ASP.NET Core 9 Api

Ali Ilci

B.A.

Einleitung

- 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

Was ist ein WebService?

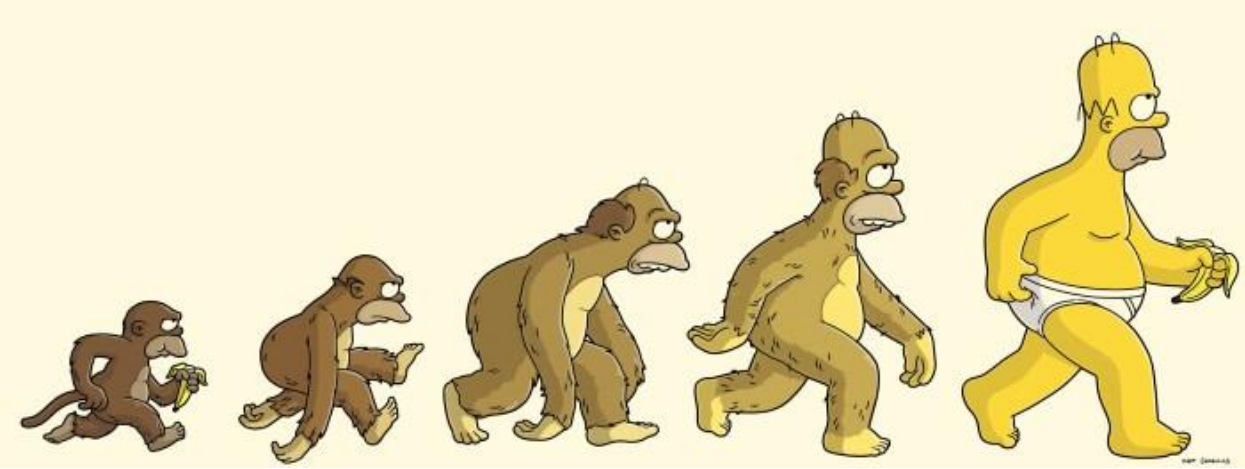
Ein **Webservice** oder **Webdienst** ist eine Softwareanwendung, die über ein Netzwerk für die direkte Maschine-zu-Maschine-Interaktion bereitgestellt wird. Jeder Webservice besitzt einen Uniform Resource Identifier (URI), über den er eindeutig identifizierbar ist, sowie eine Schnittstellenbeschreibung 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

