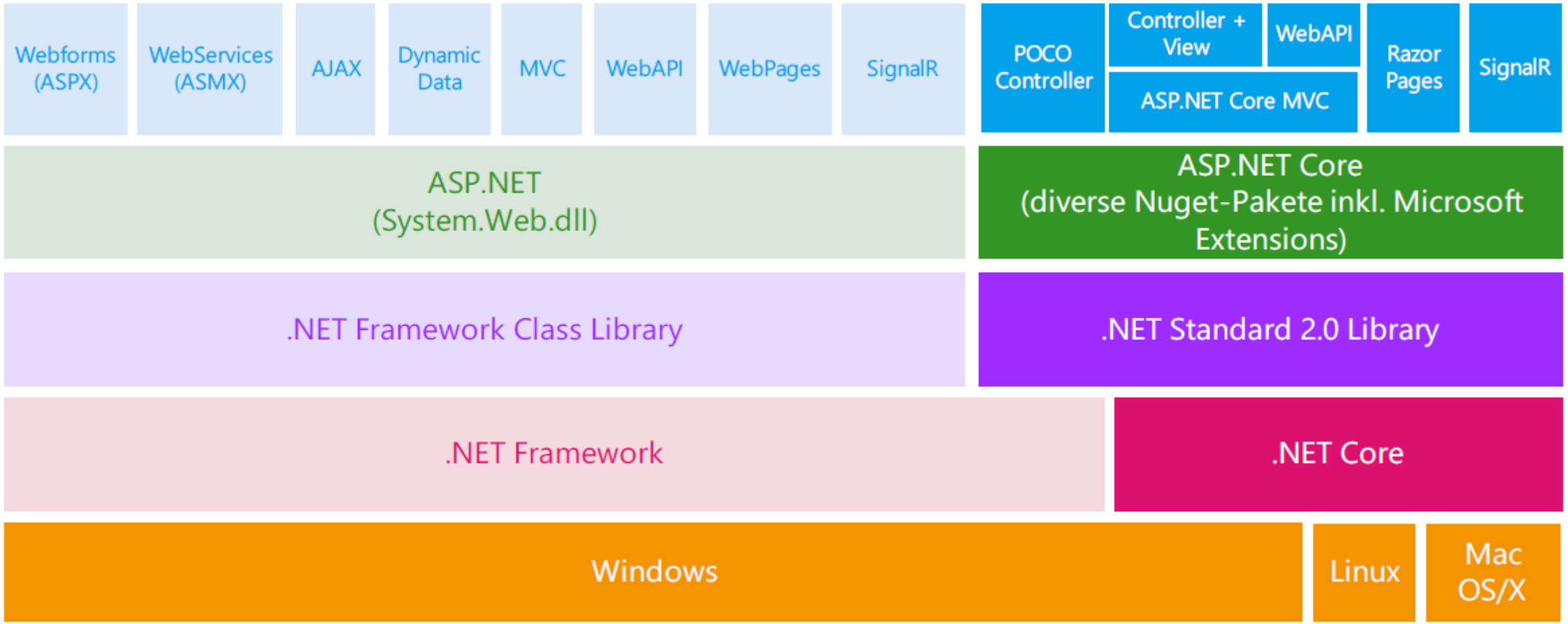
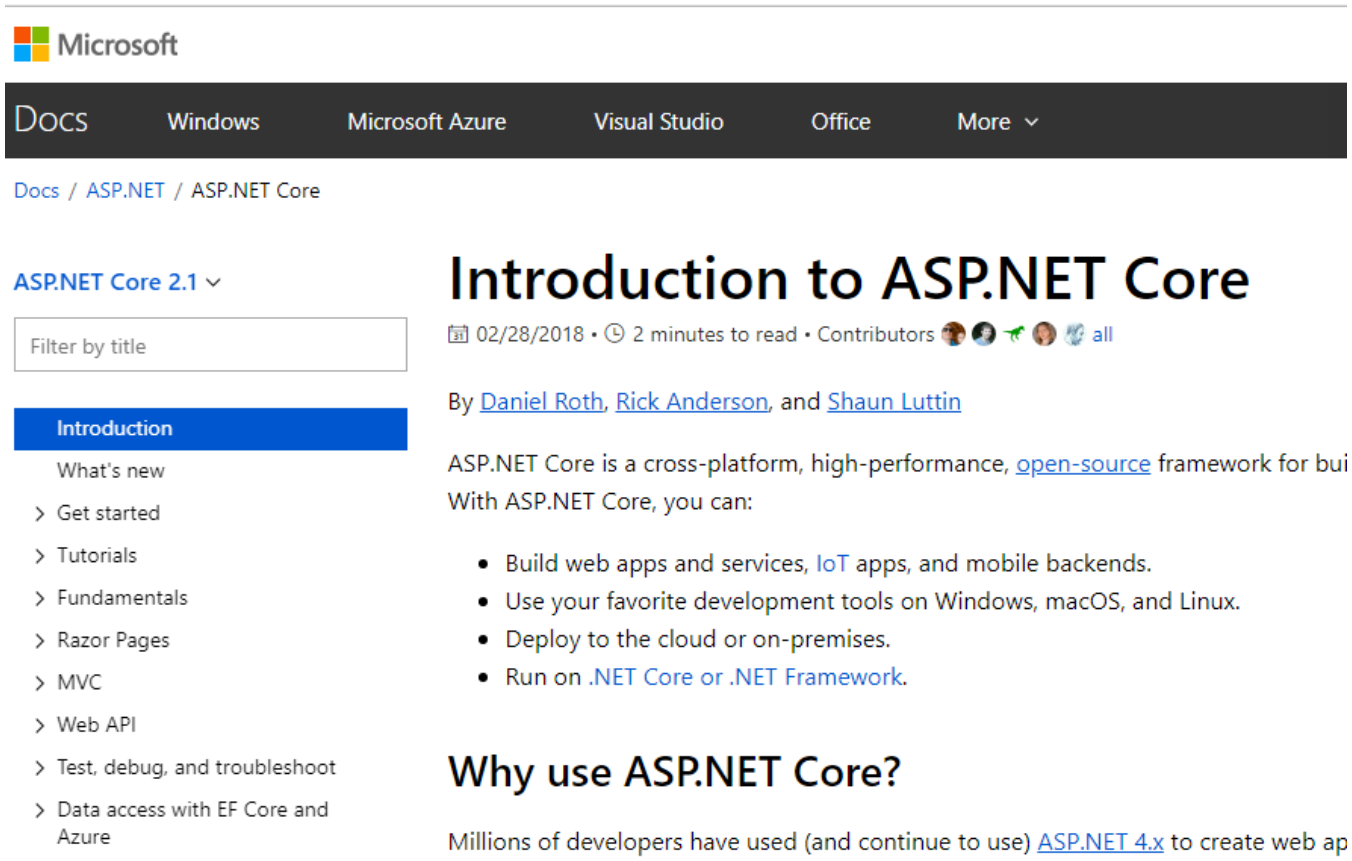
ASP.NET Core 2.x

Anwendungsmodelle in ASP.NET und ASP.NET Core



Dokumentation

https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1



Startup - Main

- Standard-Setup über CreateDefaultBuilder

https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.webhost.createdefaultbuilder?view=aspnetcore-2.0#Microsoft_AspNetCore_WebHost_CreateDefaultBuilder_System_String___

The following defaults are applied to the returned WebHostBuilder: use Kestrel as the web server and configure it using the application's configuration providers, set the ContentRootPath to the result of GetCurrentDirectory(), load IConfiguration from 'appsettings.json' and 'appsettings.[EnvironmentName].json', load IConfiguration from User Secrets when EnvironmentName is 'Development' using the entry assembly, load IConfiguration from environment variables, configures the ILoggerFactory to log to the console and debug output, enables IIS integration, and enables the ability for frameworks to bind their options to their default configuration sections.

