternativContent { get; set; } //alt. Named Tag in Eltern-K
```

```
}
```



Parameter

Daten- und Event-Binding

- Anzeigen und die Verarbeitung von Daten bzw. Ereignissen als zentrale Funktionalität
- One-Way-Datenbindung
- Two-Way-Datenbindung
- Ereignisbehandlung

One / Two-Way-Bindung

- One-Way ist Bindung einer C#-Variablen an ein DOM-Element. Bei Änderung des Variablenwerts spiegelt es sich im UI wieder
- Two-Way ist bidirektional d.h. wenn Wert in UI-Element ändert, aktualisiert sich die C#-Variable und umgekehrt. Standardmäßig erfolgt die Synchronisierung im **onchange**-Event.

```
<div>@title</div> // One-Way
<input @bind="_name" /> // Two-Way mit C# Feld
<input @bind="Name" /> // Two-Way mit C# Property Kurzversion
<input @bind-value="Name" @bind-value:event="onchange" /> //Two-Way mit C# Langversion
```

```
// Two-Way XXL - Langversion
```

```
<input value="@Name" @onchange="@((ChangeEventArgs e) => Name = e.Value.ToString()) "
```

```
// Two-Way-Binding mit oninput-Event
```

```
<input @bind-value="Name" @bind-value:event="oninput" />
```

```
@code {
    private string title = "Mein Titel";
    private string _name;
    private string Name { get; set; }
}
```

Ereignisbehandlung

Die Behandlung von Html-DOM-Ereignissen erfolgt über das Muster **@on[Event]**

```
<button @onclick="ErsteMethode">Klick mich</button>  
<button @onclick="() => ZweiteMethode(item)">Klick mich 2</button> //mit Parameter als Lambda  
<button @onclick="() => { item++ }">Klick mich 3</button> //Code direkt Abkürzung  
  
<input @onkeypress="KeyPress" /> //Standardereignis mit Argument  
<input @onkeypress="(e) => KeyPress(e)" /> //Explizit mit Argument zwingend bei zusätzl. Parameter  
  
<a @onclick="OnClickHandler" @onclick:preventDefault>Klick</a> //Standardereignis unterbinden  
<p @onclick="OnClickHandler" @onclick:stopPropagation></p> //Ereignisweitergabe unterbinden
```

```
@code {  
    private void ErsteMethode() {}  
    private void ZweiteMethode(string item) {}  
    public void KeyPress(KeyboardEventArgs args) {}  
}
```


Ereignisweitergabe

Komponenten können Ereignisse auslösen und z.B. Elternkomponenten informieren

```
<button @onclick="() => EineMethode(item)">Klick mich</button>
```

```
@code {  
    [Parameter]  
    public EventCallback<int> OnEventCallback { get; set; } //Rückgabe Typ integer  
  
    public void EineMethode(int item)  
    {  
        OnEventCallback.InvokeAsync(item); //Auslösen des Ereignisses  
    }  
}
```

