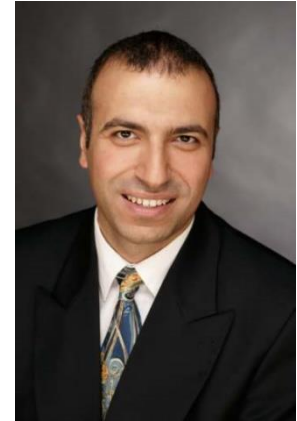


# ASP.NET Core 5 Api

Ali Ilci

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

- Freiberuflicher IT - Trainer (MCT)
- Themenschwerpunkte:
  - ASP.NET / MVC / WebApi / Forms
  - C#
  - Html5 & Css3
  - Angular JavaScript / TypeScript
- [www.ilci.de](http://www.ilci.de)



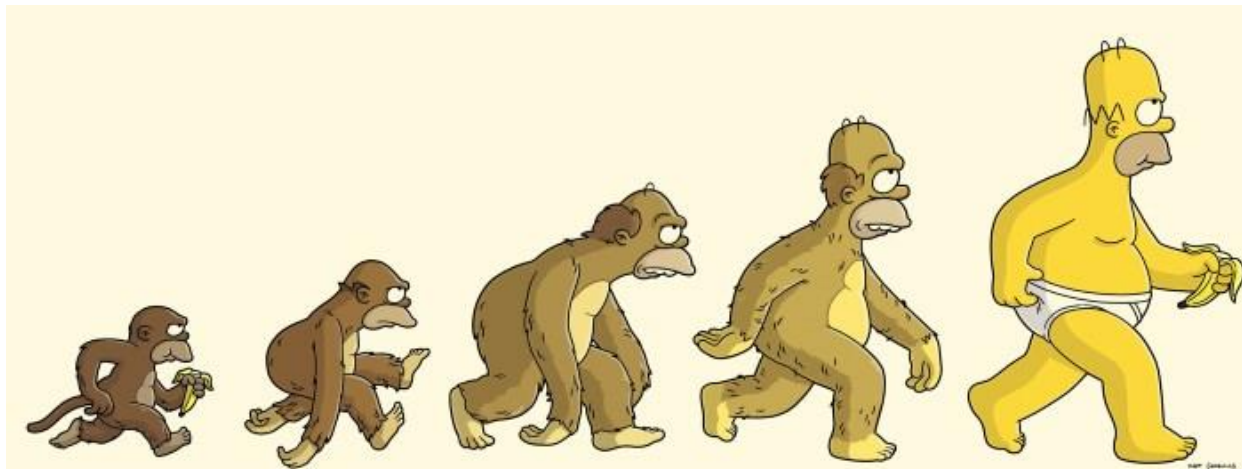
# Was ist ein Webservice?