- Startup-Klasse

```
public class Program
{
  public static void Main(string[] args)
  {
    CreateWebHostBuilder(args).Build().Run();
  }

  public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
      WebHost.CreateDefaultBuilder(args)
          .UseStartup<Startup>();
}
```

Startup-Klasse

- Konfiguration
- Dependency-Injection
- Aufruf der gewünschten Module

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the con
    public void ConfigureServices(IServiceCollection services)...

    // This method gets called by the runtime. Use this method to configure the HTTP requ
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)...
}
```

Dependency Injection

- Container-Aufbau in *ConfigureServices(IServiceCollection services)*
- diverse Methoden zum Hinzufügen von Objekten, Factories etc.
- Drei verschiedene Instanzierungsvarianten
  - Singleton
  - Scoped
  - Transient
- Vorgefertigte Methoden wie AddMvc, AddDbContext, AddLogging etc.

Konfiguration

- Verschiedene Quellen
  - JSON-Dateien
  - XML
  - Systemvariablen
  - Standardquellen:
    - appsettings.json
    - appsettings.[EnvironmentName].json
    - User Secrets für "Development"

Logging

- Einbau über Dependency Injection (ILogger<T>)
- Build-In-Provider:
  - Console, Debug, EventLog usw.

- Beliebig erweiterbar
  - z. B. Serilog-File-Logging

  NuGet: Serilog.Extensions.Logging.File

  https://github.com/serilog/serilog-extensions-logging-file

  https://andrewlock.net/creating-a-rolling-file-logging-provider-for-asp-net-core-2-0/

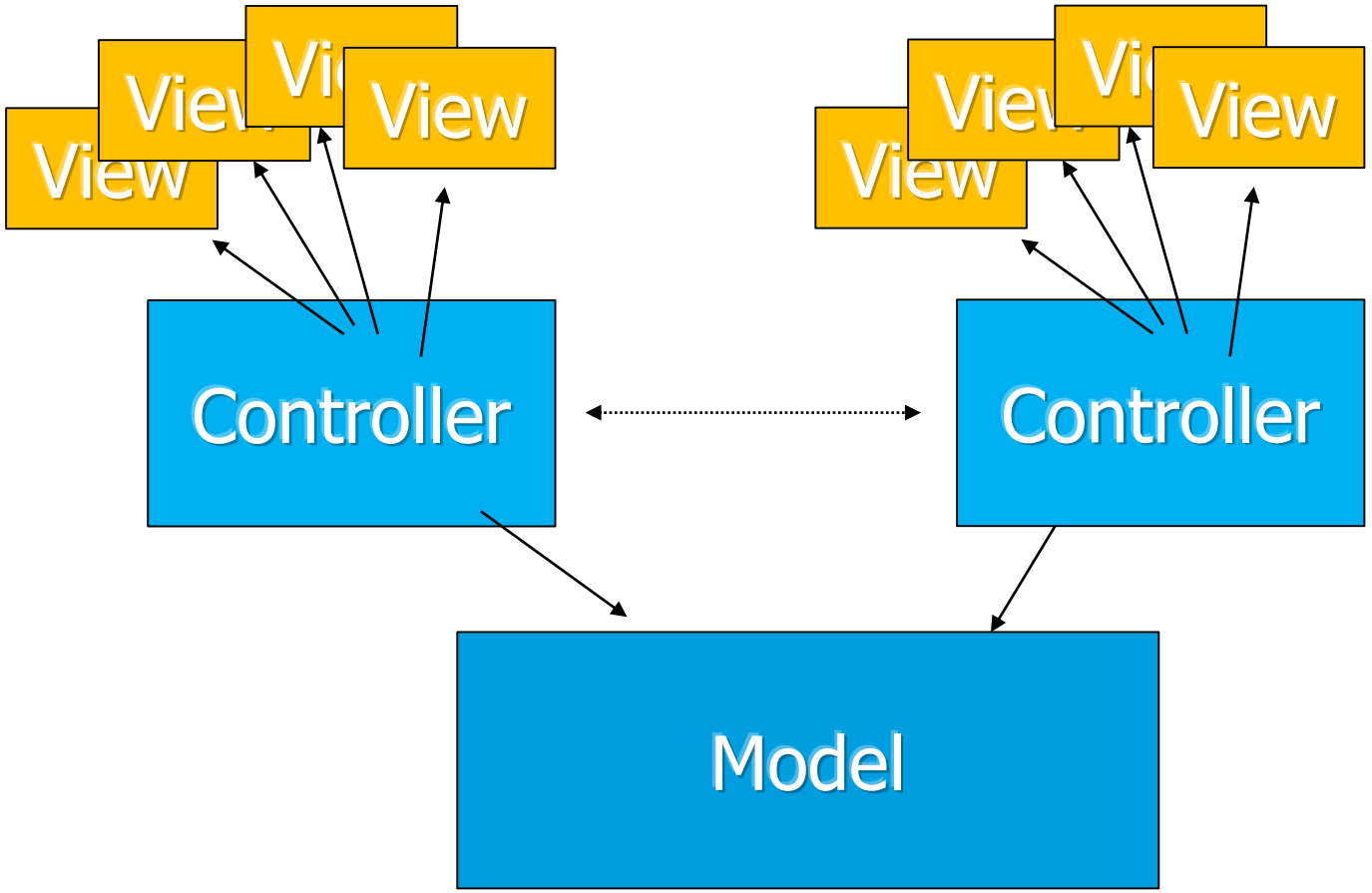
- Konfigurierbar über Konfigurationsdatei

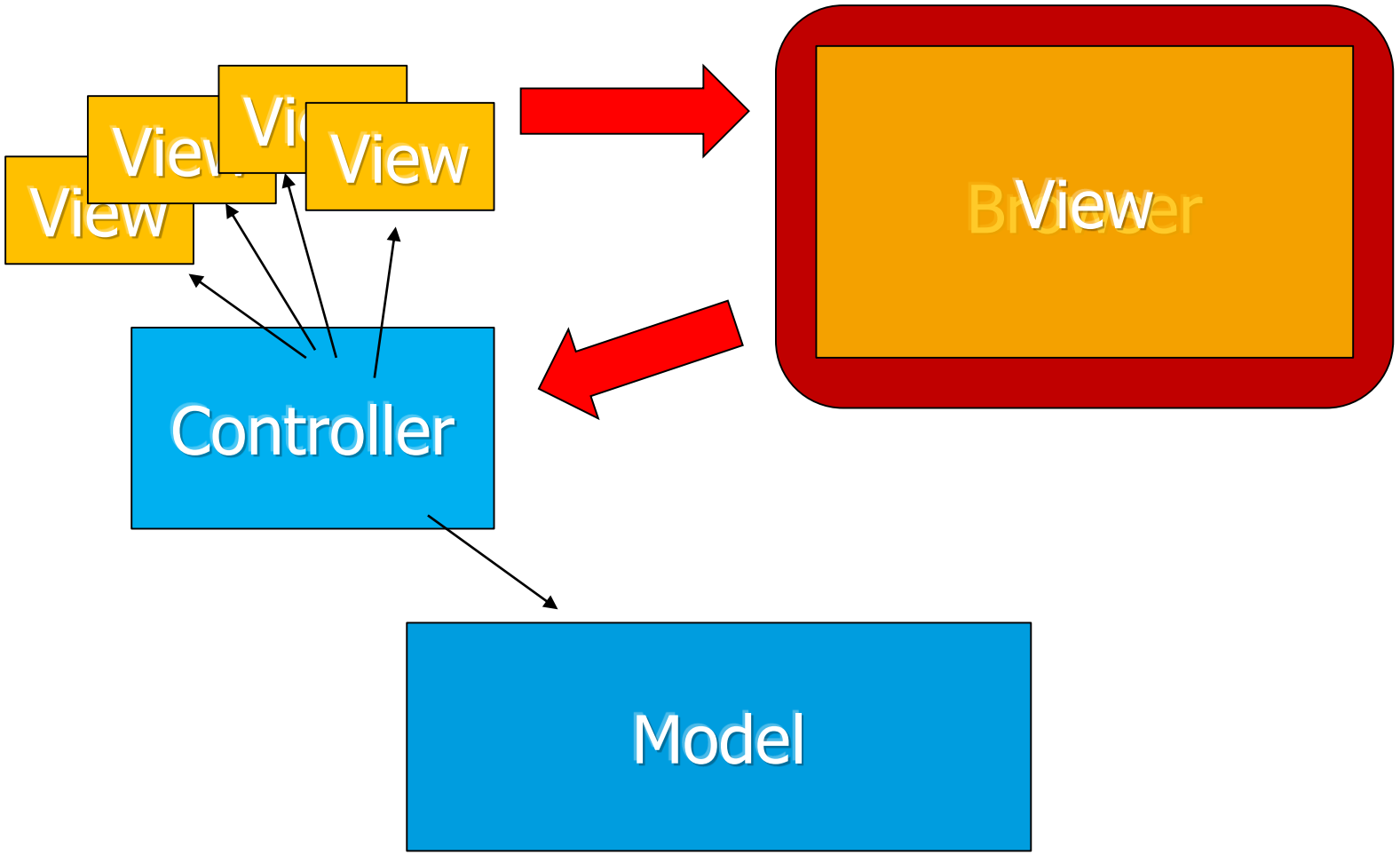
Was ist MVC?

MVC

- Model
  - Beinhaltet Geschäftslogik, Datenmodell
- View
  - Definiert, was der Browser anzeigen soll
- Controller
  - Steuert die Bedienlogik, wählt die Views aus und versorgt sie mit Informationen

Model, Controller, Views