```
<KindKomponente OnEventCallback="MeineMethode"></KindKomponente>
```

```
@code {  
    public void MeineMethode(int item) {}  
}
```



Data-Binding

Lebenszyklus

Lebenszyklus

| Methode | Beschreibung |
|---|--|
| OnInitialized & OnInitializedAsync | Diese Methode wird ausgelöst, sobald die Komponente initialisiert wird und alle Parameter von der übergeordneten Komponente erhält. Achtung: Blazor Server ruft die Methode bei ServerPrerendered die Startkomponenten zweimal auf. |
| OnParametersSet & OnParametersSetAsync | Diese Methode wird ausgelöst, nachdem die Komponente ihre Parameter erhalten hat und ihre Werte ihren jeweiligen Eigenschaften zugewiesen wurden. Diese Methode wird <i>wiederholt</i> ausgelöst, sobald der Wert des Parameters aktualisiert wird |
| OnAfterRender & OnAfterRenderAsync | Sobald die Komponente das Rendern beendet hat, wird diese Methode aufgerufen. Es wird in dieser Phase verwendet, um zusätzliche Initialisierungsschritte unter Verwendung des gerenderten Inhalts durchzuführen. Bei dieser Methode ist der firstRender-Parameter nur bei der <i>ersten Ausführung</i> der Methode auf "true" gesetzt. Jedes andere Mal ist der Wert „false“ |

Lebenszyklus II

| Methode | Beschreibung |
|---------------------------|---|
| ShouldRender | Diese Methode kann verwendet werden, um die Aktualisierung der Benutzeroberfläche zu unterdrücken. Diese Methode wird jedes Mal aufgerufen, wenn die Komponente gerendert wird. Unabhängig davon, ob wir diese Methode so einstellen, dass sie "false" zurückgibt, wird die Komponente immer anfänglich gerendert |
| SetParametersAsync | Diese Methode wird zuerst ausgelöst, sobald die Komponente erstellt wird. Alle an die Komponente gesendeten Parameter werden im ParameterView-Objekt gespeichert. Falls einige asynchrone Aktionen ausgeführt werden sollen, bevor ein Parameterwert zugewiesen wird |
| StateHasChanged | Diese Methode wird immer dann aufgerufen, wenn Blazor benachrichtigt werden soll, dass sich etwas in der Komponente geändert hat, und diese Komponente neu gerendert werden soll. |



Lebenszyklus

Routing & Navigation

- **App.razor** definiert die Router-Komponente
- Assembly wird gescannt um Routeninformationen für die Komponenten der Anwendung zu erfassen
- **Routes**-Komponente empfängt Route-Data-Objekt mit Routenparameter
- Rendert die angegebene Komponente mit deren Layout
- **FocusOnNavigate** setzt den UI-Fokus auf eine Element
- Falls zur angeforderten Route keine Komponente gefunden wird **<NotFound>** dargestellt
- Durch **AdditionalAssemblies** können zusätzliche Assemblies bei der Suche nach Komponenten für das Routing eingebunden werden
- **BaseUri** wird in App.razor gesetzt

@Page Direktive

- **@page** definiert eine Routing Regel (oder mehrere) damit Blazor die richtige Komponente aufruft
- Case-Insensitiv
- Müssen eindeutig sein für Komponenten
- Ohne @page Direktive kann Komponente nicht über URL aufgerufen werden, sondern kann nur in andere Komponenten eingebettet sein
- Können Parameter beinhalten mit optionalen Parameter-Typ-Einschränkung

```
@page "/"
```

```
@page "/home"
```

```
@page "/kunde/autrag"
```

```
@page "/autor/details/{name}" //Routen-Parameter => Parameter-Typ string ist Standard
```

```
@page "/autor/auftrag/{name?}" //Routen-Parameter => Parameter-Typ optional
```

```
@page "/autor/details/{AutorId:guid}" //Routen-Parameter mit Parameter-Typ-Einschränkung
```

Routing Einschränkungen

| Einschränkung | Beschreibung | Beispiel |
|---------------|--------------------------|-----------------|
| bool | Boolean (true, false) | {active:bool} |
| datetime | DateTime | {dob:datetime} |
| decimal | Matches a decimal value. | {price:decimal} |
| double | Double | {weight:double} |
| float | Float | {weight:float} |
| guid | GUID value. | {id:guid} |
| int | 32-bit integer. | {id:int} |
| long | Long | {ticks:long} |

Navigation

- Per Html-Hyperlink kann zu einer Komponente navigiert werden
- Zusätzlich ist eine Navigation auch im Code möglich mit dem **NavigationManager**

//Html Variante

```
<a href="/autors">Liste der Autoren</a>
```

//Code Variante

```
public partial class Home
{
    [Inject]
    public NavigationManager NavManager { get; set; }