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

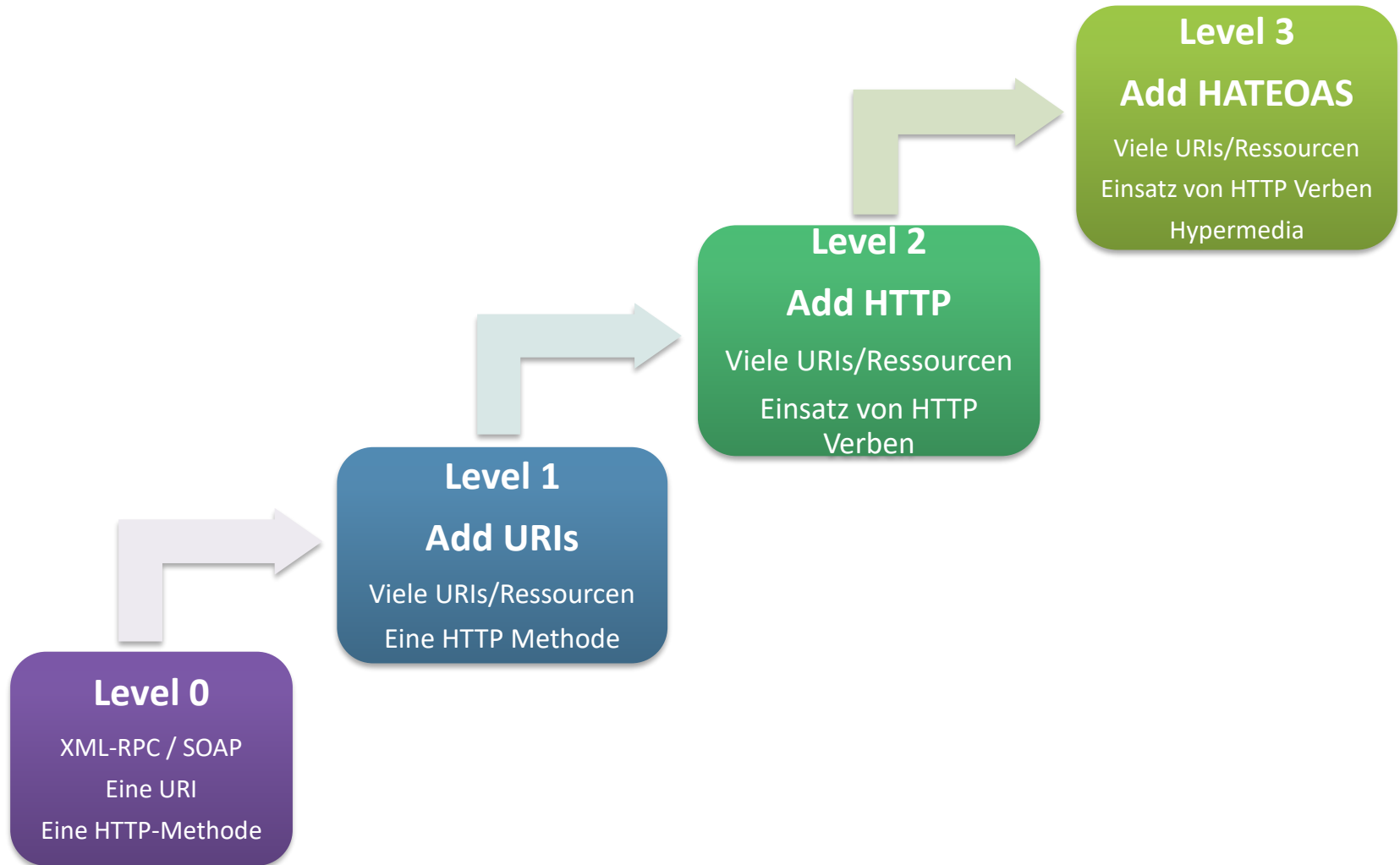
## ■ Hypermedia

- Verlinkung von Ressourcen zwecks Navigation

# HTTP Verben / REST Operationen

HTTP Verb	Beschreibung
<b>GET</b>	Gibt einen einzelnen oder eine Sammlung von Einträgen zurück
<b>POST</b>	Erzeugt einen neuen Eintrag
<b>PUT</b>	Aktualisiert einen bestehenden Eintrag (alle Eigenschaften)
<b>PATCH</b>	Aktualisiert einzelne Eigenschaften eines existierenden Eintrags
<b>DELETE</b>	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"
    }
  ]
}
```



# .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-Platform
	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

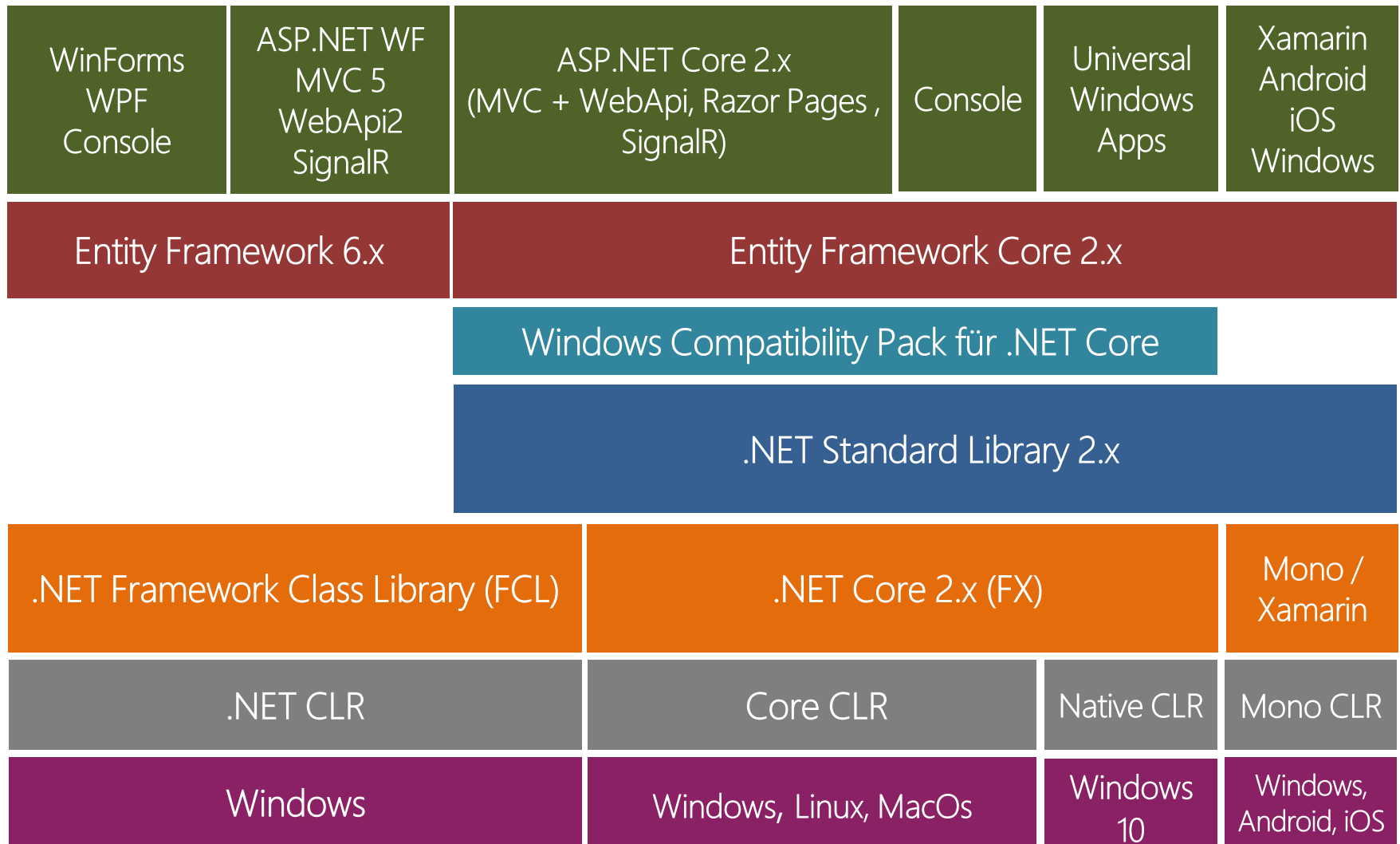
<https://docs.microsoft.com/de-de/dotnet/articles/standard/choosing-core-framework-server>

# Was ist .NET Standard 2.0

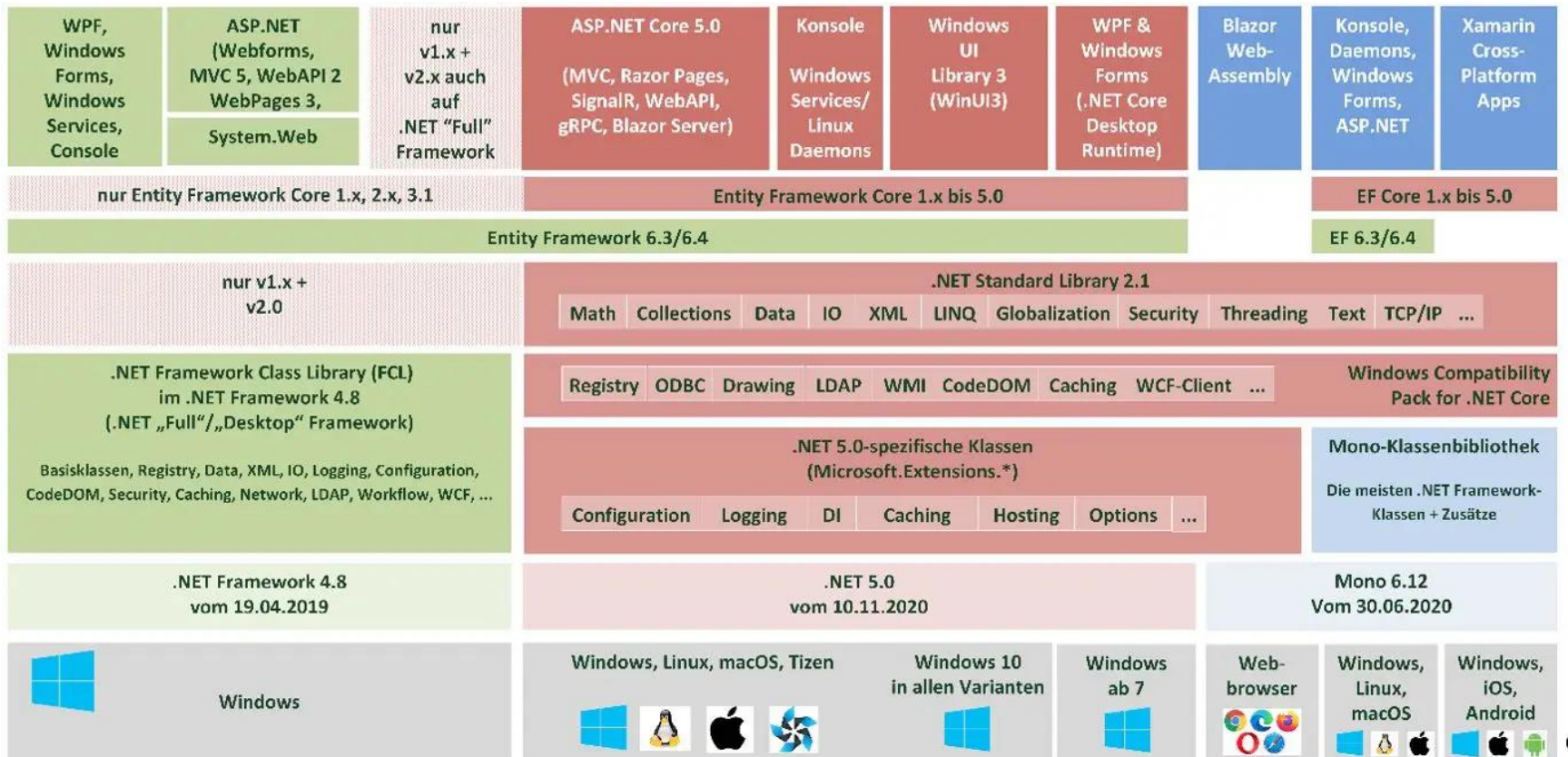
*„Der Standard ist eine technische Vorgabe für Programmierschnittstellen (APIs), die es ermöglicht, dass sich Programmcode zwischen verschiedenen Implementierungen von .NET austauschen lässt. Entwickler, die eine Klassenbibliothek auf Basis einer Version des .NET Standard realisieren, können sicher sein, dass diese Bibliothek auf allen .NET-Implementierungen funktioniert, die diese Version des Standards umsetzen.“*

- Immo Landwerth Programmanger .NET Entwicklungsteam

# Überblick .NET 2018



# .NET Familie 2020/2021



<https://www.heise.de/hintergrund/.NET-5-0-ist-erschienenen-4952894.html>

Holger Schwichtenberg Stand: 10.November 2020

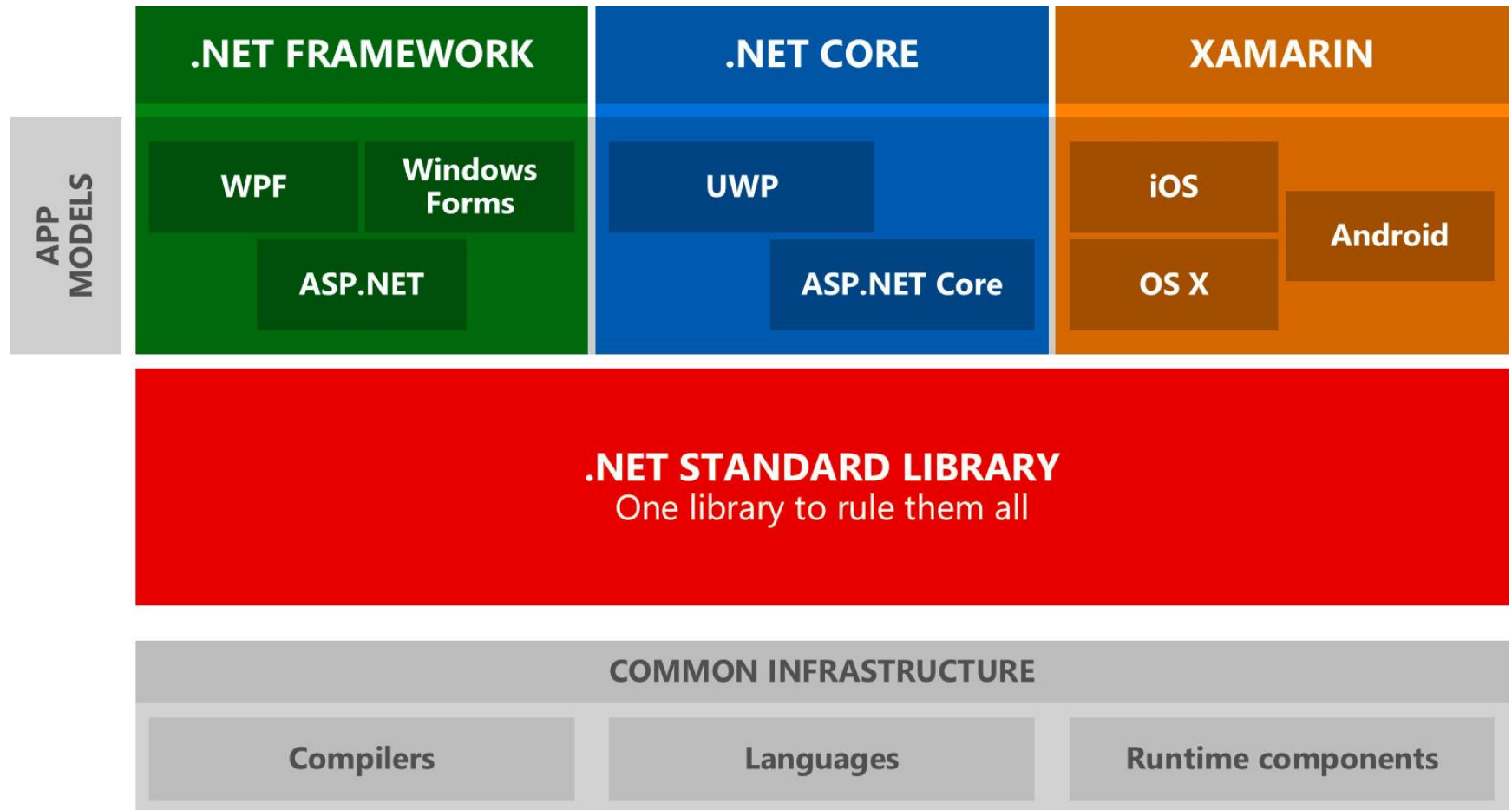
# Zukünftige .NET 5/6 Platform

## .NET – A unified platform





# .NET Standard 2 Familie



## .NET Schedule



- .NET Core 3.0 release in September
- .NET Core 3.1 = Long Term Support (LTS)
- .NET 5.0 release in November 2020
- Major releases every year, LTS for even numbered releases
- Predictable schedule, minor releases if needed

- 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/?tabs=netcore2x>

# .NET CLI Befehle „dotnet“ (Auszug)

Befehl	Beschreibung
<b>new</b>	Erstellt ein neues Projekt, eine Konfigurationsdatei oder eine Lösung auf Grundlage der angegebenen Vorlage.