View
View
View
View
Controller
Browser
View
Model

Standard-Projektstruktur

Verzeichnis	Ablage für
Controllers	Controller-Klassen
Models	Geschäftslogik, Datenstrukturen
Views	Views, nach Controller geordnet
Scripts	JavaScript-Bibliotheken
Images	Eigene Bilder
Content	CSS-Files etc.
App_Data	Anwendungsdaten
App_Start	Konfiguration von Routing, Bundling etc.

Konventionen

- Namenskonventionen ersetzen Deklarationen
  - Methodenname im Controller entspricht Name einer View
  - Name der Controller-Klasse / View korrespondieren mit Url
    …/mondial/countries ->
    Controller: MondialController.cs
    View: Views\Mondial\Countries.cshtml
  - Parameternamen der Controller-Methoden werden auf Http-Parameter / Query-Parameter abgebildet
  - Innerhalb der Views können Eigenschaften von Model und ViewBag über Name / Id mit Input-Controls gebunden werden

Abarbeitung einer Anforderung

1. MvcHandler instanziert eine Controller-Factory
2. Die Factory erzeugt einen Controller entsprechend der URL und der MvcHandler ruft dessen Execute-Methode auf
3. ControllerActionInvoker analysiert den RequestContext und wählt die richtige Action aus
4. ControllerActionInvoker ermittelt die Parameter für die Action
5. ControllerActionInvoker führt die Action aus

ActionResult-Typen (Auswahl)

- ContentResult
- ViewResult
- PartialViewResult
- RedirectResult
- FileResult
- FileStreamResult
- JsonResult
- JavaScriptResult
- HttpNotFoundResult
- HttpStatusCodeResult
- HttpUnauthorizedResult

MIME-Types

- Einige Result-Typen benötigen die Angabe eines Content-Types
- z. B.

MIME-Typ	Dateiendung	Bedeutung
application/pdf	*.pdf	Adobe PDF-Dateien
application/xml	*.xml	XML-Dateien
image/jpeg	*.jpeg *.jpg	JPEG-Dateien
text/xml	*.xml	XML-Dateien

siehe: http://de.selfhtml.org/diverses/mimetypen.htm

Methoden, die ActionResult zurückgeben

- View
- PartialView
- Content
- File
- Json