    public void Methode()
    { NavManager.NavigateTo("/kunde"); }
}
```

Dependency Injection

IoC

- Builder.Services in Program verantwortlich für die Bereitstellung der Services

| Scope | Bedeutung | Erläuterung |
|-----------|-----------------|---|
| Transient | vorübergehend | Bei jeder Verwendung (Anforderung einer Klasse beim Dependency Injection-Container) erhält man eine eigene Instanz |
| Scoped | bereichsbezogen | Blazor Server Der Bereich (Scope) bezieht sich bei Blazor Server auf eine Websocketsverbindung, also die Verbindung eines Browserfensters mit der Blazor Server-Anwendung ("Circuit"). Dies entspricht in der Regel einer Benutzersitzung. Blazor WebAssembly Dies entspricht in der Regel einer Benutzersitzung (Browser/Tab). Innerhalb einer Benutzersitzung erhält man bei einem Scoped Service bei jeder Verwendung dieselbe Instanz. |

DI Scopes

| Scope | Bedeutung | Erläuterung |
|-----------|--------------|--|
| Singleton | übergreifend | <p>Blazor Server</p> <p>Es gibt nur eine einzige globale Instanz für alle Verwendungen übergreifend innerhalb eines Prozesses. Singleton bedeutet bei Blazor Server, dass alle Benutzer die dieselbe Dienst-Instanz erhalten, da es ja nur einen Prozess für alle Benutzer gibt.</p> <p>Blazor WebAssembly</p> <p>Bei Blazor WebAssembly bedeutet Singleton, dass jeder Benutzer in jeder Instanz der Webanwendung (d.h. pro Browser-Tab) eine eigene Instanz des Services erhält. Da es bei Blazor WebAssembly nur eine Benutzersitzung in einem Prozess gibt, verhält sich die Lebensdauer "Scoped" bei Blazor WebAssembly genau wie "Singleton"</p> |

DI in Komponenten

- In Razor-Template mit **@inject** oder Property-Injektion **[Inject]**
- Im Code-behind mit Property-Injektion **[Inject]**
- In anderen Klassen mit Konstruktorinjektion
- Custom Dienste müssen in der ServiceCollection registriert werden mit einem von drei Lebensdauerkonzepten (Scopes)

@inject NavigationManager NavManager *//Razor Template Variante*

```
public partial class Home
{
    [Inject] //Code Behind Variante
    public NavigationManager NavManager { get; set; }
}
```

//Service Collection in Program.cs

```
builder.Services.AddScoped<ISchnittstelle, Schnittstelleimplementierung>();
```

Dependency Constructor-Injection

```
public class AuthorService
{
    private readonly IAuthorRepository _authorRepository;

    protected AuthorService(IAuthorRepository authorRepository)
    {
        _authorRepository = authorRepository;
    }
}
```

Interface, *keine* konkrete Implementierung

Konstruktor Injection



DI

WebServices/API via Http

HttpClient

- System.Net.Http.HttpClient in Projektvorlage vorhanden
- In Razor via [Inject] bzw. Konstruktorinjektion

//Service Collection in Program.cs

```
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new  
Uri(builder.HostEnvironment.BaseAddress) });
```

//Razor Komponente

@inject HttpClient Client

//Codebehind bei Razor Komponente

[Inject] HttpClient Client {get; set;}

//Konstruktor Injektion bei regulären Klassen

```
public class Service {  
    private HttpClient _client;  
    Service(HttpClient client)  
    { _client = client; }  
}
```

- Möglichkeit den HttpClient durch `IHttpClientFactory` zu erstellen (Empfehlung)

```
builder.Services.AddHttpClient("MeineAPI", opt =>
{
    opt.BaseAddress = new Uri("https://localhost:5011/api/");
});
```

```
//----- Komponente -----
```

```
[Inject] private IHttpClientFactory HttpClientFactory { get; set; }
```

```
protected override async Task OnInitializedAsync()
{
    var client = HttpClientFactory.CreateClient("MeineAPI");
}
```

- Weitere Möglichkeit den Http-Client als TypedClient zu nutzen

```
builder.Services.AddHttpClient<IMeinService,MeinService>(opt =>
{
    opt.BaseAddress = new Uri("https://localhost:5011/api/");
});
```

```
//----- MeinService.cs -----
public class MeinService
{
    private readonly HttpClient _client;

    MeinService(HttpClient client)
    {
        _client = client;
    }
}
```

Http-Client Methoden (Auszug)

| Methode | Beschreibung |
|--|---|
| GetAsync (string requestUri) | Sendet einen GET-Request an die Uri |
| GetStringAsync (string requestUri) | Sendet eine GET-Request an den angegebenen Uri und gibt den Antworttext als Zeichenfolge zurück |
| PatchAsync (string requestUri, HttpContent content) | Sendet einen PATCH-Request an die Uri mit Content |
| PostAsync (string requestUri, HttpContent content) | Sendet einen POST-Request an die Uri mit Content |
| PutAsync (string requestUri, HttpContent content) | Sendet einen POST-Request an die Uri mit Content |

| Erweiterungsmethode | Beschreibung |
|--|--|
| GetFromJsonAsync <T>(string requestUri) | Sendet einen GET-Request an die Uri und deserialisiert die Antwort als JSON-Objekt |
| PostAsJsonAsync <T>(string requestUri, T value) | Sendet einen POST-Request an die Uri mit serialisiertem Content als JSON im Body |
| PutAsJsonAsync (string requestUri, T value) | Sendet einen PUT-Request an die Uri mit serialisiertem Content als JSON im Body |

HttpClient-Beispiel