<b>restore</b>	Stellt die Abhängigkeiten und Tools eines Projekts wieder her
<b>build</b>	Erstellt ein Projekt und alle seine Abhängigkeiten
<b>publish</b>	Packt die Anwendung und ihre Abhängigkeiten in einen Ordner für die Bereitstellung auf einem Hostsystem
<b>run</b>	Führt Quellcode ohne explizite Kompilierungs- oder Startbefehle aus.
<b>test</b>	.NET-Testtreiber, der verwendet wird, um Komponententests auszuführen.
<b>pack</b>	Packt den Code in ein NuGet-Paket
<b>clean</b>	Löscht die Ausgabe eines Projekts.
<b>sln</b>	Ändert eine .NET Core-Projektmappendatei.
<b>store</b>	Speichert die angegebenen Assemblys im Laufzeitpaketspeicher
<b>tool</b>	Installation von globalen .NET Core Tools
<b>watch</b>	Dateisystemüberwachung und automatische Projektneuerstellung (ab 2.1)

# Themen

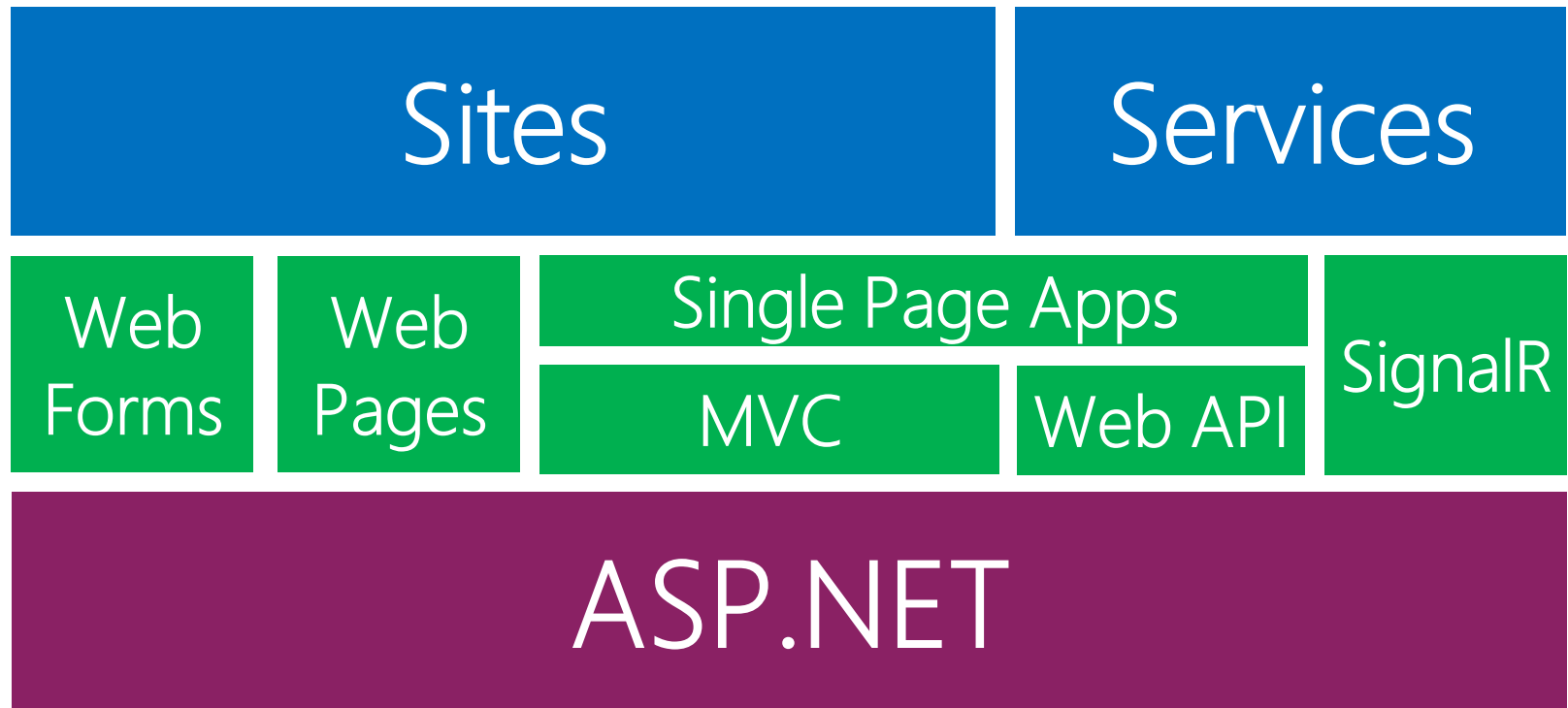
Repository  
REST  
Tools  
Model  
Web Technologien  
DTO  
CLI  
JQuery  
ASP.NET Core Api  
Filter  
Authentifizierung  
Logging  
HttpClient  
Routing  
Formatter  
Hosting  
Validation  
AJAX  
Webessentials  
Actions  
Cors  
Fehlerbehandlung  
Entity Framework  
Dependency Injection  
Action Filter  
Asynchronität  
Security  
Controller  
Data Annotations  
Best Practices

# ASP.NET Core 5.0

# Microsoft Web Technologien

Entwicklertools	Host	Code Ausführtechnologien	
Visual Studio	IIS	Server-Side	Client-Side
WebMatrix	SQL Server	ASP.NET	JavaScript
Visual Studio Code	Windows Azure	ASP.NET Core	JQuery
		ASP.NET Blazor	AJAX
			Knockout
			TypeScript
			ASP.NET Blazor Webassembly

# Überblick ASP.NET 4.5





# Überblick ASP.NET Core 2.0

Web Forms  
MVC  
Web API  
SignalR  
Web Pages

ASP.NET Core  
(MVC + WebAPI + Razor  
Pages, SignalR)

ASP.NET 4.6

ASP.NET Core  
2.X

.NET Framework

Nur Windows

.NET Core 2.x

Windows & Linux & MacOS

ASP.NET Core 5

(MVC + WebAPI + Razor Pages + SignalR + gRPC + Blazor)

EF Core 5

.NET Standard Library 2.1

.NET Core 5

Windows & Linux & MacOS

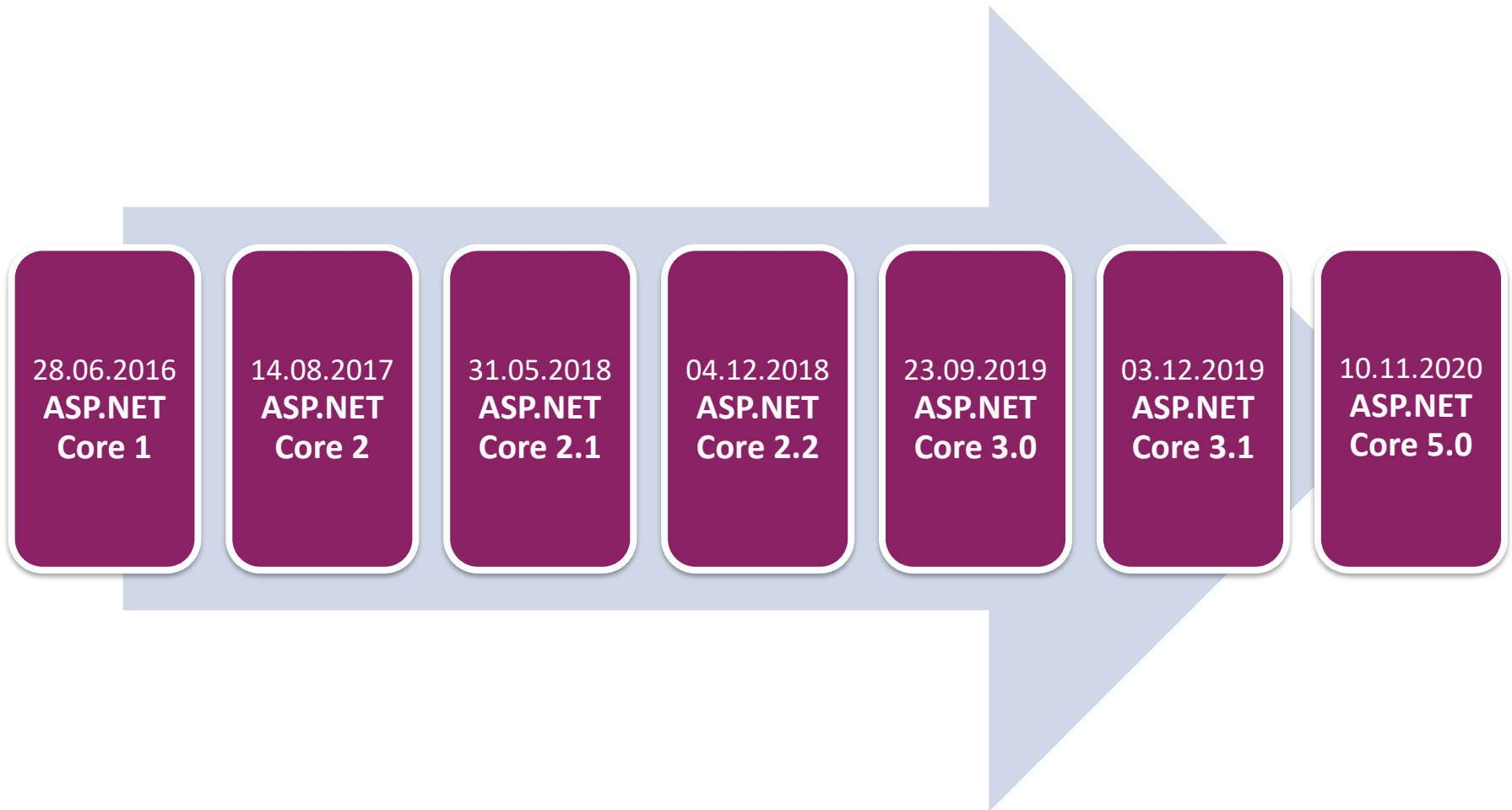
# NEU in ASP.NET Core 2.0 / 2.1

- Razor Pages
- ASP.NET Core Metapaket
- Runtime Store