Entwicklung Web Services

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

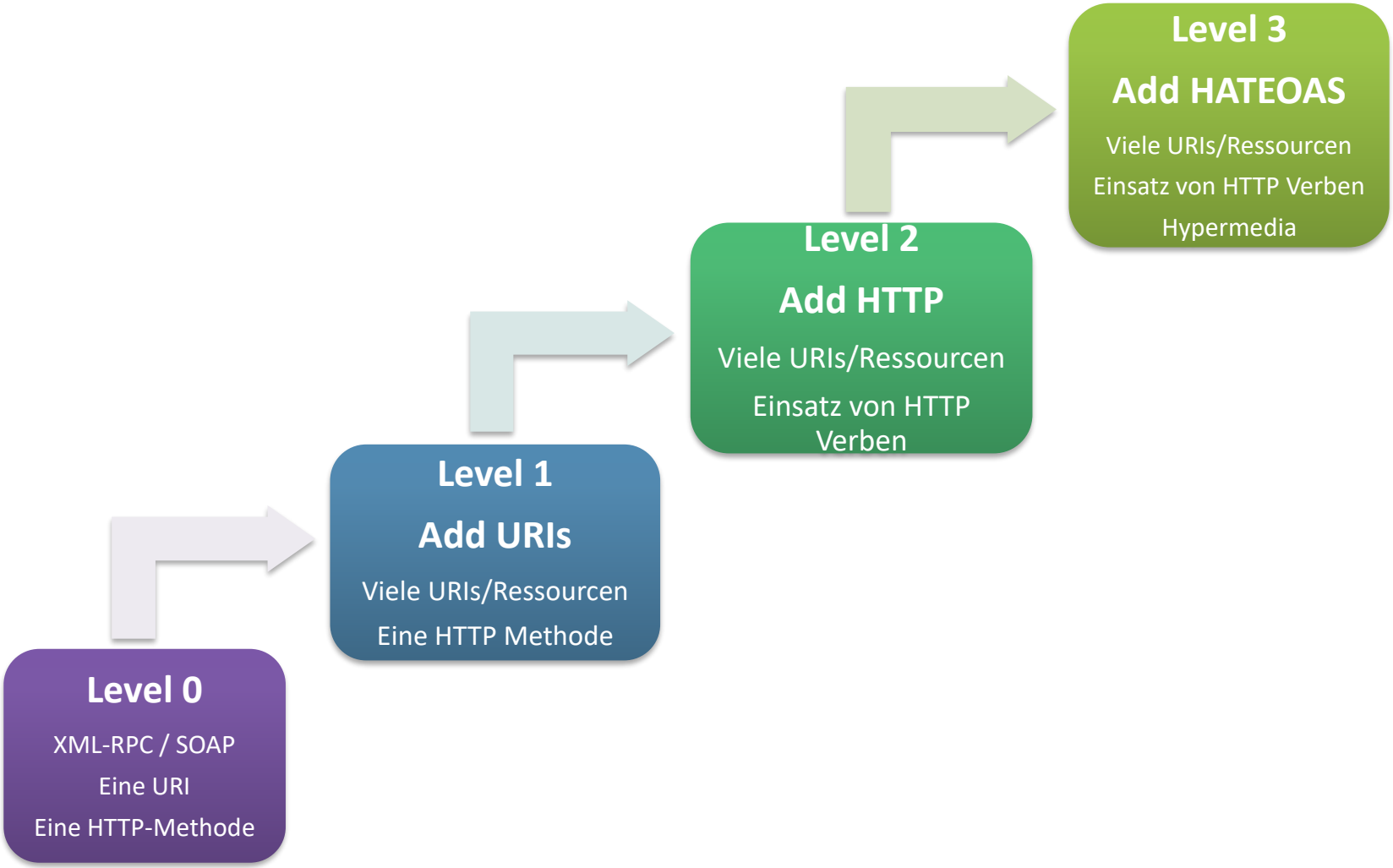
REST - Begrifflichkeiten

- **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/Resourcen 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/Resourcen 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**	PUT