```
public partial class Home
{
    [Inject ]private HttpClient _client {get; set;}
    private readonly JsonSerializerOptions _options;
    public Qotd Qotd {get; set;}

    protected override async Task OnInitializedAsync()
    {
        _options = new JsonSerializerOptions { PropertyNameCaseInsensitive = true };
        var response = await _client.GetAsync("https://localhost:1234/api/qotd");
        var content = await response.Content.ReadAsStringAsync();
        if (!response.IsSuccessStatusCode) {
            throw new ApplicationException(content);
        }
        Qotd = JsonSerializer.Deserialize<Qotd>(content, _options);

        //Alternative Kurzversion
        Qotd = await _client.GetFromJsonAsync<Qotd>("https://localhost:1234/api/qotd");
    }
}
```

Formulare

- Komponente **EditForm** als zentrales Webformular
- Eröffnet einen **EditContext**, den alle untergeordneten Eingabe- und Validierungskomponenten zum Datenaustausch verwenden
- Built-In Eingabe-Komponenten
- Validation via DataAnnotations des Models
 - Built-In-Validation
 - Custom ValidationsAttribute
- Handler für Formularübermittlung

■ Bindung an Model oder EditContext

```
<EditForm Model=@author>...</EditForm>
```

```
@code { private Author author {get; set;} = new();}
```

```
<EditForm EditContext=@editContext>...</EditForm>
```

```
@code {  
    private Author author {get; set;} = new() { Name = "Simpson"};  
    private EditContext? editContext {get; set};  
  
    protected override void OnInitialized()  
    {  
        editContext = new EditContext(author);  
    }  
}
```

EditForm-Formularübermittlungsereignisse

- EditForm besitzt drei Ereignisse
- Nicht alle drei Ereignisse dürfen gleichzeitig Handler haben (OnValidSubmit/OnInvalidSubmit oder OnSubmit)

| Ereignis | Beschreibung |
|------------------------|---|
| OnSubmit | Wird ausgelöst, wenn Formular übermittelt wird |
| OnValidSubmit | Wird ausgelöst, wenn Formular übermittelt wird und alle Validatoren erfolgreich geprüft haben |
| OnInvalidSubmit | Wird ausgelöst, wenn Formular übermittelt wird und mindestens ein Validator einen Fehler meldet |

Eingabe-Komponenten

| Komponente | HTML-Repräsentation |
|-------------------------------|--|
| <EditForm> | <form> |
| <InputText> | <input type="text"> |
| <InputTextArea> | <textarea> |
| <InputSelect> | <select> |
| <InputNumber> | <input type="number"> |
| <InputCheckbox> | <input type="checkbox"> |
| <InputRadio><InputRadioGroup> | <input type="radio"> |
| <InputDate> | <input type="date"> |
| <InputFile> | <input type="file"> |
| <DataAnnotationsValidator> | - |
| <ValidationSummary> | <ul class="validation-errors"> <li class="validation-message">... |
| <ValidationMessage> | <div class="validation-message">...</div> |

Datenbindung & Validierung

- Eingabekomponenten werden via Two-Way-Datenbindung eingebunden
- Validation via ValidationMessage-Komponente und **DataAnnotationValidator**

```
<EditForm Model="@autor" OnValidSubmit="@HandleValidSubmit">  
  
  <DataAnnotationsValidator />  
  <ValidationSummary />  
  
  <div>  
    <InputText id="Name" @bind-Value="@autor.Name">/InputText>  
    <ValidationMessage For="@(() => autor.Name)" />  
  </div>  
  
  ...  
</EditForm>  
  
@code {}
```

Interoperabilität JavaScript

.NET ↔ Javascript

- Blazor kann nicht direkt auf das DOM zugreifen
- Zentrale Schnittstelle ist die **IJSRuntime**, welche mit [Inject] injiziert wird
- Möglichkeit 3rd-Party-Bibliotheken zu nutzen
- JavaScript-Dateien werden im **wwwroot** abgelegt und müssen in der **index.html** oder **App.razor** verwiesen werden
- Alternativ kann man die JS-Isolation als Modul in der Komponente nutzen **{KomponentenName}.razor.js**
- Komponenteninstanz kann JS übergeben werden
- JavaScript kann statische wie auf Instanzmethoden aufrufen

.NET nach JavaScript

- **InvokeVoidAsync** => keine Rückgabe von JS
- **InvokeAsync<T>** => mit Rückgabe von JS

//JS in Datei example.js in wwwroot/js