- .NET Standard 2.0
- Konfiguration / Logging in DI integriert
- SPA-Vorlagen
- Kestrel / HTTP.sys
- Automatisches Vorkompilieren von Views
- SignalR (ab 2.1)
- Razor Class libraries (ab 2.1)
- HTTPS als Standard (ab 2.1)
- [ApiController], ActionResult<T>
- IHttpConnectionFactory
- MetaPaket Microsoft.AspNetCore.App

- Unterstützung C# 8
- .NET Standard 2.1
- Blazor Server
- gRPC
- JSON-Serialisierung mit System.Text
- Aktualisierung der Vorlagen
- Generischer Host
- HTTP/2
- Endpunktrouting
- Workerdienst

# ASP.NET Core 5.0 Api

# Versionshistorie ASP.NET Core



- 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 5.0 erfordert .NET Core 5.0



- 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



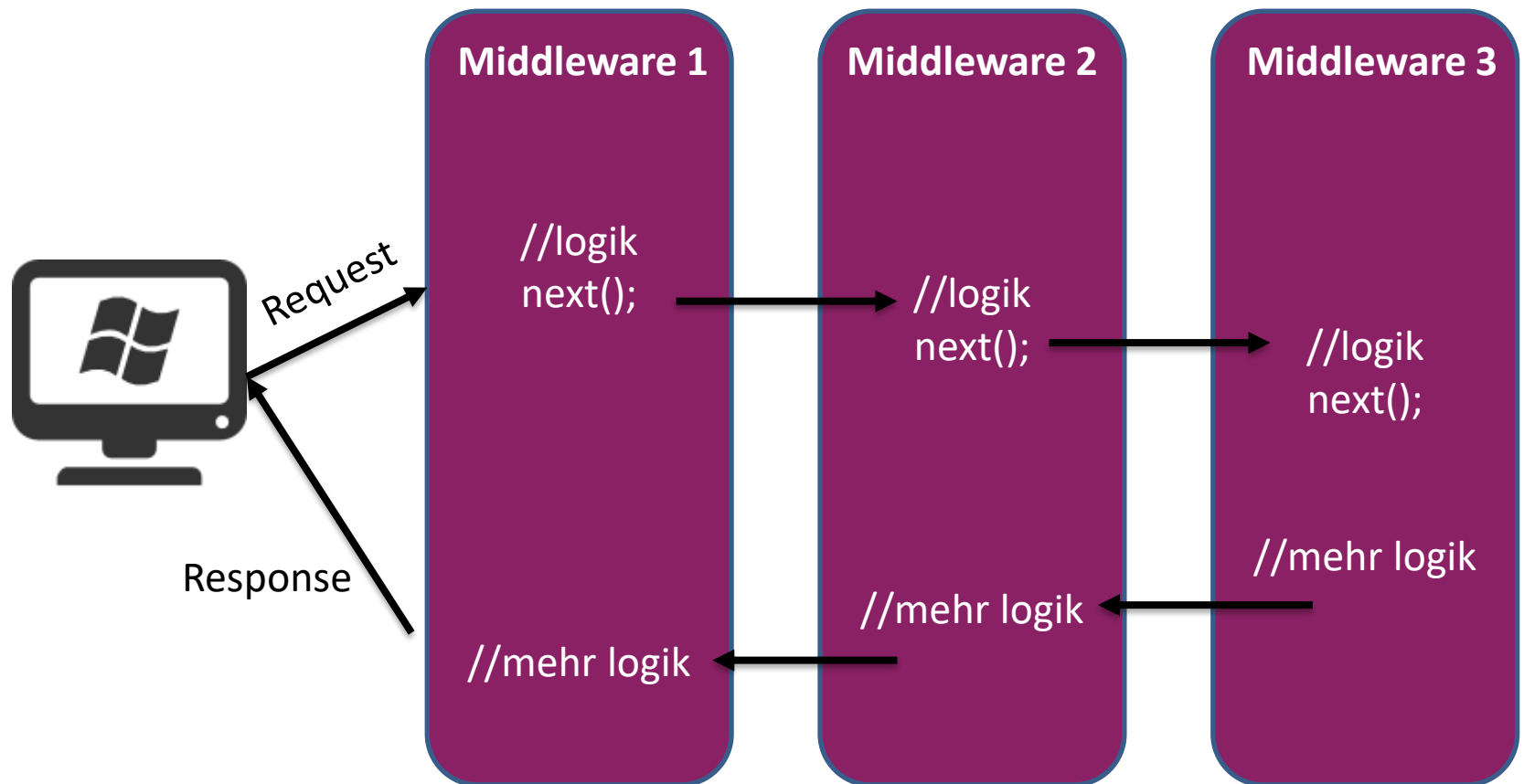
## Projekt SpruchDesTages

# Middleware

- **Program-Klasse**
  - Verantwortlich für die Konfigurierung/Ausführen der Anwendung
- **Startup-Klasse**
  - Einstiegspunkt der Webapplikation
- **ConfigureServices** wird genutzt um Services dem Container hinzuzufügen und zu konfigurieren
- **Configure** wird genutzt um zu spezifizieren wie eine ASP.NET Applikation auf individuelle HTTP-Anfragen zu antworten
- Middlewares um HTTP-Anfrage-Pipeline zu konfigurieren

# 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 in **Configure**-Methode in Startup-Klassen durch Extension-Methoden des **IApplicationBuilder**
- Der IApplicationBuilder 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

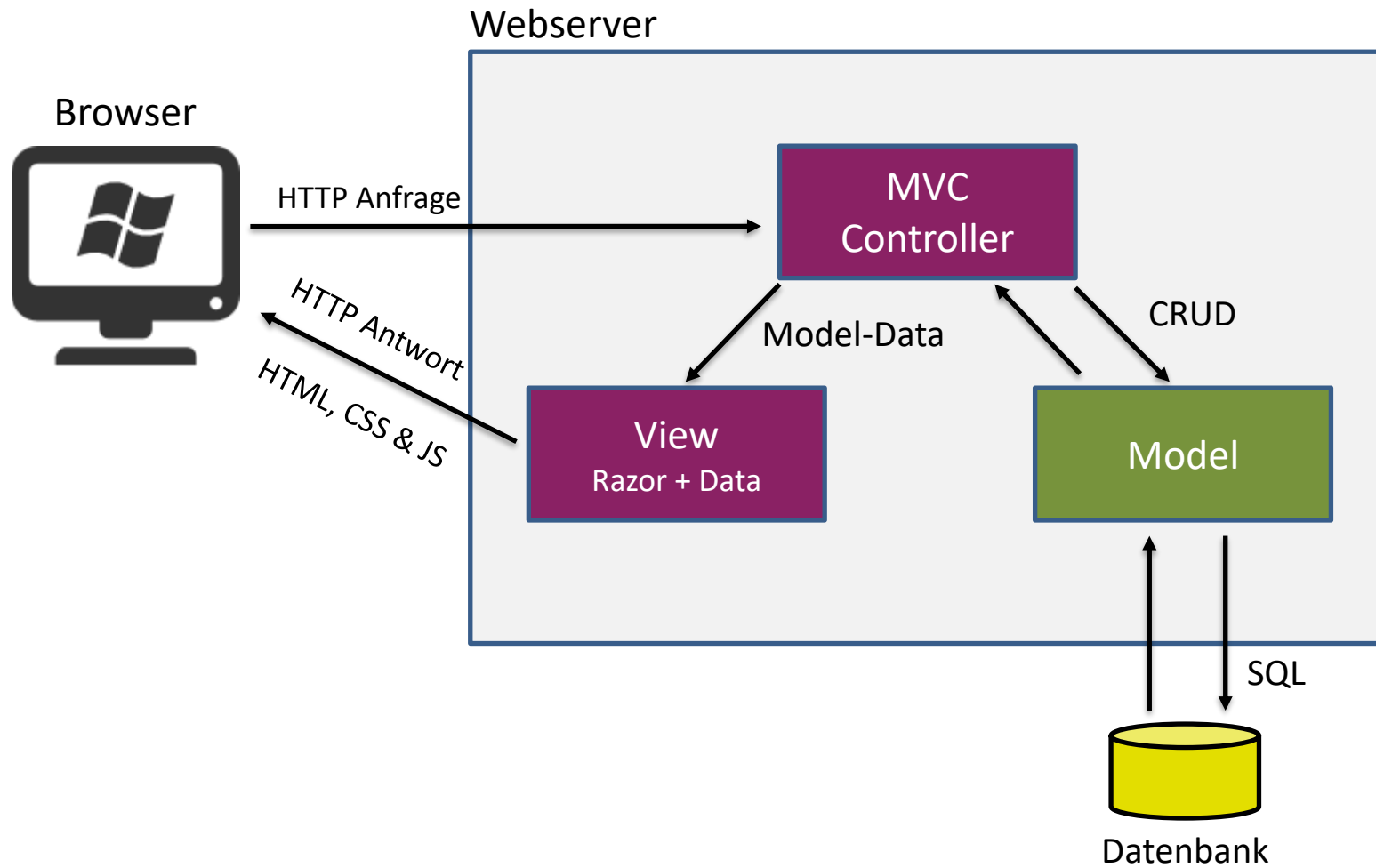


## Demo Middleware

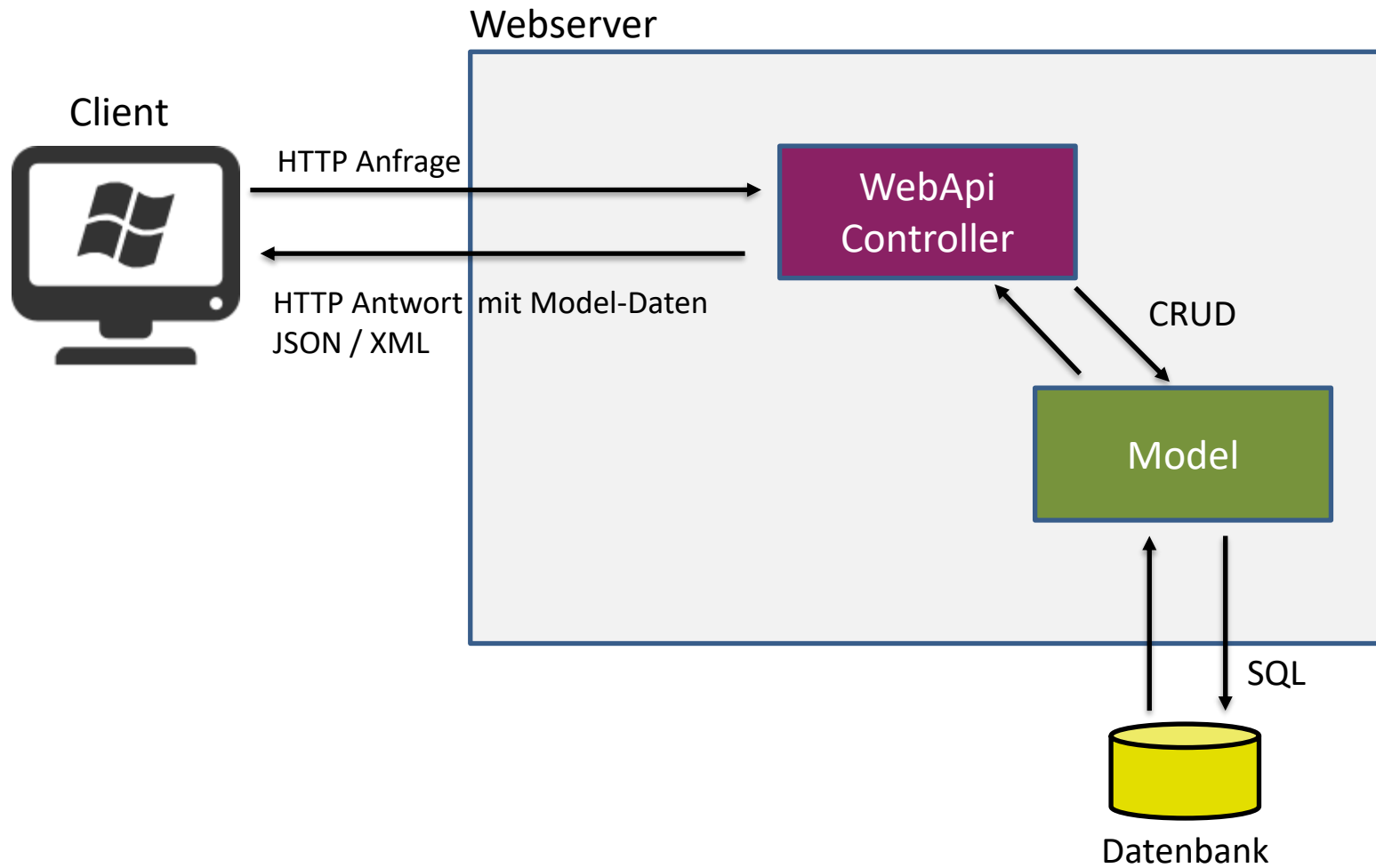


# Controller & Actions

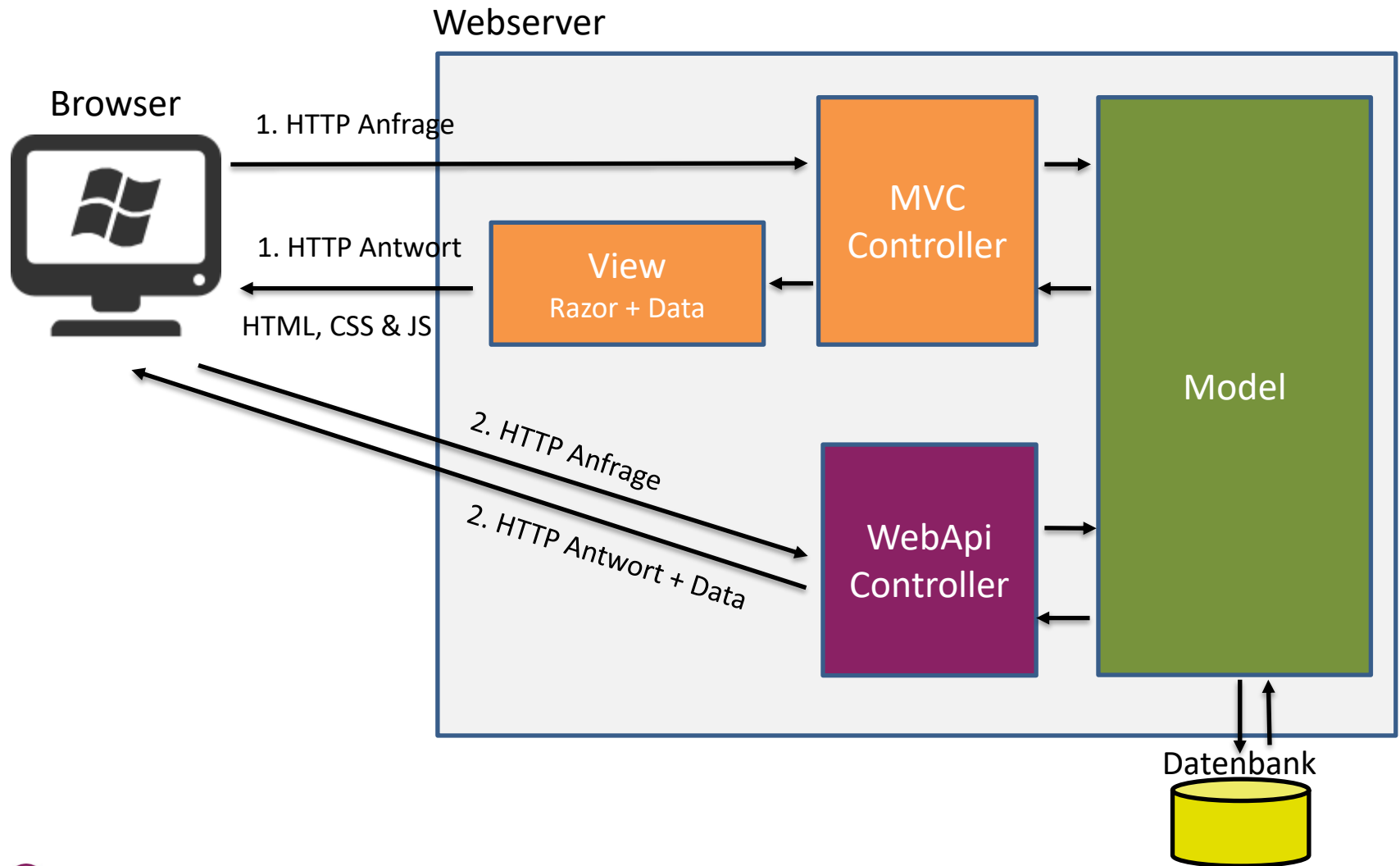
# Request-Response Ablauf MVC



# Request-Response-Ablauf ASP.NET Core 3 Api



# Request-Response Single Page Application



- 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 bzw. ActionResult<T>
  - anderen Typ

# Action Results von IActionResult (Auszug)

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

# Action Results von IActionResult (Auszug)

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



- 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 Ok(autor);
}
```