- JavaScript
- Redirect…
- HttpNotFound

Datenübergabe Controller -> View

- Model
  - beliebiger Datentyp
  - Intellisense und Typsicherheit auf der View-Seite
- ViewBag / ViewData
  - ViewData ist ein Dictionary. Keys können beliebige Strings sein
  - ViewBag ist das dynamic-Pendant hierzu. Die Eigenschaftsnamen müssen gültige .NET-Namen sein. Im Hintergrund hält ViewData die Key/Value-Paare

@Razor

Intuitive C#-Programmierung im HTML-Code

http://weblogs.asp.net/scottgu/introducing-razor

@Razor - Code Expressions

- Umschaltung zu C#-Code in HTML
- Sollte intuitiv benutzt werden
- Code Expressions
  - Ergebnis wird in Response integriert
  - … @Model.Vorname …
  - bei Bedarf Klammern setzen:
    @(Model.Alter * 7 + 1)
  - oder Escape-Sequenz @@, wenn @ als Text vorkommen soll (Twitter, E-Mail etc.)
- Automatische HTML-Codierung der Ausdrücke
  - Bei Bedarf abschaltbar:
    @Ajax.JavaScriptStringEncode(@Model.Vorname)
    @Html.Raw("<h1>dingsbums</h1>")

@Razor - Code-Blöcke

- Beinhalten C#-Code
  - @{ int zahl=77; //überall in der View verfügbar
    ...
    }
  - @if(ausdruck){...}
  - @foreach(...){...}
- Keine automatische Weitergabe an Response-Stream
- Jederzeit kombinierbar mit HTML

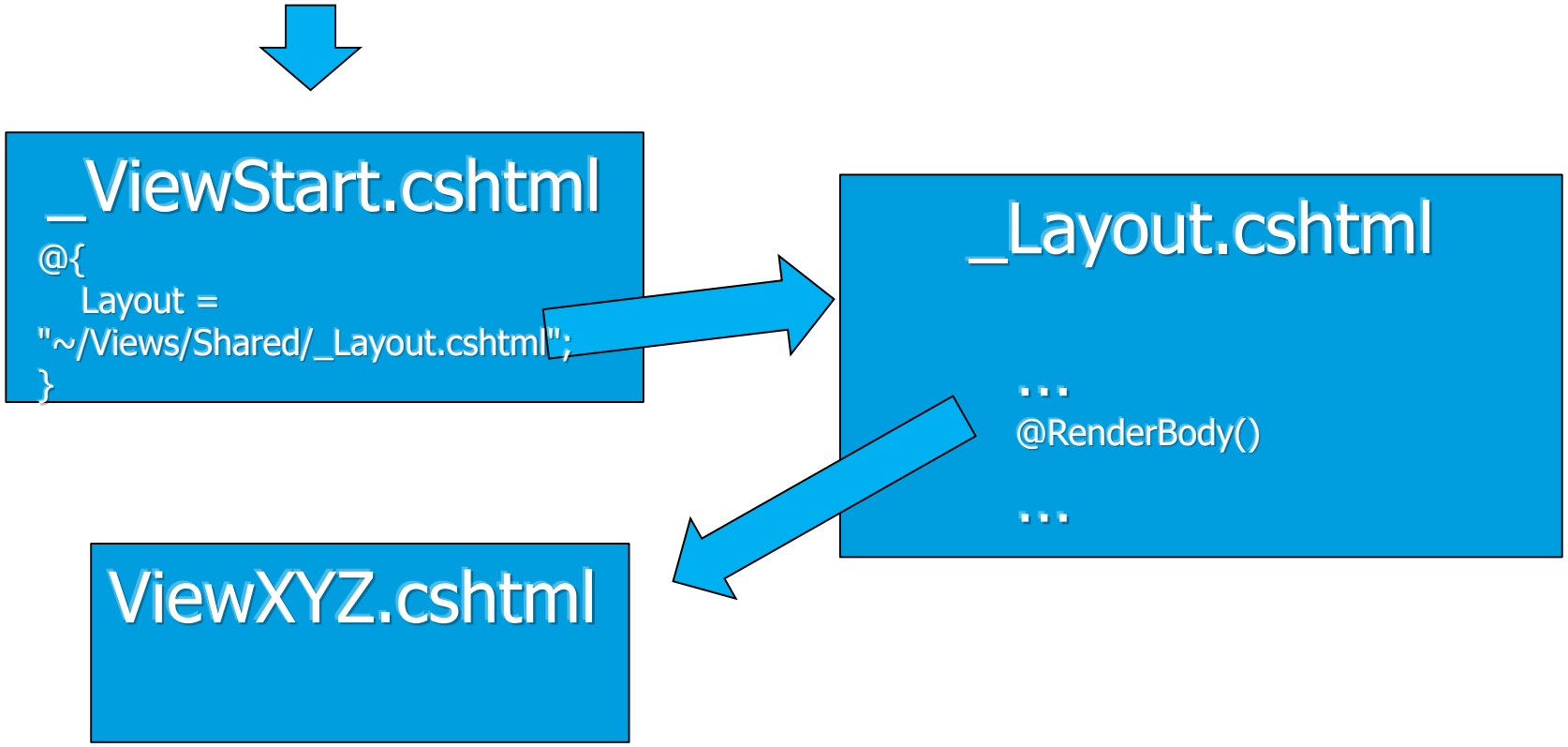
```
@if(...)
{
   int i=7;
   <h1>@i</h1>
   i++;
    <h2>@i</h2>
}
```

@Razor - Sonderfälle

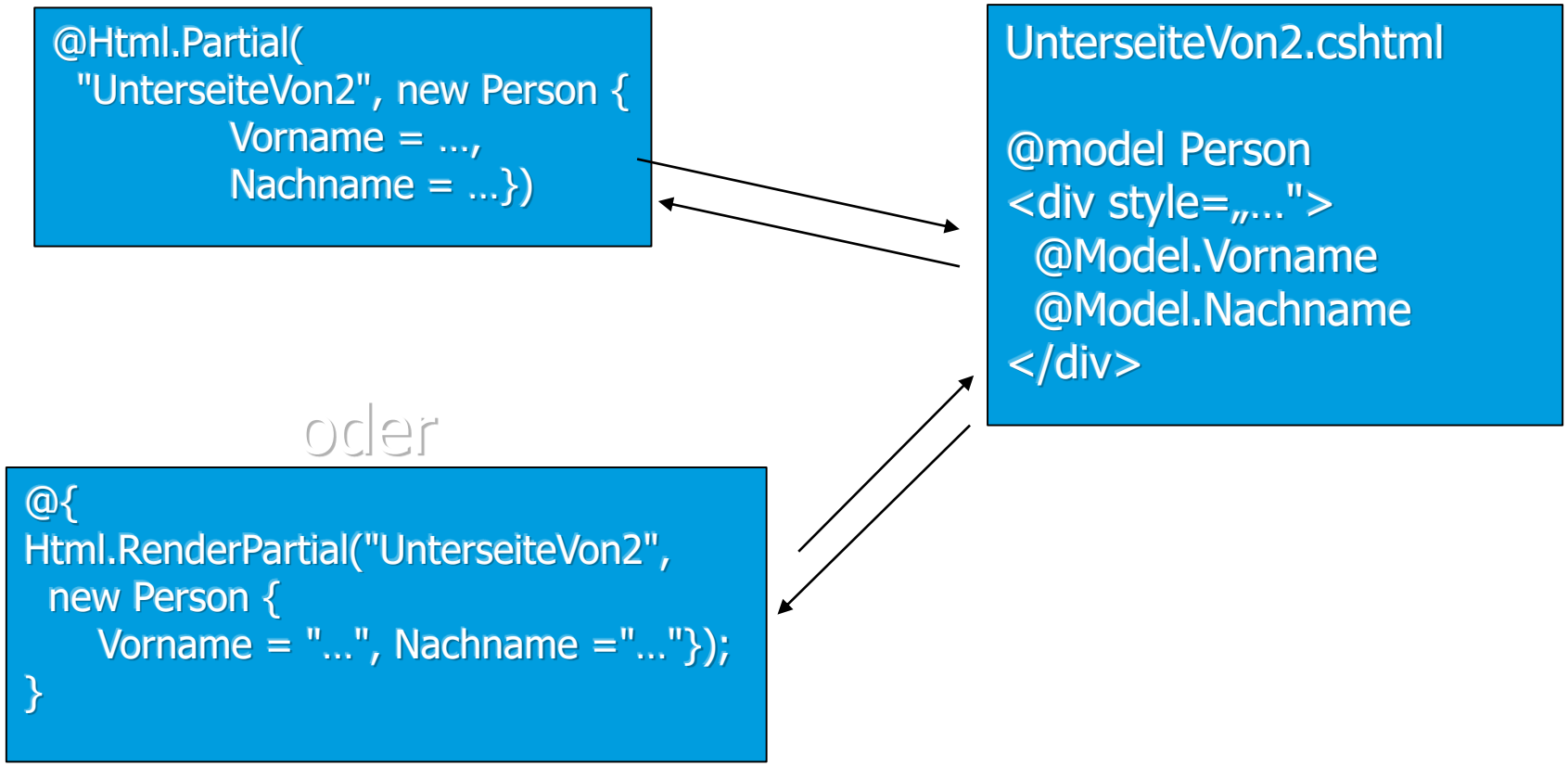
- Text innerhalb von Code-Blöcken:
  - @:
    @{… @:auszugebender Text (bis Zeilenende)
    …}
  - <text>-Blöcke:
    @{… <text> irgendwas </text> … }
- Kommentare
  @* alles Kommentar, @auch das und @das *@
- Aufruf generischer Methoden mit Klammern:
  @(Model.GenerischeMethode<EinTyp>(…))

Layout, Partial Views, Binding

Zusammenbau der Views zur Laufzeit



Partial Views (ohne Controller)



Model Binding

Model Binding

http://localhost:8403/Seminar/Seite4?vorname=Petra&nachname=Peters

Automatisch