```
function showAlert(message) { alert(message); }  
function confirm(message) { return window.confirm(message); }
```

//index.html

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

//Komponente

```
[Inject] IJSRuntime JsRuntime { get; set; }
```

```
public async Task EineMethode()  
{
```

```
    //Syntax => InvokeVoidAsync("Funktionsname","Parameter")
```

```
    await JsRuntime.InvokeVoidAsync("showAlert", "JS function called from .NET");
```

```
    var bool = await JsRuntime.InvokeAsync<bool>("confirm", "Wollen Sie wirklich ...? ");
```

```
}
```


.NET nach JavaScript als Modul

- Globaler window namespace bleibt unberührt
- Import in index.html ist nicht nötig. Laden bei Bedarf
- Variablen / Funktionen müssen mit **export** markiert werden
- Auch als Komponenten-Datei möglich (Komponente.razor.js)

```
//JS in Datei example.js in wwwroot/js
```

```
export function showAlert(message) { alert(message); }  
export function confirm(message) { return window.confirm(message); }
```

```
//Komponente
```

```
[Inject] IJSRuntime JsRnt { get; set; }  
private IJSObjectReference _jsModule;
```

```
public async Task EineMethode()  
{  
    _jsModule = await JsRnt.InvokeAsync<IJSObjectReference>("import", "./js/example.js");  
    _jsModule.InvokeVoidAsync("showAlert", "Js Function aufgerufen von .NET");  
}
```

JavaScript nach .NET

- JS kann statische oder .NET Instanzmethoden aufrufen
- **[JSInvokable]** Attribut markiert, dass Methode durch JS aufgerufen werden kann. Methode muss public sein
- **DotNet**-Objekt in JS um statische C# Methoden aufzurufen

```
//Komponente C#
```

```
[JSInvokable] //Methodenname muss unique sein oder mit Name [JSInvokable("cqw")]
```

```
public static string CalculateQuadratWurzel(int number) =>
```

```
    $"Die Quadratwurzel von {number} beträgt {Math.Sqrt(number)} ";
```

```
//JavaScript
```

```
export function calculateQuadratwurzel() {
```

```
    //DotNet.invokeMethodAsync("Name Assembly", "Methode" , Nutzdaten)
```

```
    DotNet.invokeMethodAsync("BlazorWasmExample", "CalculateQuadratWurzel", 10)
```

```
        .then(result => {
```

```
            var el = document.getElementById("eineid");
```

```
            el.innerHTML = result;
```

```
        });
```

```
}
```

JavaScript zu C# mit Instanz

- DotNetObjectReference kann JS übergeben werden
- invokeMethod bzw. invokeMethodAsync verfügbar

//Komponente C#

```
public async Task SendDotnetInstanceToJs()
{
    var dotNetObjRef = DotNetObjectReference.Create(this);
    await JsRuntime.InvokeVoidAsync("showOnlineStatus", dotNetObjRef);
}
```

[JSInvokable]

```
public void SetOnlineStatus(bool isOnline) { ... }
```

//JavaScript

```
export function showOnlineStatus(dotNetObjRef) {
    //C# Methodenname und Parameter
    dotNetObjRef.invokeMethodAsync("SetOnlineStatus", navigator.OnLine)
}
```



JavaScript

Razor Class Library (RCL)

- Ziel der projektübergreifenden Wiederverwendbarkeit und Kapselung von Komponenten
- Normale .NET Assemblys
- Referenzieren Microsoft.AspNetCore.Components.Web
- Importieren des Namespace in **_Imports.razor** des Blazor-Projekts via @using
- CSS-Dateien aus wwwroot werden mit **<link href=„_content/{Package ID}/{Pfad und Datei-Name} />** in _index.html eingebunden
 - Packageld ist normalerweise der AssemblyName
 - Einbindung in index.html nicht nötig bei CSS- oder JS-Isolation

Beispiel

//RCL

//Komponente

```
namespace MeineLibrary.Components;  
public partial class OnlineStatus {...}
```

//CSS in wwwroot der Komponente

//Datei meinekomponente.css

//Blazor - Anwendung

//Datei _Imports.razor

@using MeineLibrary.Components

//Index.html bzw. App.razor

<link href="_content/MeineLibrary/meinekomponente.css" rel="stylesheet" />

//z.B. MainLayout.razor

<OnlineStatus />



RCL