# 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.

# ASP.NET Models

- Was ist ein Model?
- Erstellung von Models
- Entity Framework

## Eine Klasse





# Model

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

```
public class Autor
{
    public int AutorId { get; set; }
    public string Name { get; set; }
    public string Beschreibung {get;set;}
    public DateTime? Geburtsdatum {get;set;}
}
```

# Exkurs: Entity Framework 6 & Core

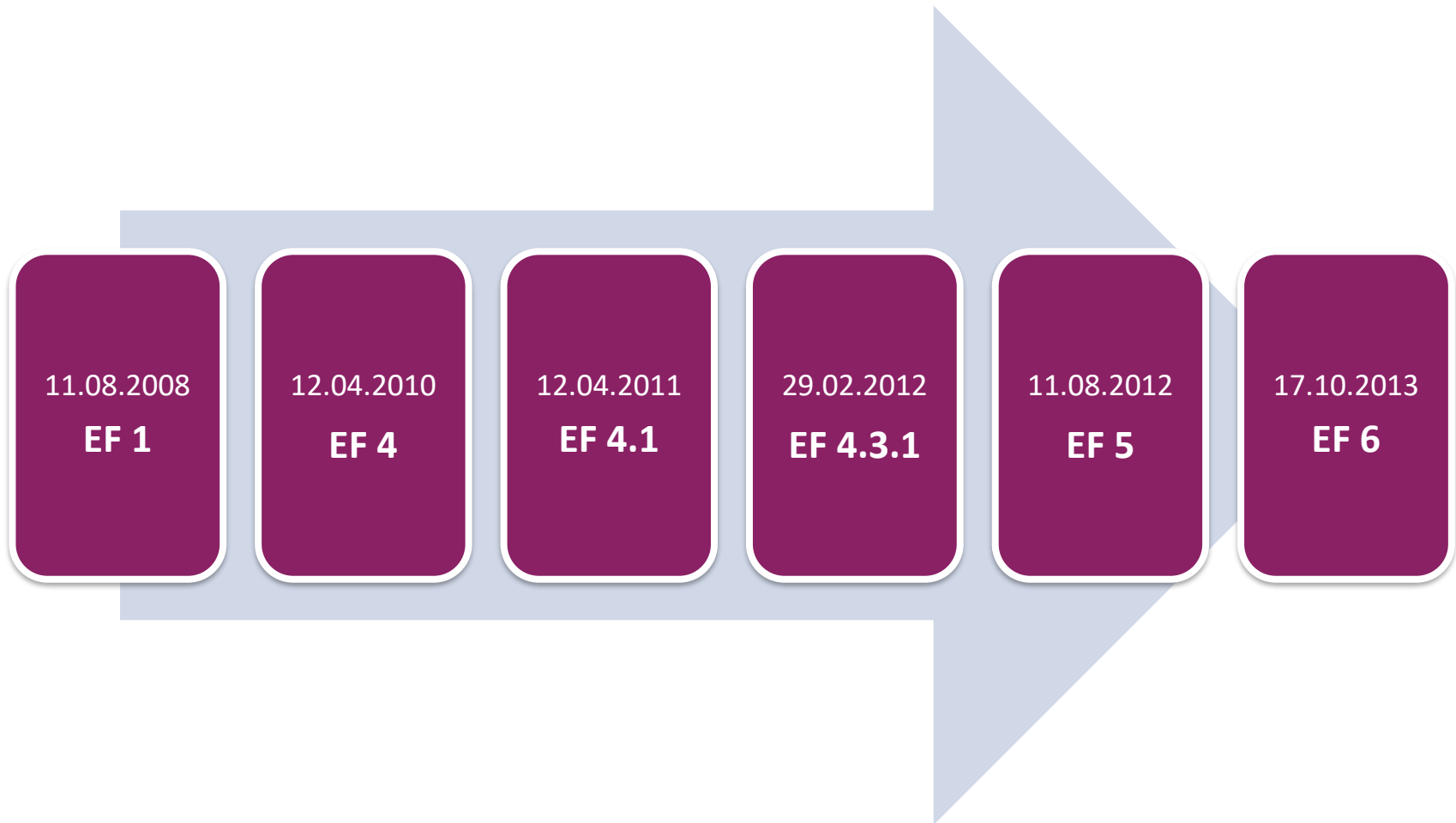
- ORM-Framework

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

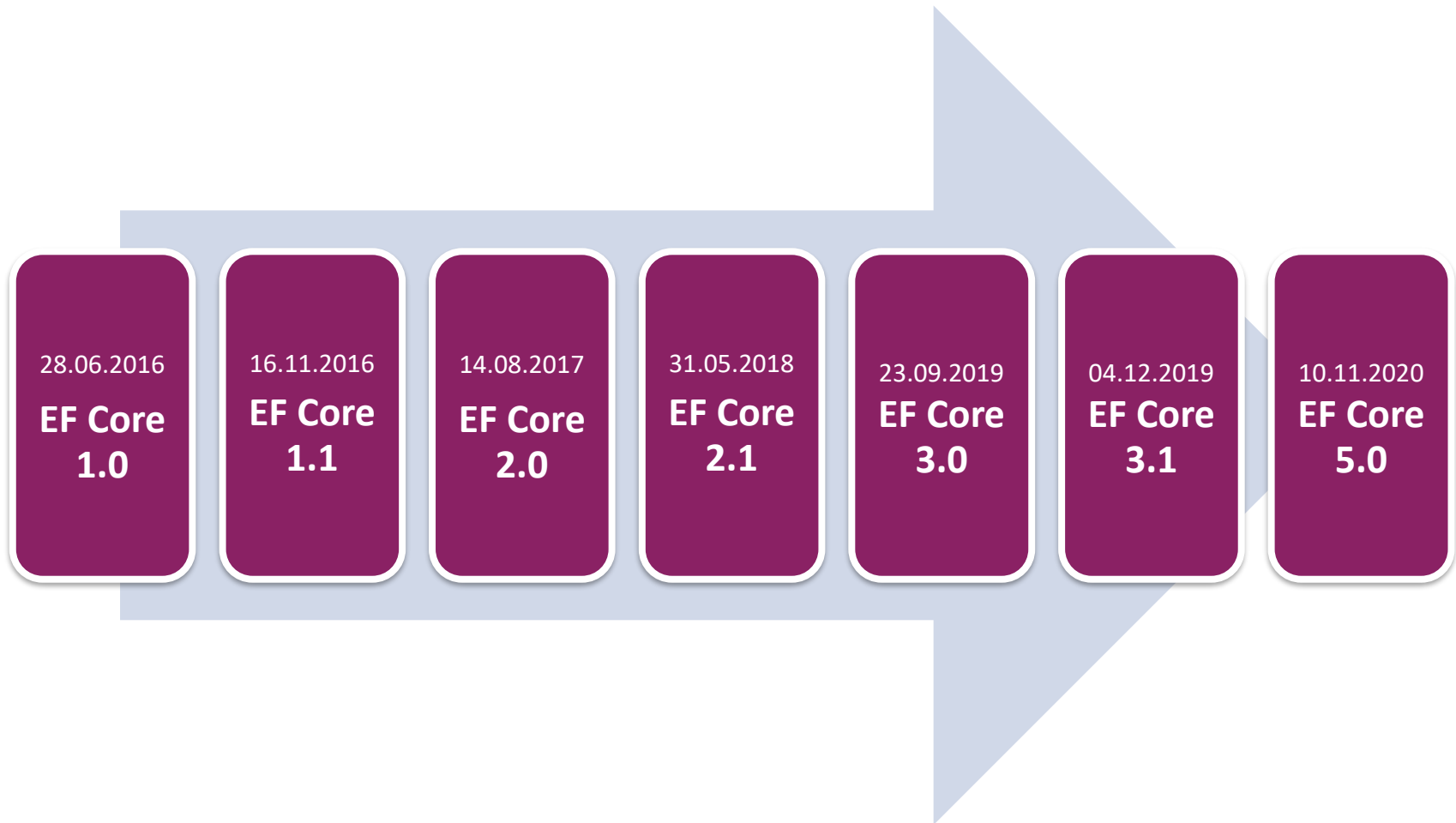


- Seit Version 6.0 Open Source

# Versionshistorie Entity Framework



# Versionshistorie Entity Framework



# Ohne Entity Framework

- Fehleranfällig, Unsicher, Kompliziert

```
string con = "Data Source=.\;Initial Catalog=AutorDB;IntegratedSecurity=True";
string cmd= "INSERT INTO Autor (Name,Vorname) 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

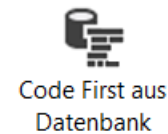
```
using (var db = new AutorDBContext())  
{  
    db.Autoren.Add(new Autor { Name = "Twain", Vorname = "Mark" });  
    db.SaveChanges();  
}
```

- 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 6



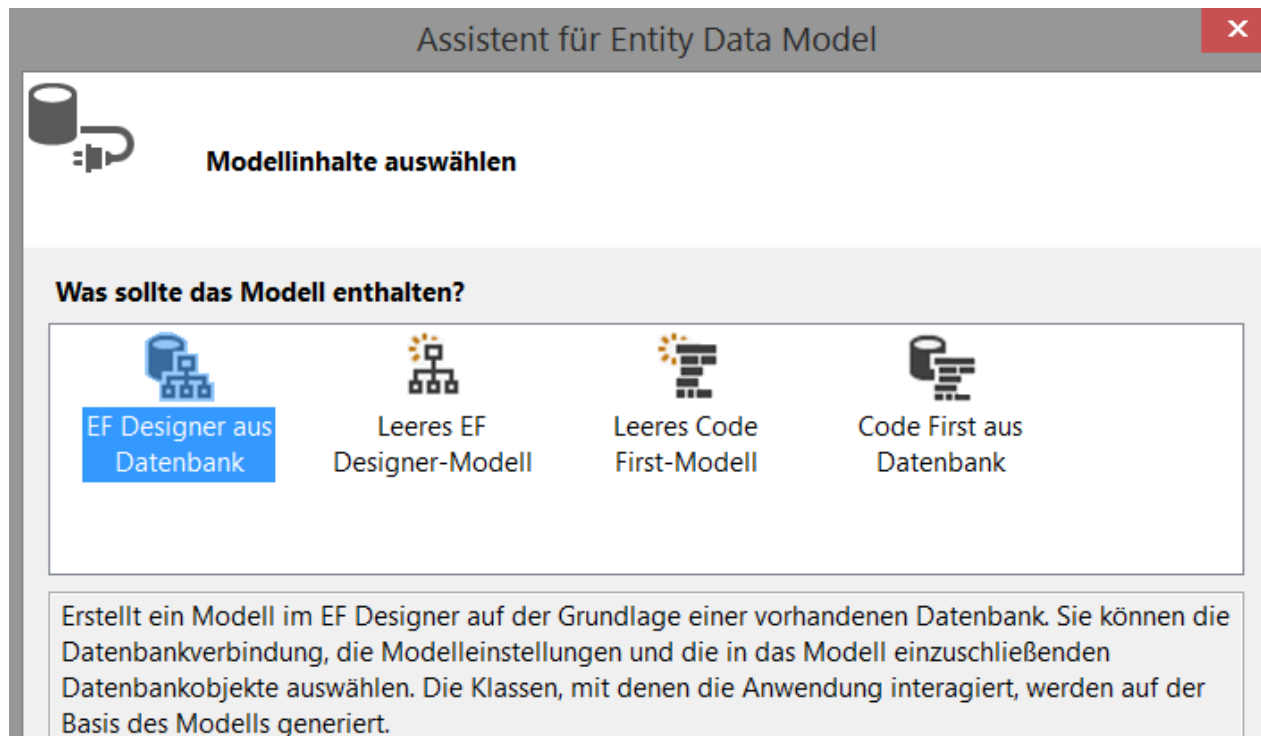
Vorgehensweise	Datenbank vorhanden	Beschreibung
<b>Code First</b>	<b>Nein</b>	Objekte und deren Beziehungen werden in Klassen beschrieben. Daraus wird dann mit dem EF die DB erstellt
<b>Model First</b>	<b>Nein</b>	Grafisches Datenmodell aus dem die DB erstellt werden kann
<b>Database First</b>	<b>Ja</b>	Entity Data Model Designer entwickelt aus einer DB ein Objekt-Modell, das grafisch im Designer dargestellt wird
<b>Reverse Engineering Code First</b>	<b>Ja</b>	Generierung von Entity Framework Klassen aus der DB

# 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

- EF 6 <= Visual Studio 2010
- NuGet-Paket-Manager
  - Ab VS 2012 standardmäßig installiert
  - Als Erweiterung für VS 2010 verfügbar
- NuGet-Konsole
  - Install-Package EntityFramework
  - Install-Package EntityFrameworkCore

- Modell Generierungsoption > EF Designer aus Datenbank



# Verbindungseigenschaften

Verbindungseigenschaften ? x

Geben Sie Informationen zum Verbinden mit der ausgewählten Datenquelle ein, oder klicken Sie auf "Ändern", um eine andere Datenquelle und/oder einen anderen Anbieter auszuwählen.

Datenquelle:  
Microsoft SQL Server (SqlClient) Ändern...

Servername:  
(localdb)\v11.0 Aktualisieren

Beim Server anmelden

☒ Windows-Authentifizierung verwenden

☐ SQL Server-Authentifizierung verwenden

Benutzername:

Kennwort:

☐ Kennwort speichern

Mit Datenbank verbinden

☒ Wählen Sie einen Datenbanknamen, oder geben Sie ihn ein:  
Northwind ▼

☐ Datenbankdatei anhängen:

# Datenbankobjekte

Assistent für Entity Data Model

Wählen Sie Ihre Datenbankobjekte und Einstellungen.

Welche Datenbankobjekte möchten Sie in Ihr Modell einschließen?

- ☒ Tabellen
  - ☒ dbo
    - ☐ Categories
    - ☐ CustomerCustomerDemo
    - ☐ CustomerDemographics
    - ☒ Customers
    - ☐ Employees
    - ☐ EmployeeTerritories
    - ☐ Order Details
    - ☒ Orders
    - ☐ Products
    - ☐ Region
    - ☐ Shippers
    - ☐ Suppliers

☒ Generierte Objektnamen in den Singular oder Plural setzen

☒ Fremdschlüsselspalten in das Modell einbeziehen

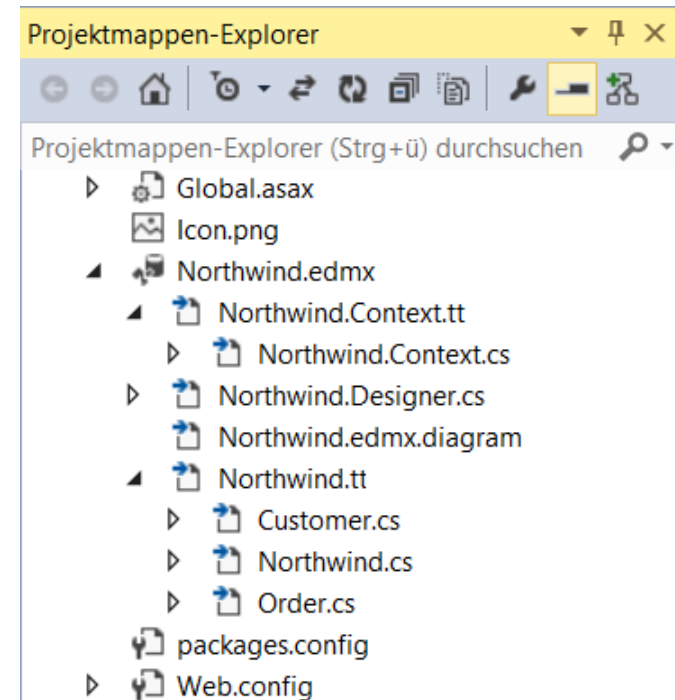
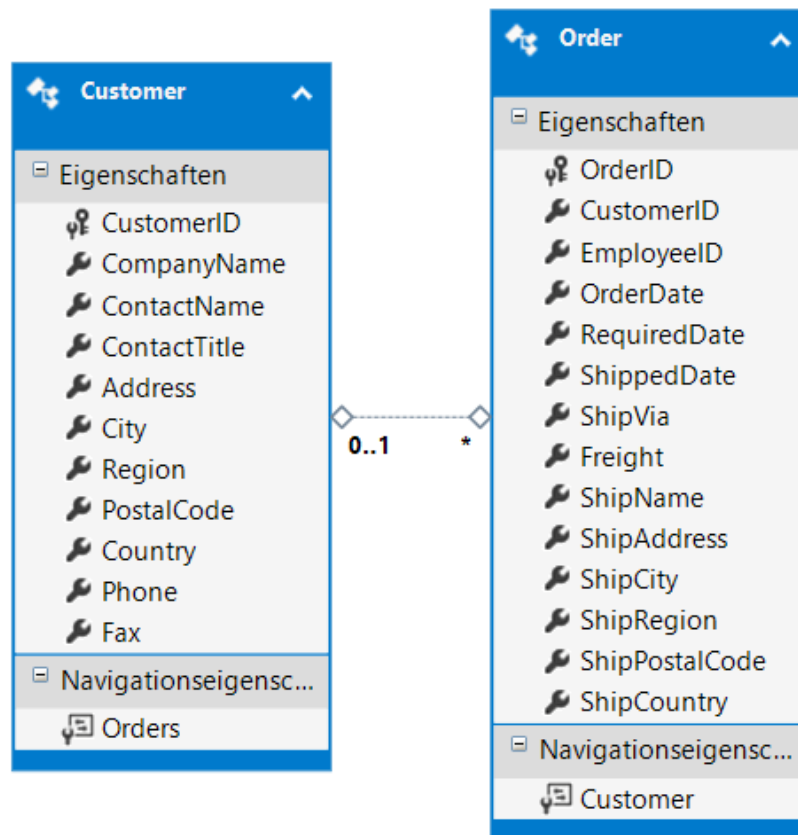
☒ Ausgewählte gespeicherte Prozeduren und Funktionen in das Entitätsmodell importieren

Modellnamespace:

NorthwindModel

< Zurück Weiter > Fertig Abbrechen

# EF Designer EF6





- Problem: Designer-generierter Code wird eventuell überschrieben
- Lösung: Metadaten-Klasse mit Annotation

```
public partial class CustomerMetadata
{