```
public ActionResult Seite4(Person person)
{
  return View(person);
}
```

Explizit

```
public ActionResult Seite4()
{
  var person = new Person();
  UpdateModel(person);
  return View(person);
}
```

```
public ActionResult Seite4()
{
  var person = new Person();
  if (TryUpdateModel(person))
    return View(person);
  person.Nachname = "NoName";
  …
}
```

Forms und PostBack

```
@model Person

@using (Html.BeginForm())
{
  @Html.Label("Vorname")
  @Html.TextBox("Vorname")
  @Html.Label("Nachname")
  @Html.TextBox("Nachname")

  <button type="submit" >Speichern</button>
}
```

```
 [HttpPost]
 public ActionResult Personeneingabe(Person
person)
 {
   if (ModelState.IsValid)
     return View("Seite3", person);
   return View(person);
 }
```

HTML- und Url-Helper

Html-Helper

- Extension-Methods, die View.Html als ersten Parameter besitzen
- Vereinfachen HTML-Code, insbesondere bei Form-Elementen
- Sorgen automatisch für die Bindung zwischen Controls und Model
- String-basierte und typisierte Varianten, z. B.
  - @Html.Label("Vorname")
  - @Html.LabelFor(p=>p.Vorname)

HTML-Helper (strongly typed)

- gleiche Namen wie zuvor, enden auf "For"
  - LabelFor, TextBoxFor, HiddenFor etc.
- 1. Parameter ist Lambda-Ausdruck, der auf die Model-Eigenschaft verweist

  @Html.LabelFor(model => model.Vorname, …

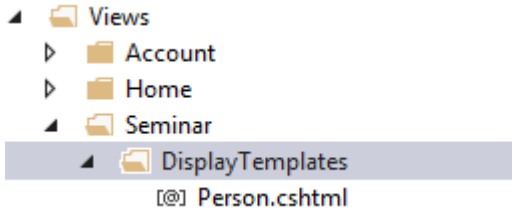
- + Intellisense
- + Compiler erkennt Fehler

Html-Helper

Typ	Aufgabe
Label	Beschriftung für Eingabefeld
TextBox, TextArea	Texteingabe (ein-, mehrzeilig)
DropDownList	<select > (Combobox)
ListBox	<select multiple="multiple"> (Listbox)
ValidationMessage	Meldung zu Eingabefehlern
Hidden	<hidden>-Field
Password	<input type="password">
RadioButton, CheckBox	<input type="radio/checkbox">
ActionLink	Hyperlink zu Controller-Action
RouteLink	Zusätzlich Angabe der Route möglich
Display	Templatebasierte Anzeige eines Typs
Editor	Templatebasierter Editor für einen Typ

Templatebasierte Helper

- EditorFor und DisplayFor suchen nach Template für den anzuzeigenden Typ (Unterordner EditorTemplates bzw. DisplayTemplates)



- Template als PartialView anlegen und nach Typ benennen

```
@model IEnumerable<Person>
@foreach (var p in Model)
{
  @Html.DisplayFor(m => p)
}
```

sucht nach Template für Typ Person

Reihenfolge der Template-Suche

1. Im Aufruf von EditorFor / DisplayFor angegebenes Template
2. [UIHint]- oder [DataType]-Attribut auf Property gesetzt
3. Template passend zum Datentyp (s. o.)
4. Behandlung als String

Eigene Helper mit @helper

- Allgemeingültige Helferlein im Ordner App_Code ablegen mit Endung .cshtml
- Definition

```
@helper RenderEingabefeld(string name,
string labeltext) {
  <div>
    <label title="@labeltext"
for="@name">@labeltext</label>
    <input type="text" id="@name"
```

- Nutzung

```
@Helferlein.RenderEingabefeld("vornam
e", "Vorname:")
@Helferlein.RenderEingabefeld("nachna
me", "Nachname:")
```

Eigene Helper als Extension-Method

- Definition als statische Klasse

```
public static class ExtensionMethods
{
    public static MvcHtmlString Taste(this
HtmlHelper helper,
                                string text, string
cssClass)
    {
      var builder = new TagBuilder("button");
```

- Nutzung

```
@using WebApplication28.Extensions
@Html.Taste("Bitte drücken", "btn btn-
default")
```

URL-Helper

- Ähnlich wie HTML-Helper, nur dass lediglich URLs generiert werden
- Action
  @Url.Action("Cities", new {Country="D"})

- Content
  @Url.Content("~/Images/bild.jpg")

/Mondial/Cities?Country=D

- RouteUrl
  @Url.RouteUrl("Default", new {Controller="Dings", Action="Bums", Id=42}, "https")

/Images/bild.jpg

https://localhost/Dings/Bums/42

URL-Helper (Forts.)

- Encode
  @Url.Encode("http://wo auch immer.com?a= b + c&usw")

```
http%3a%2f%2fwo+auch+immer.com%3fa%3d+b+%2b+c%26usw
```

- IsLocalUrl
  @Url.IsLocalUrl("/Mondial/UrlBeispiele") -> True
  @Url.IsLocalUrl("http://www.aldi.de") -> False

Konvertierungen über @Html

- @Html.AttributeEncode(" Hallo Welt ")

<p>Hallo Welt</p>

- @Html.Encode(" äöü ")

<p>äöü</p>

- @Html.Raw("<p>äöü</p>")

äöü

Tag Helpers

Tag Helpers

- HTML-Inline statt @Html.xxx

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.1

- Einbinden:
  - Pages/_ViewImports.cshtml (Razor Pages)
  - Views/_ViewImports.cshtml (MVC)

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<input type="hidden" asp-for="Kurs.Id" />
<div class="form-group">
    <label asp-for="Kurs.Kursnummer" class="control-label"></label>
    <input asp-for="Kurs.Kursnummer" class="form-control" />
    <span asp-validation-for="Kurs.Kursnummer" class="text-danger"></span>
</div>
```

Built-in Tag Helpers

- Anchor Tag Helper
- Cache Tag Helper
- Distributed Cache Tag Helper
- Environment Tag Helper
- Form Tag Helper
- Image Tag Helper
- Input Tag Helper
- Label Tag Helper
- Partial Tag Helper
- Select Tag Helper
- Textarea Tag Helper
- Validation Message Tag Helper
- Validation Summary Tag Helper

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/?view=aspnetcore-2.1

Data-Annotations und Validierungen

Data Annotations

- Attribute aus System.ComponentModel, System.ComponentModel.DataAnnotations und System.Web.Mvc
- Model- oder ViewModel-Klassen können über Attribute Darstellung und Validierung steuern
- Sprachanpassung über Ressourcen möglich (Demo)
- Vorteile:
  - deklarative Steuerung der Validierung in den Datenklassen
  - Deklaration von Standard-Ansichten
- Nachteile:
  - Aufgabenbereiche der Darstellungsschicht werden teilweise in die Models verlagert

Annotationen für die Anzeige

Attribut	Verwendung
Display	Anzeigename, Reihenfolge
ScaffoldColumn	Verbergen der Eigenschaft
DisplayFormat	Textformatierung des Eigenschaftswerts
DisplayColumn	Angabe, was z. B. bei DisplayTextFor von einem Objekt angezeigt werden soll
ReadOnly	Verhindert, dass der ModelBinder den Wert ändert
DataType	Typen für Strings, z. B. Password, EMail, Date, MultilineText
UIHint	Vorgabe eines Templates bei DisplayFor und EditorFor
HiddenInput	Eigenschaft soll als <hidden> gerendert werden