    ...
    [Required]
    [StringLength(59)]
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
    ...
}
```

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer
{
    public string CompanyName { get; set; }
    public string ContactName { get; set; }
}
```

# Code First Entitäten

- Für jede Entity eine Klasse erstellen

```
public class Autor
{
    public int AutorId { get; set; }
    public string Name { get; set; }
    public string Beschreibung { get; set; }
    public virtual List<Spruch> Sprueche { get; set; }
}
```

```
public class Spruch
{
    public int SpruchId { get; set; }
    public string SpruchText { get; set; }
    public int AutorId { get; set; }
    public virtual Autor Autor { 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 SdtDataContext : DbContext
{
    public SdtDataContext(DbContextOptions<SdtDataContext> options) :
        base(options)
    {}

    public DbSet<Autor> Autoren { get; set; }

    public DbSet<Spruch> Sprueche { get; set; }
}
```

# Verwenden EF in Controller

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

```
public class AutorController
{
    private SdtDataContext db = new SdtDataContext();

    public IActionResult GetAutoren()
    {
        var autoren = db.Autoren.ToList();
        return Ok(autoren);
    }
}
```

- 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
  - Befehl **Enable-Migrations** (nur EF 6) erstellt neuen Ordner Migrations und ermöglicht Migrationen für DbContext
  - **Add-Migration** fügt neue Migration ein
  - **Update-Database** synchronisiert Modell mit Datenbank

- Entity Framework stellt Datenbankinitialisierer zur Verfügung
- Erstellen von Klassen, die einen der folgenden Klassen erben
  - **CreateDatabaseIfNotExists**
  - **DropCreateDatabaseWhenModelChanges**
  - **DropCreateDatabaseAlways**
- Überschreiben der **Seed**-Methode um Datenbankinhalte einzufügen
- Methode mit **Database.SetInitializer** registrieren

# EF Core Commands

**Microsoft.EntityFrameworkCore.Tools.DotNet** NuGet für CLI in Visual Studio  
Installation des Tools für CLI: `dotnet tool install --global dotnet-ef`

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

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

# 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

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

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

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

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

- 3 Möglichkeiten der Model-Validierung
  - Built-In-Attribute
  - Benutzerdefinierte Attribute
  - Selbstvalidierendes Model

# Built-In-Validierungsattribute

Validation Attribute	Beschreibung
<b>Compare</b>	Prüft, ob zwei Eigenschaften denselben Wert haben. (z.B. Passwort)
<b>CreditCard</b>	Prüft, ob der zu validierende Wert eine Kreditkartennummer ist
<b>EmailAddress</b>	Prüft auf Email-Format
<b>MaxLength</b>	Prüft auf eine maximale Länge
<b>MinLength</b>	Prüft auf eine minimale Länge
<b>Range</b>	Prüft, ob sich der zu validierende Wert in einem bestimmten Wertebereich befindet
<b>RegularExpression</b>	Validiert die Eigenschaft mit regulären Ausdrücken
<b>Required</b>	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([FromBody] Autor autor)
{
    if(!ModelState.IsValid)
    {
        ModelState.AddModelError("Name","So ein Pech auch");
        return BadRequest(ModelState);
    }
    return Ok(autor);
}
```



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

# Selbstvalidierendes Model

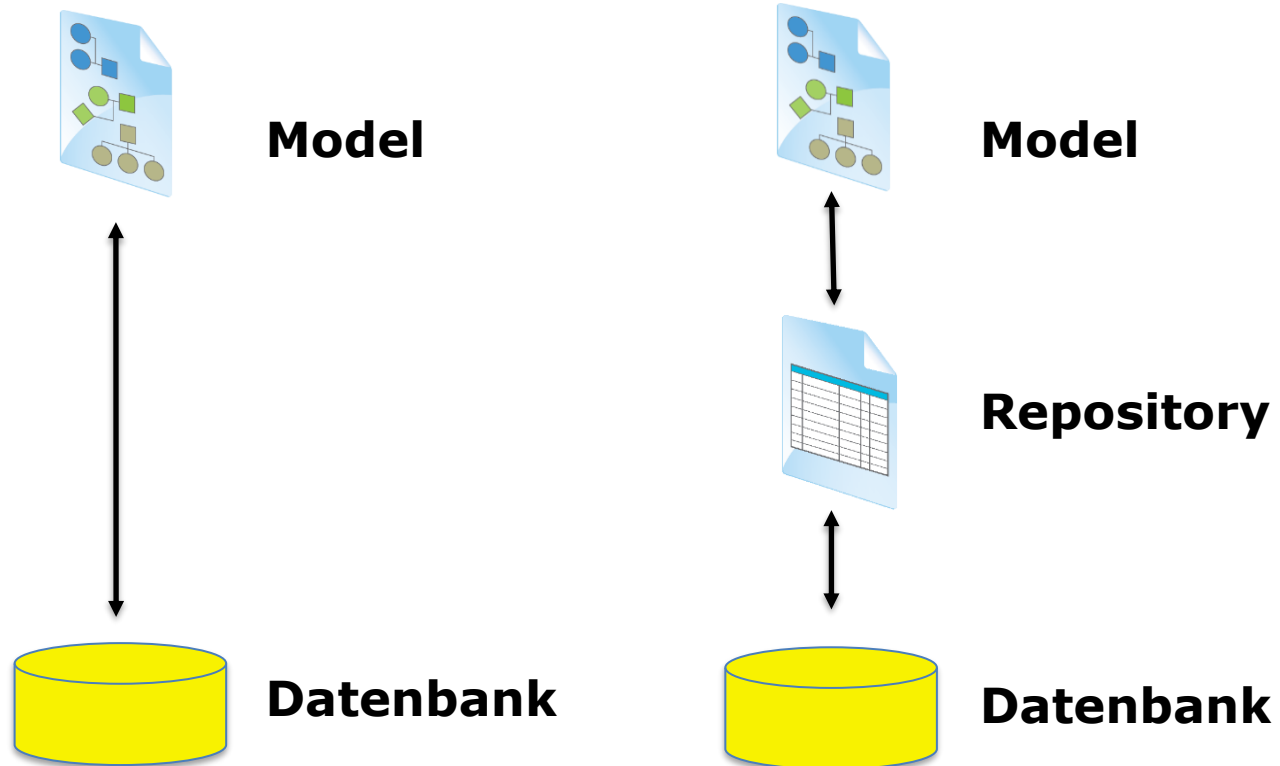
```
public class Teilnehmer : IValidateObject
{
    ...
    [StringLength(50)]
    //[NoAdmin]
    public string Nachname { get; set; }
    ...
}

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;
}
```

# Repository

# Repository

- Separation of Concerns
- Lose Kopplung



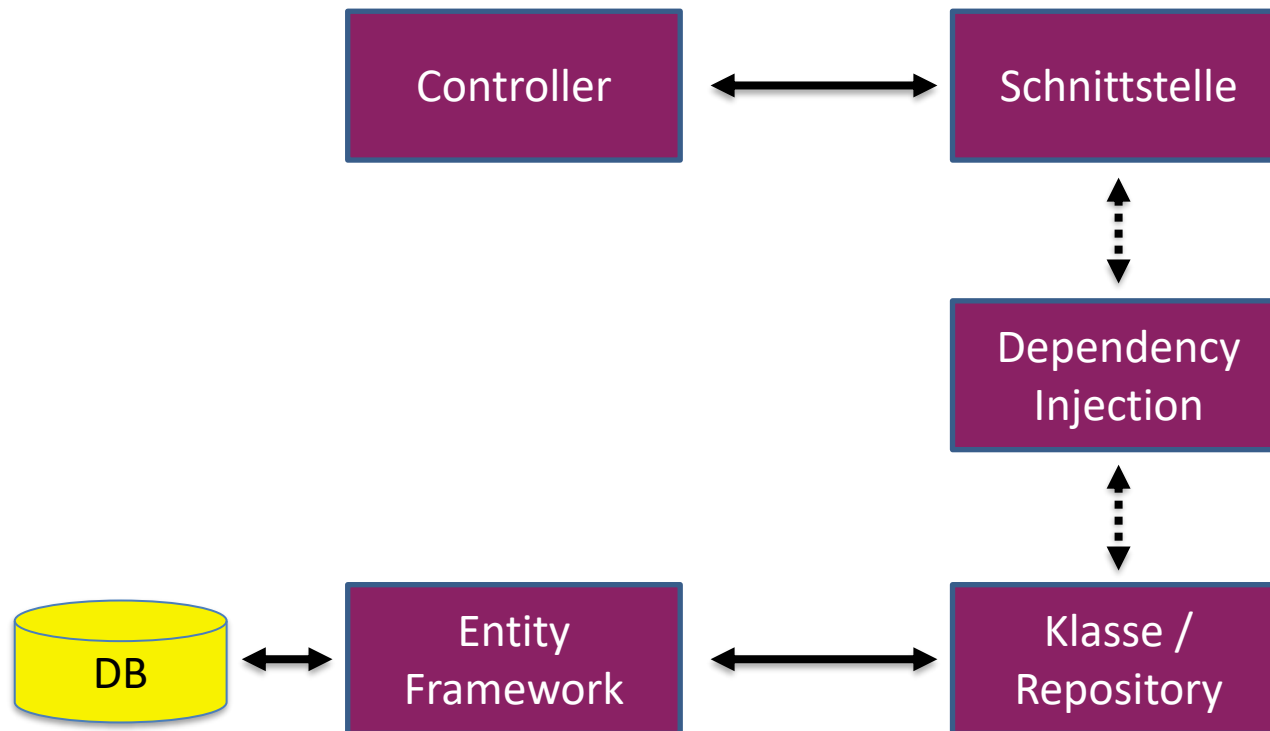
# Dependency Injection

IoC

# IoC / Dependency Injection

- *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 AutorController : ControllerBase
{
    private readonly IAutorRepository _autorRepository;

    protected AutorController(IAutorRepository autorRepository)
    {
        _autorRepository = autorRepository;
    }
}
```

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

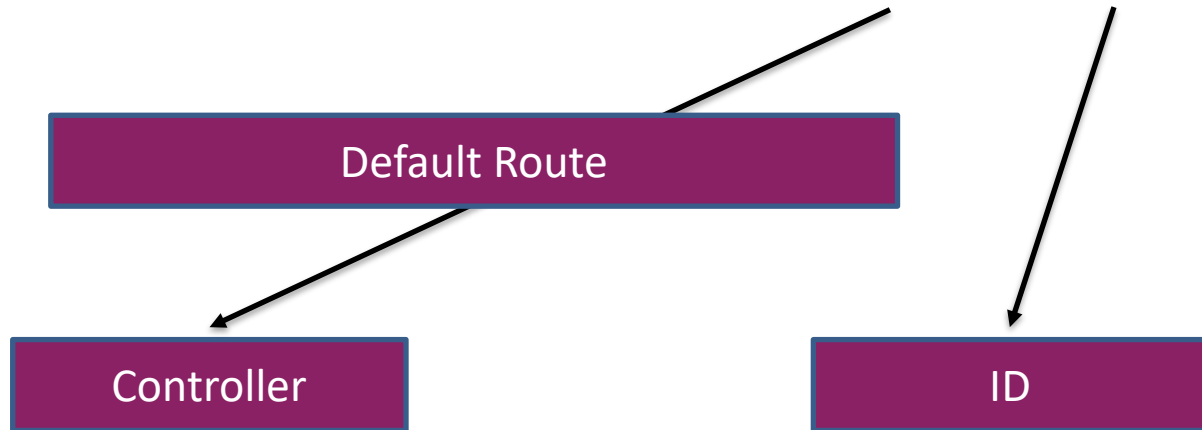
- ConfigureServices-Methode in Startup verantwortlich für die Bereitstellung der Services

Scope	Bedeutung	Erläuterung
<b>Transient</b>	<b>vorübergehend</b>	Eine neue Instanz des Typs wird bei jeder Anforderung des Typs verwendet.
<b>Scoped</b>	<b>bereichsbezogen</b>	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
<b>Singleton</b>	<b>übergreifend</b>	Eine Instanz des Typs wird ein Mal erstellt und von allen nachfolgenden Anforderungen für diesen Typ verwendet.

# 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

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

# Attribut Routing Beispiel

```
[Route("api/autors")]
public class AutorsController : ControllerBase
{
    [HttpGet] // z.B. ./api/autors
    public IActionResult Get() { return Ok(autoren); }

    [HttpGet("{id}")] // Opt. Para. z.B. /api/autors/5
    public IActionResult Get(int id) { return Ok(autor);}

    [HttpGet("autorname/{name}", Name = "GetAutorByName")]
    // z.B. /api/autors/autorname/Albert%20Einstein
    public IActionResult GetAutorByName(string name)
    { return Ok(autor); }
}
```



# 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 <b>DateTime</b> 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 AutorBeschreibungConstraint : IRouteConstraint
{
    private readonly string[] _validAutorBeschreibung;
    public AutorBeschreibungConstraint(string options)
    {
        _validAutorBeschreibung = options.Split(',');
    }

    public bool Match(HttpContext httpContext, IRouter route, string routeKey,
                     RouteValueDictionary values, RouteDirection routeDirection)
    {
        if (values.TryGetValue(routeKey, out var value) && value != null)
        {
            return _validAutorBeschreibung.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("autorBeschreibung",typeof(AutorConstraint));
        ....
    })
}
```

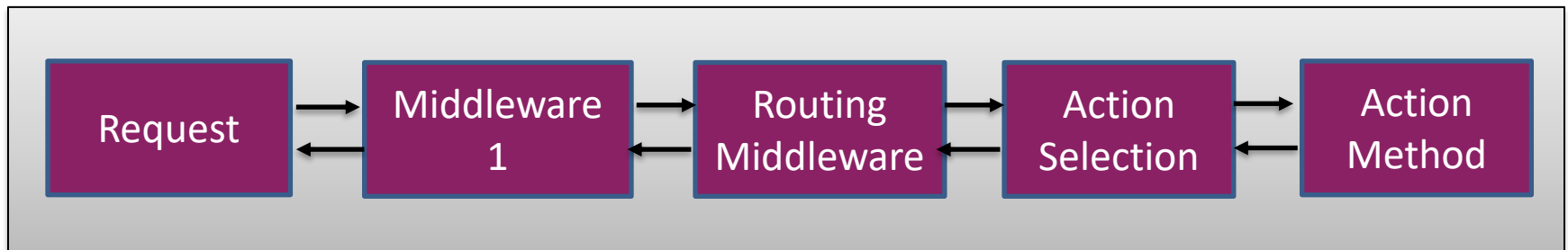
## 3. Action Attribute