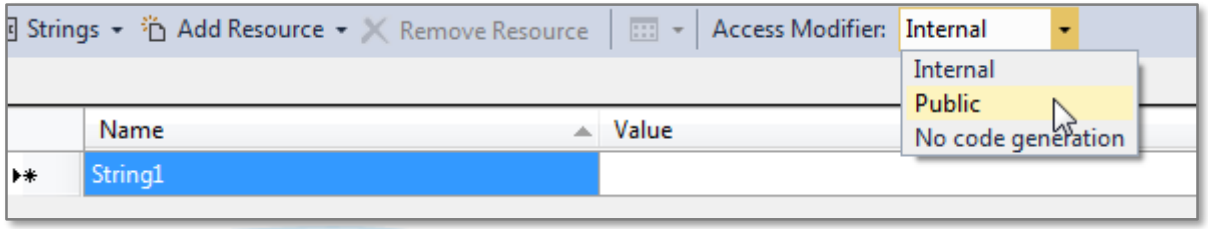
Unterschiedliche Auswirkungen der Attribute bei DisplayFor, EditorFor und DisplayTextFor

Lokalisierung der Attributtexte

- Statt fester Texte können in den Attributen auch Verweise auf Textressourcen eingetragen werden
  - ResourceType gibt einen Typ an, der öffentliche statische Eigenschaften enthält, die die Texte liefern
  - Name gibt den betreffenden Eigenschaftsnamen an
  - Einfache Vorgehensweise: Resource-Datei anlegen (auf Einstellung "public" achten)

web.config: System.Web

```
<globalization culture="auto" uiCulture="auto"
        enableClientBasedCulture="true"/>
```



Attribute bei automatisch generierten Klassen

- Anwendung z. B. beim Entity Framework
- Partial class anlegen (muss im selben Namespace sein!)
- Hier können neue Eigenschaften hinzugefügt und direkt attributiert werden
- Um vorhandene Eigenschaften mit Attributen zu versehen:
  - Attribut MetaDataType für partial class setzen und auf eine neue Klasse verweisen
  - In dieser Klasse öffentliche Felder oder Eigenschaften mit den Namen der zu attributierenden Eigenschaften anlegen (Typ ist unerheblich, kann object sein)
  - Diese Member mit den gewünschten Attributen versehen

Validierungen - ModelState

- ModelState gibt Auskunft über Validierungsfehler
- Abfragen:
  - ModelState.IsValid
  - ModelState.IsValidField("Vorname")
  - ModelState["Vorname"].Errors
- Setzen
  - ModelState.AddModelError("", "Oh weh - alles falsch...");
  - ModelState.AddModelError("Vorname", "ungültiger Vorname");
  - ModelState["Vorname"].Errors.Add("Nicht die schon wieder");
- Zurücksetzen
  - ModelState.Clear()

Validierungen

- Zugriff im Controller: Eigenschaft ModelState
- Zugriff in den Views: ViewData.ModelState
- Vorbereitete HTML-Helper für die Anzeige
  - @Html.ValidationSummary
  - @Html.ValidationMessageFor
- Eigene Validierung im Model implementieren:
  - Implementierung von IValidatableObject

```
public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
{
  if (this.Vorname == this.Nachname)
    yield return new ValidationResult("Vorname = Nachname??? Seltsam...",
      new string[] { "Vorname", "Nachname" });
}
```

Validierung über Attribute

Attribut	Verwendung
Required	Eingabe darf nicht leer sein
StringLength	Maximale und minimale Eingabelänge
RegularExpression	RegEx-Ausdruck für die Prüfung
Range	Wertebereich für numerische Daten
Remote	Aufruf einer Controller-Methode Hinweis: ~/bundles/jqueryval muss eingebunden werden
Compare	Vergleich mit anderem Eingabefeld (z. B. bei Passwort-Neueingabe
Eigene Attribute	Von ValidationAttribute ableiten und IsValid überschreiben
AllowHtml	Erlaubt die Eingabe von HTML-Code

Für clientseitige Validierung muss ~/bundles/jqueryval eingebunden werden! Insbesondere sind hier jquery.validate.js und

Eigene Validierungsattribute

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public class EvenNumberValidationAttribute:ValidationAttribute
{
  public override bool IsValid(object value)
  {
    if (value is int)
    {
      int number = (int)value;
      return (number % 2) == 0;
    }
    return false;

  }
}
```

Eigene clientseitige Validierung

- Zusätzlicher Aufwand auf Server- und vor allem Clientseite
  - Attributklasse muss IClientValidatable implementieren
  - Hinzufügen der Validierung im JavaScript-Code
    - $.validator.addMethod
    - $.validator.unobtrusive.adapters.addSingleVal
- Beispiele im Web
  - http://www.codeproject.com/Articles/275056/Custom-Client-Side-Validation-in-ASP-NET-MVC
  - http://www.codeproject.com/Articles/613330/Building-Client-JavaScript-Custom-Validation-in-AS
  - http://www.devtrends.co.uk/blog/the-complete-guide-to-validation-in-asp.net-mvc-3-part-2

Eigene Validation-Summary

```
<ol>
  @foreach (var err in this.ViewData.ModelState)
  {
    var id = err.Key;
    foreach (var item in err.Value.Errors)
    {
      <li>
        <a href="#@id">@item.ErrorMessage</a>
      </li>
    }
  }
</ol>
```

Sicherheitshinweis

Lassen Sie Ihre Eingabedaten nicht unbeaufsichtigt!

- Serverseitige Validierung
  - zwingend erforderlich
- Clientseitige Validierung
  - bietet KEINE Sicherheit
  - hilft nur dem Anwender

Asynchrone Controller Actions

async / await in Action-Methoden

```
[AsyncTimeout(300)]
[HandleError(ExceptionType =typeof(System.TimeoutException), View ="ZuLange")]
public async Task<ActionResult> About(CancellationToken ct)
{
  …
  await TuWasSchwieriges(ct);

  return View();
}
```

```
private  Task TuWasSchwieriges(CancellationToken ct)
{
  return Task.Run(new Action(()=>
  {
      …
      ct.ThrowIfCancellationRequested();
  }));
}
```

```
<p>@Model.Exception.Message</p>
```

Areas

Anwendungsbereiche mit Areas unterteilen

- Kapseln von Controllern, Models und Views
- Routendefinition über …AreaRegistration
- Registrierung über AreaRegistration.RegisterAllAreas() in Application_Start (Global.asax)
- Bei Links zusätzlich die Area angeben:
  @Html.ActionLink(…new { area = "Area51" }, …)

Routing

Routing-Attribute

- Prefix für Controller

```
[RoutePrefix("holla")]
[Route("{action}")]
public class DingsBumsController…
```

- Route Constraints bei Überladungen

```
[Route ("country/{carcode=D}")]
public ActionResult Action1(string carcode)…

[Route ("country/{id:int}")]
public ActionResult Action1(int id)…
```

- Optionale Parameter

```
[Route ("irgendwo/{id?}")]…
```

Route-Inline-Constraints (Auswahl)

Name	Beispiel	Beschreibung
bool	{n:bool}	A Boolean value
int	{n:int}	An Int32 value
minlength	{n:minlength(2)}	A String value with at least two characters
length	{n:length(2)} {n:length(2,4)}	A String value with exactly two characters A String value with two, three, or four characters
range	{n:range(1,3)}	The Int64 value 1, 2, or 3
alpha	{n:alpha}	A String value containing only the A–Z and a–z characters
regex	{n:regex (^a+$)}	A String value containing only one or more 'a' characters (a Regex match for the ^a+$ pattern)

File upload / download

File download

- HTML:

```
<a href="..." download="name">...</a>
```

- Controller:

```
public FileResult Get(string name)
    {
      name = Path.GetFileName(name);
      return File(System.IO.File.OpenRead(pfad), mimetype [, name]);
    }
```

File upload HTML

```
<form method="post" enctype="multipart/form-data">
    <input type="file" multiple="multiple" accept="image/*" asp-for="FileToUpload" />
</form>
```

File upload PageModel

```
[BindProperty]
public IEnumerable< IFormFile> FileToUpload { get; set; }
public async Task OnPostAsync()
   {
     foreach (var item in FileToUpload)
     {
       var stream = item.OpenReadStream();
       var buffer = new byte[item.Length];
       var n = await stream.ReadAsync(buffer, 0, buffer.Length);
       await System.IO.File.WriteAllBytesAsync(pfad, buffer);
     }
   }
```

Authorization

Beispiel

Start

- MVC-Anwendung anlegen
- Mondial-Lib einbinden
- Home/index -> Tabelle Kontinente anzeigen
- Home/continents -> Tabelle Countries anzeigen

- s. https://github.com/blowdart/aspnetauthorizationworkshop

Authentication einrichten

- Cookie-Authentication zur vereinfachten Demo

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(CookieAuthenticationDefaults.AuthenticationScheme,
        options =>
        {
                options.LoginPath = new PathString("/Account/Login/");
                options.AccessDeniedPath = new PathString("/Account/Forbidden/");
        });
```

```
app.UseAuthentication();
```

- Controller "Account" / Views und Actions "Login" und "Forbidden" anlegen
- [Authorize] HomeController... -> leitet um auf Login-Seite

Log-in-Form

```
public IActionResult Login(string returnUrl = null)
{
  return View(new LoginModel { ReturnUrl = returnUrl });
}
```

```
<form method="post">
  <label asp-for="Username">Name</label>
  <input asp-for="Username" />
  <input asp-for="ReturnUrl" type="hidden"/>
  <button type="submit">Log in</button>
</form>
```

Principal in Login erzeugen

```
[HttpPost]
public async Task<IActionResult> Login(LoginModel loginModel)
{
  const string Issuer = "https://mondial.com";
  var claims = new List<Claim>();
  claims.Add(new Claim(ClaimTypes.Name, loginModel.Username, ClaimValueTypes.String, Issuer));

  var userIdentity = new ClaimsIdentity("SuperSecureLogin");
  userIdentity.AddClaims(claims);
  var userPrincipal = new ClaimsPrincipal(userIdentity);

  await HttpContext.SignInAsync(
      CookieAuthenticationDefaults.AuthenticationScheme,
      userPrincipal,
          new AuthenticationProperties
          {
            ExpiresUtc = DateTime.UtcNow.AddMinutes(20),
            IsPersistent = false,
            AllowRefresh = false
          });

  if (Url.IsLocalUrl(loginModel.ReturnUrl))
  {
    return Redirect(loginModel.ReturnUrl);
  }
  return RedirectToAction("Index", "Home");
}
```

Benutzer anzeigen

```
@if (User.Identity.IsAuthenticated)
      {
        <div class="nav navbar-right navbar-brand">Hallo @User.Identity.Name</div>
      }
```

Authorization für alles

- [Authorize] für HomeController wieder entfernen
- in startup:

```
services.AddMvc(config =>
{
    var policy = new AuthorizationPolicyBuilder()
                     .RequireAuthenticatedUser()
                     .Build();
    config.Filters.Add(new AuthorizeFilter(policy));
})
```

- [AllowAnonymous] für AccountController

Roles (Demo-User-Repository)

- User / UserRepo anlegen:

```
public class User
{
  public string Name { get; set; }
  public string[] Roles { get; set; }
  public DateTime DateOfBirth { get; set; }
  public string[] ContinentsAllowedToEdit { get; set; }
}

public class UserRepo
{
  public IEnumerable<User> Users { get; set; } = new List<User>
  {
    new User{Name="Peter", Roles=new string[]{"Guest" }, DateOfBirth=new DateTime(1950,3,2) },
    new User{Name="Paul", Roles=new string[]{"Admin","Guest" }, DateOfBirth=new DateTime(1960,6,12) },
    new User{Name="Mary", Roles=new string[]{"Admin" }, DateOfBirth=new DateTime(1970,7,22) }
  };
}
```

- in startup:

```
services.AddSingleton<UserRepo>();
```

Rollen über Claims hinzufügen / anzeigen

- Im AccountController ergänzen

```
var user = userRepo.Users.FirstOrDefault(u => u.Name.ToLower() == loginModel.Username.ToLower());
if (user == null) return View();
      ...
claims.Add(new Claim(ClaimTypes.Name, user.Name, ClaimValueTypes.String, Issuer));
foreach (var role in user.Roles)
{
  claims.Add(new Claim(ClaimTypes.Role, role, ClaimValueTypes.String, Issuer));
}
```

- in _layout:

```
@if (User.Identity.IsAuthenticated)
      {
      <div class="nav navbar-right navbar-brand">
        Hallo @User.Identity.Name
        [@string.Join(", ", User.Claims.Where(c => c.Type == System.Security.Claims.ClaimTypes.Role).Select(r => r.Value))]

      </div>
      }
```

Rolle erzwingen

- Variante A: (sehr unflexibel)

```
[Authorize(Roles ="Admin")]
  public class HomeController
```

- Variante B: (über Policies)

```
services.AddAuthorization(options =>
{
  options.AddPolicy("AdministratorOnly", policy => policy.RequireRole("Admin"));
});
```

```
[Authorize(Policy = "AdministratorOnly")]
public class HomeController
```

Übung

- User um Eigenschaft *bool? HasChildren* ergänzen
- in AccountController Claim hinzufügen, wenn *HasChilden != null*
- Zweite Policy hinzufügen (*RequireClaim, nur Claimname, ohne Wert*)
- Auf HomeController testen
- dann noch einmal mit Wert ("True" bzw. "False") testen

Code based Policies

- Beispiel: IsExperienced -> Born before 1970
- Claim hinzufügen

```
claims.Add(new Claim(ClaimTypes.DateOfBirth, user.DateOfBirth.ToString(CultureInfo.InstalledUICulture),
        ClaimValueTypes.Date, Issuer));
```

- Requirement-Klasse anlegen (implementiert IAuthorizationRequirement)
- Handler anlegen (abgeleitet von AuthorizationHandler<Requirement-Klasse>)
  - HandleRequirementAsync implementieren
  - context.Succeed() aufrufen, wenn Bedingung erfüllt ist
- Policy hinzufügen (Requirements.Add(Instanz von Requirement-Klasse))
- Handler als Singleton bereitstellen
  AddSingleton<IAuthorizationHandler, Handler-Klasse>()

Übung: zweiten Handler hinzufügen

- Property IsExpert der User-Klasse hinzufügen
- Claim hinzufügen
- 2. Handler für IsExperiencedRequirement implementieren, der diese Property prüft
- Handler über DI bereitstellen nicht vergessen!
- Für User nach 1970 IsExpert auf True setzen, testen

Prüfung zur Laufzeit: Resource based requirements

- Requirement definieren wie zuvor
- Handler ableiten von *AuthorizationHandler<requirement-Klasse, Model>*
- Im Code prüfen:

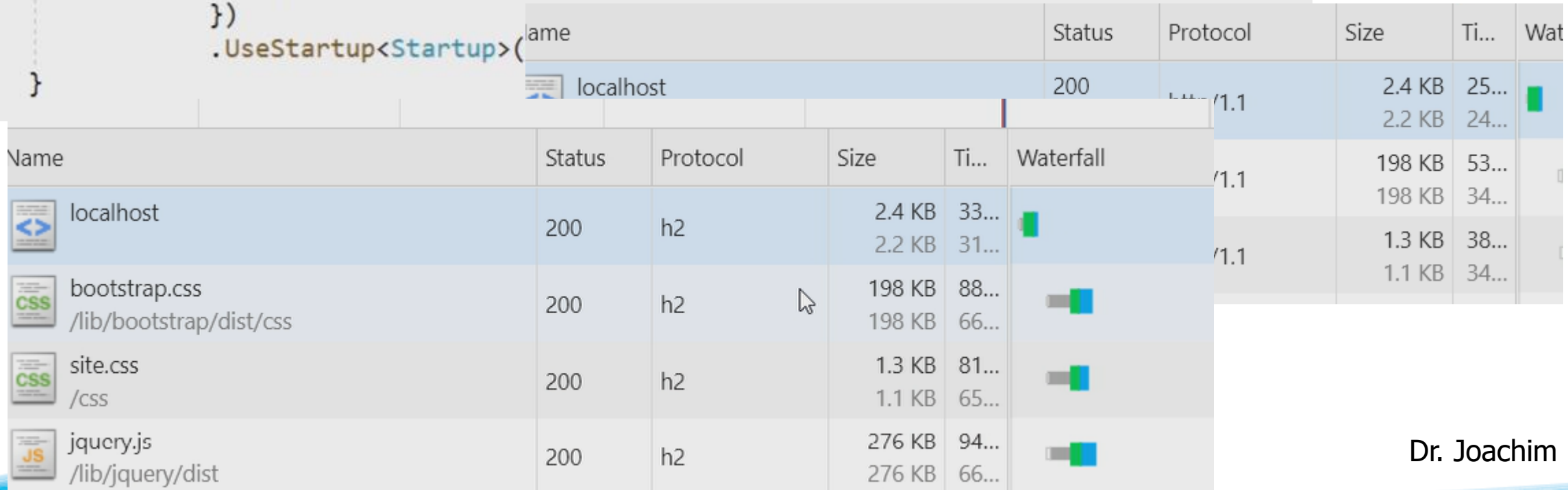
```
var authorizationResult = await authorizationService.AuthorizeAsync(User, model, requirement);
    if (authorizationResult.Succeeded)…
```

- oder in der View:

```
@{
    var requirement =new XYZRequirement ();
    var authResult = await AuthorizationService.AuthorizeAsync(User, model, requirement);
    if (authResult.Succeeded)      {…}
}
```

HTTP 2.0

```
public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .ConfigureKestrel(options =>
        {
            options.ListenAnyIP(5001, listenOptions =>
            {
                listenOptions.Protocols = HttpProtocols.Http1AndHttp2;
                listenOptions.UseHttps();
            });
        })
        .UseStartup<Startup>(
}
```



Health checks

- https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/health-checks?view=aspnetcore-2.2

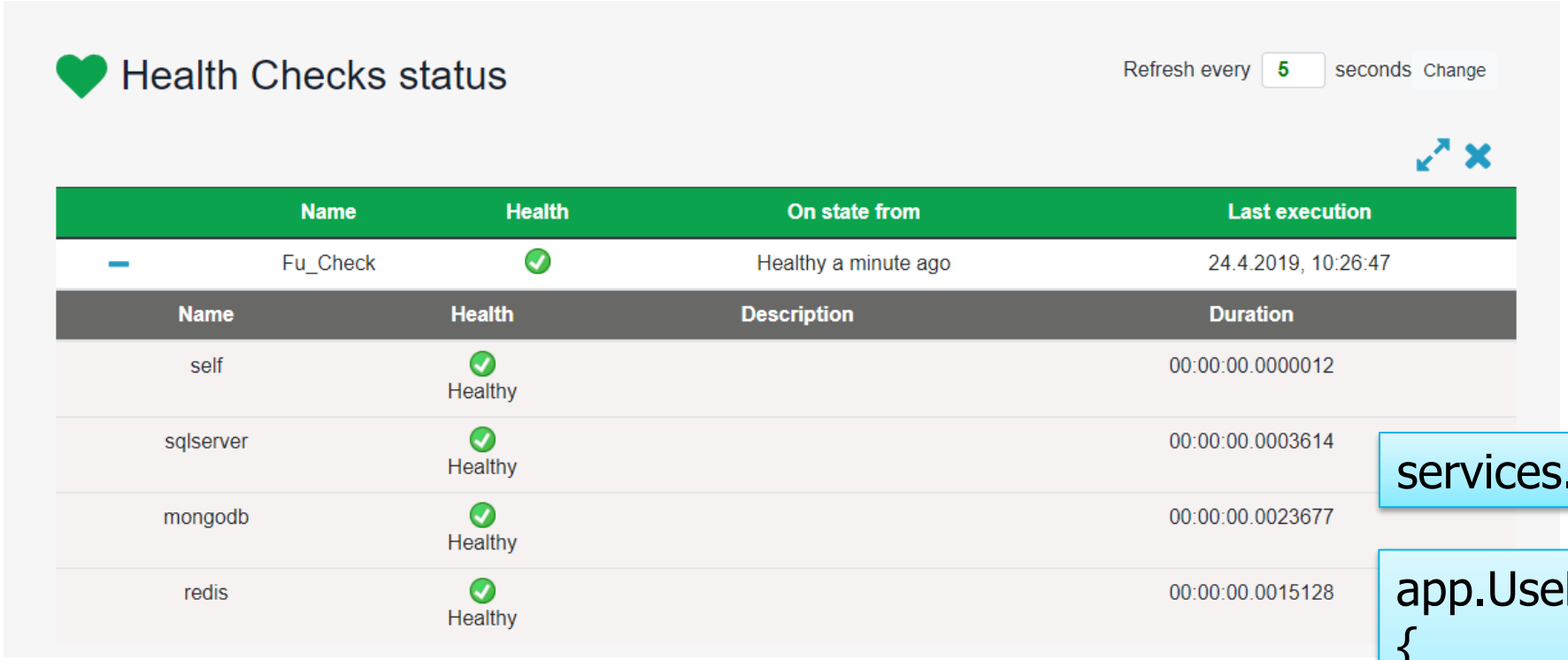
- Beat Pulse abgelöst durch Xabaril Enterprise HealthChecks for ASP.NET Core Diagnostics Package
- Nuget-Pakete für verschiedene Anwendungen:
  https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks

Health checks einrichten

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddHealthChecks()
    .AddCheck("self", c => HealthCheckResult.Healthy())
        .AddSqlServer(connString)
        .AddMongoDb(new MongoClientSettings() {...})
        .AddRedis("192.168.0.123:6379");
}
```

```
app.UseHealthChecks("/health", new HealthCheckOptions
    {
      Predicate = registration => true,// registration.Name=="self"
      ResponseWriter=UIResponseWriter.WriteHealthCheckUIResponse
    });
```

Health checks Monitor




```
services.AddHealthChecksUI();
```

```
app.UseHealthChecksUI(setup =>
{
  setup.UIPath = "/ui";
});
```

https://www.youtube.com/watch?v=kzRKGCmGbqo