```
[Route(„autorbeschreibung/{beschreibung:autorBeschreibung(Dozent,IT-Trainer)}")]
public IActionResult GetAutorByDescription (string beschreibung)
{
    var autors = db.Autoren.Where(c => c.Beschreibung.Contains(beschreibung));
    return Ok(autos);
}
```

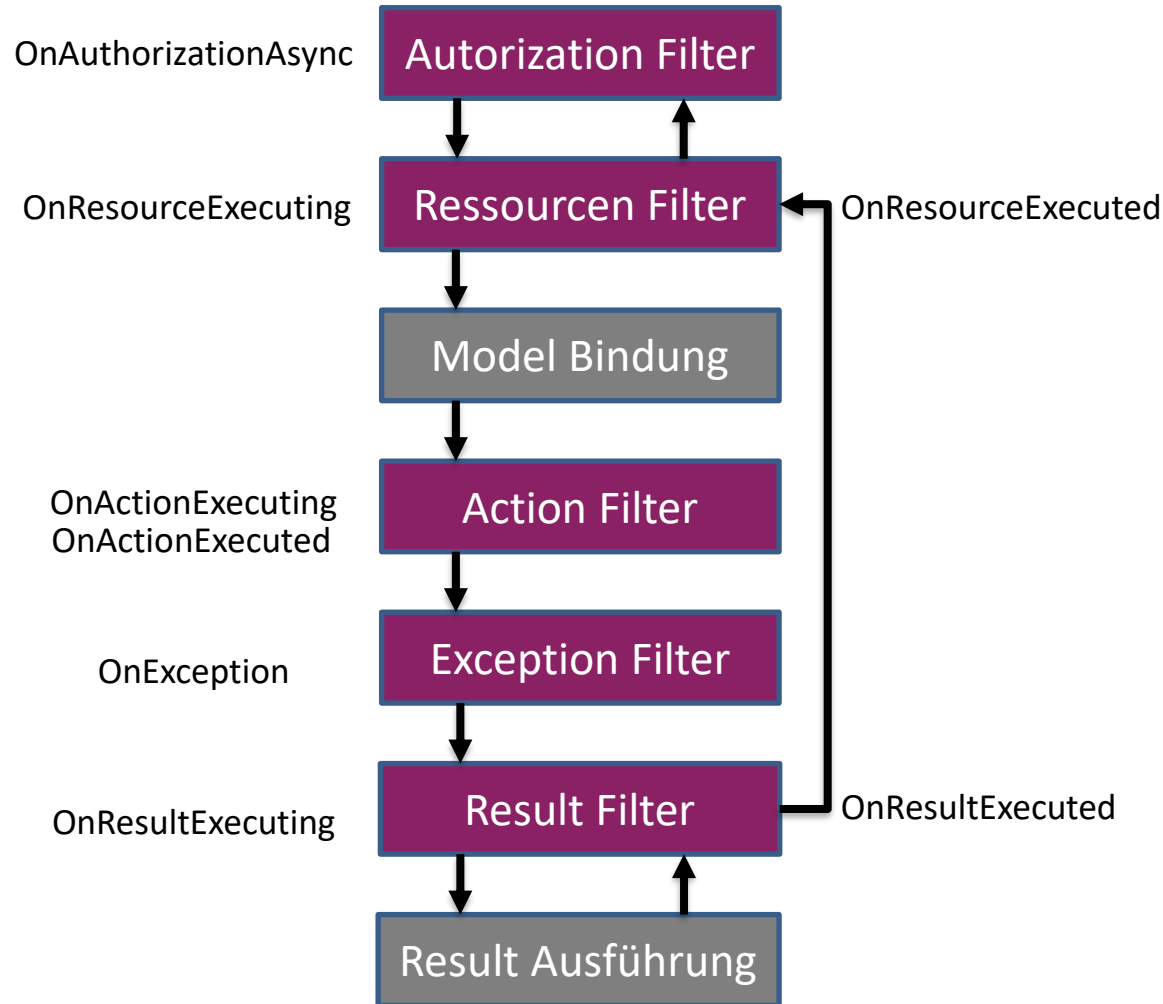
# Filter

# 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
<b>Authorization</b>	IAsyncAuthorizationFilter	OnAuthorizationAsync	Ausführung vor dem Beginn einer Aktion zwecks Autorisierung
<b>Resource</b>	IResourceFilter/ IAsyncResourceFilter	OnResourceExecuting OnResourceExecuted	Werden am Anfang/Ende einer Anforderung aktiv, zum Bearbeiten einer Anforderung
<b>Action</b>	IActionFilter	OnActionExecuting OnActionExecuted	Ausführung vor/nach einer Aktion
<b>Exception</b>	IExceptionHandler/ IAsyncExceptionHandler	OnException	Ausführung, wenn eine Ausnahme im Controller auftritt
<b>Result</b>	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");
    }
}
```

# DEMO

„Filter – SampleActionFilter,  
TimeAttribute, TimeReloaded“

# 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 = "http://localhost/api/sprueche/spruchdestages";
    var client = new HttpClient();
    var response = await client.GetAsync(url);

    response.EnsureSuccessStatusCode();

    var spruch = response.Content.ReadAsAsync<dynamic>().Result;

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

    Console.ReadLine()
}
```

# DEMO

„Http-Client via  
Konsolenanwendung“

# Media Type Formatter

# Media Type Formatter

- Primär unterstützt ASP.NET Core JSON
- Möglichkeit eigene Formate bereitzustellen
- XML-Unterstützung muss manuell hinzugefügt werden via NuGet-Paket „Microsoft.AspNetCore.Mvc.Formatters.Xml“
- 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 MVC-Optionen ConfigureServices
  - `services.AddControllers(options => {  
    options.OutputFormatters.Add(new  
        CsvSpruchDesTagesFormatter());  
});`

# 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
  - ConfigureServices -> services.AddCors()
  - Configure -> app.UseCors();
  - Annotierung der gewünschten Methoden/Controller mit [EnableCors] oder Global

# DEMO

CORS

# Tracing / Logging

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

# 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
}
```

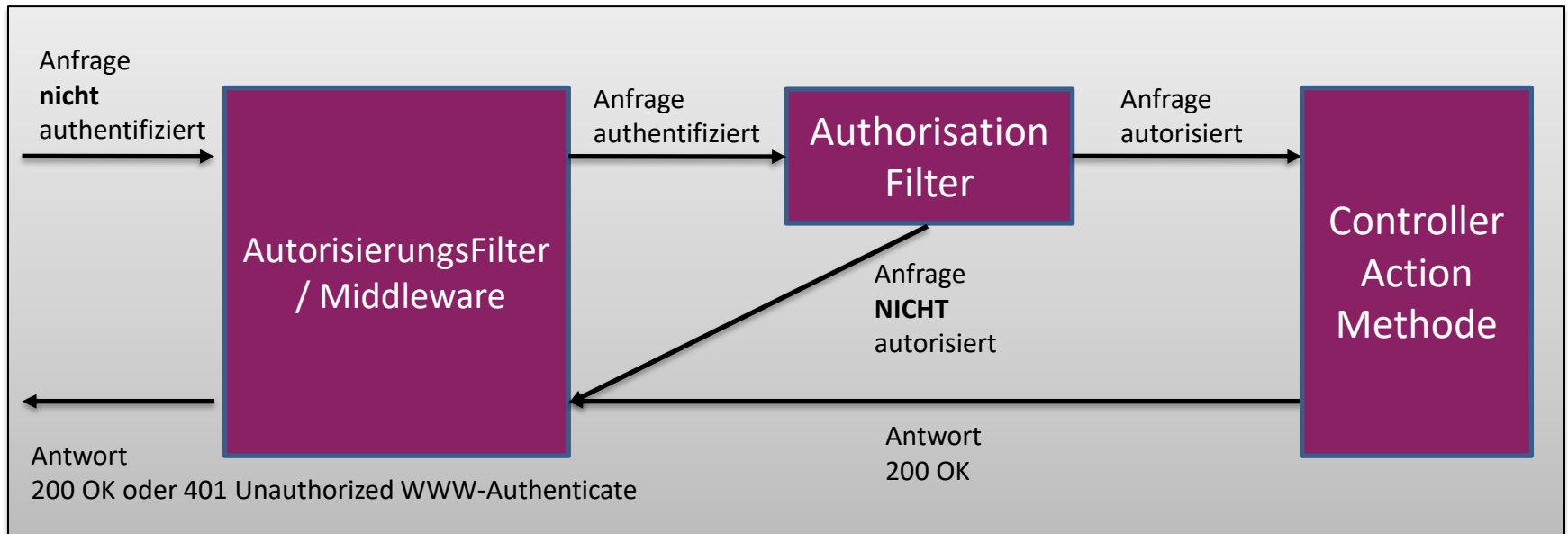
# Authentifizierung / Autorisierung

- 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)

# Basic-Authentifizierung

- Klassiker unter den Authentifizierungsmechanismen
- Sollte **immer** mit SSL genutzt werden
- 1996 eingeführt von RFC 1945 als Teil des HTTP 1.0
- Besteht aus einem Authorisations-Header mit Base64 codiertem Username und Passwort getrennt durch Doppelpunkt
  - Authorization: Basic YWxpQGV4YW1wbGUuY29tOINhb
- Im Falle eines falschen oder fehlenden Credentials antwortet der Server mit eine 401 Not Authorized und fügt ein WWW-Authenticate header hinzu
  - WWW-Authenticate: Basic realm="Test"

# Authentifizierungs-/Autorisierungsablauf



# DEMO

„Basic-Authentifizierung in Client &  
WebFrontend“

# JWT - Bearer-Authentifizierung

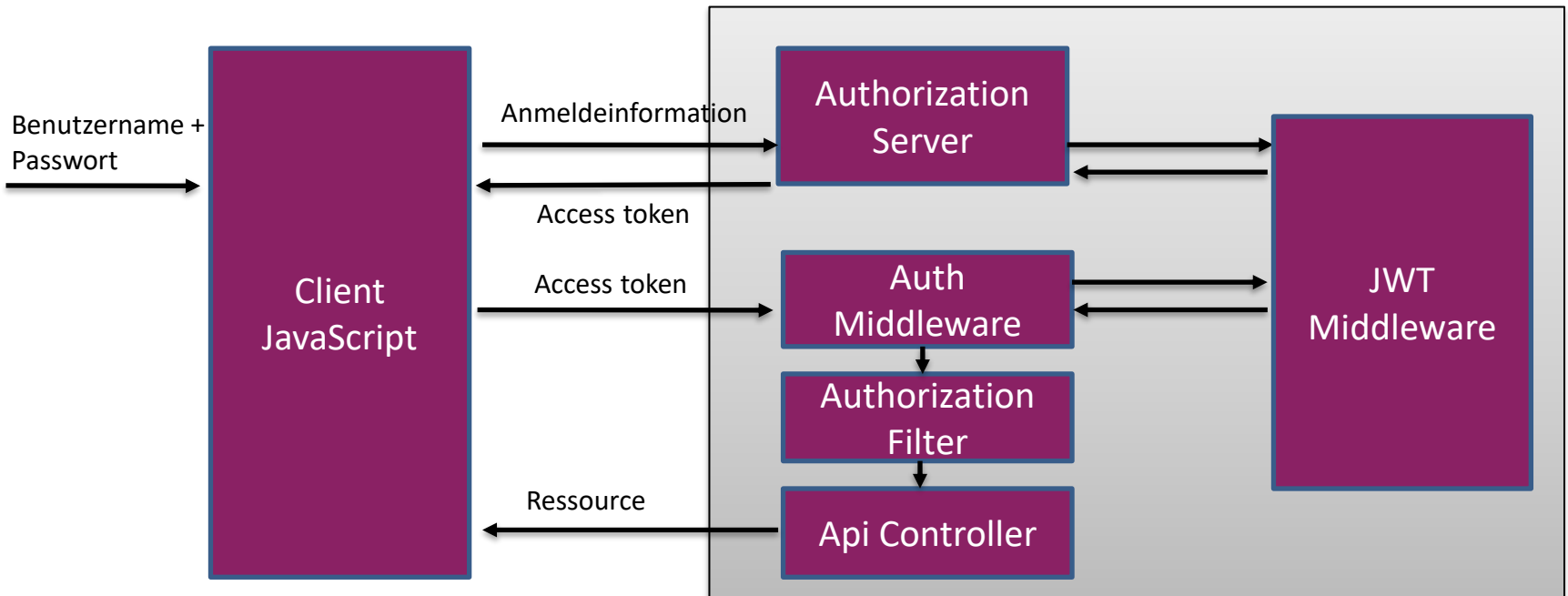
- Definition
  - *A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession) (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

# DEMO

„Bearer-Authentifizierung Web- und  
Client Anwendung“

# Versionierung

- Wie werden verschiedene Versionen der Web API verwaltet?
- Philosophische Frage: Sollen Web API versioniert werden?
- Vorgehensweise:
  - NuGet-Paket `Microsoft.AspNetCore.Mvc.Versioning`
  - `ConfigureServices -> AddApiVersioning()`
  - Attributierung der Controller mit `[ApiVersion()]`
- Verschiedene Möglichkeiten der Versionierung
  - `QueryString` Parameter
  - `Url Path` Segment
  - Header

# 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)
- Seit April 2013: jQuery 2.0 (aber kein IE 6-8 Support!)

- 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