RMM Level 3 – Beispiel Aufgabe

- Viele URIs/Resourcen und Einsatz HTTP-Verben & **H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate

```
<?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

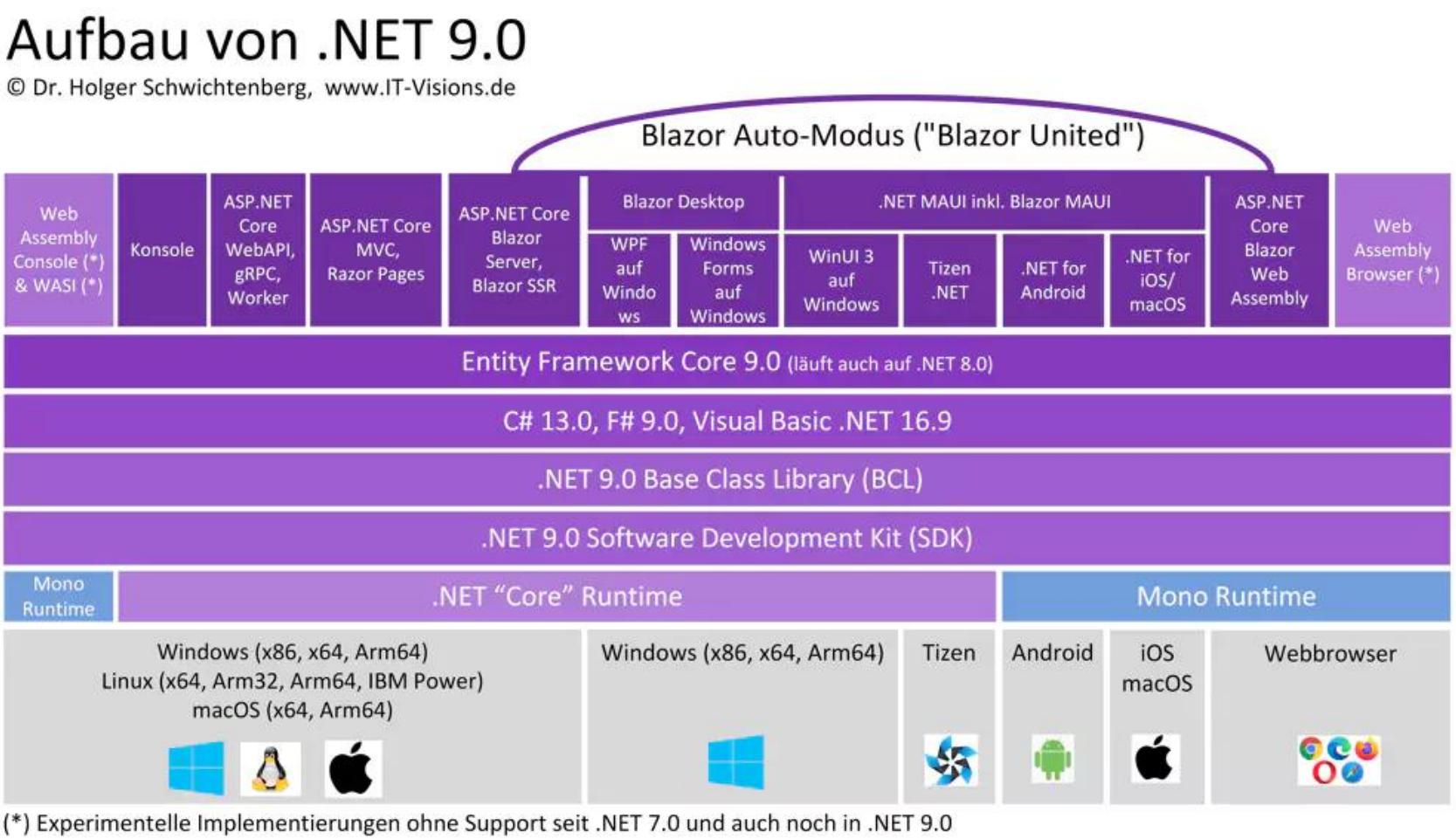
Full .NET Framework	.NET Core
Vollständiges „etablierte“ Framework	Modulare Version des .NET Framework
Nur Windows	Cross-Platform
	Keine Untermenge vom .NET Framework
	Windows, Linux, Mac
	Implementation des .NET Standard

Entscheidung zwischen Full .NET Framework vs. .NET Core

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

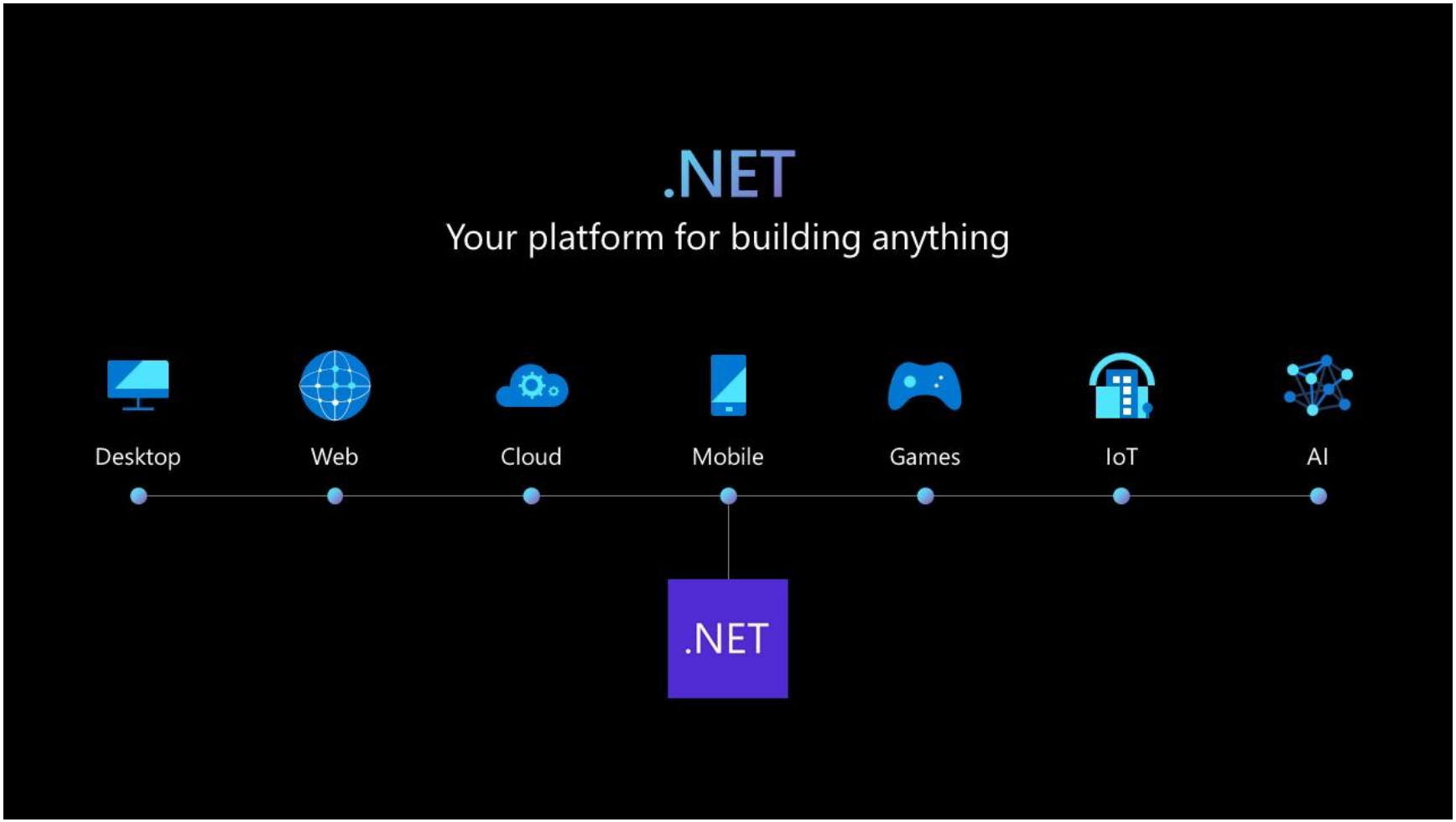
https://docs.microsoft.com/de-de/dotnet/articles/standard/choosing-core-framework-server

.NET Familie 2023/2024

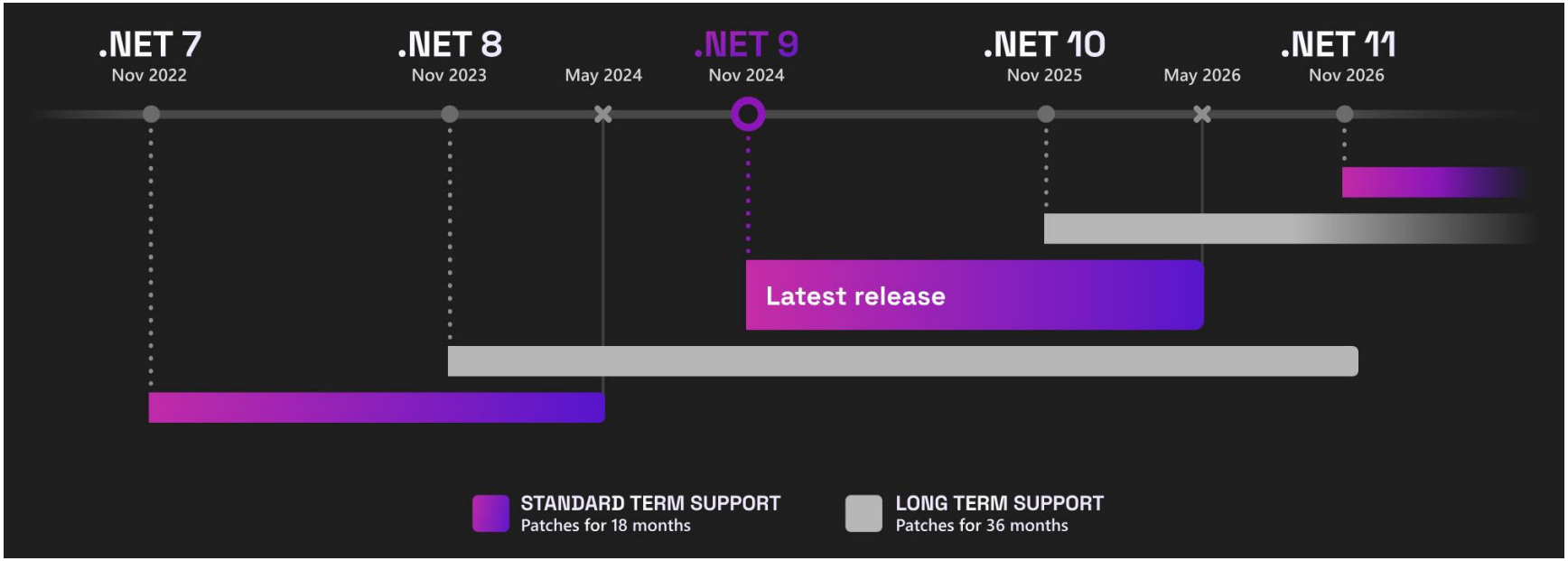


https://www.heise.de/news/Microsoft-NET-9-0-bringt-Breaking-Changes-und-neue-KI-Bibliothek-10031831.html
Holger Schwichtenberg Stand: 13.November 2024

.NET 9 Platform



.NET Schedule



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

.NET CLI

- 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
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
tool	Installation von globalen .NET Core Tools
watch	Dateisystemüberwachung und automatische Projektneuerstellung (ab 2.1)

Themen

Tools

Repository

REST

Model

Web Technologien

DTO

CLI

JQuery

ASP.NET Core Api

Filter

Logging

HttpClient

Authentifizierung

Routing

Formatter

AJAX

Hosting

Validation

Webessentials

Fehlerbehandlung

Actions

Cors

Entity Framework

Action Filter

Dependency Injection

Asynchronität

Security

Controller

Data Annotations

Best Practices

C# 10 / 11 / 12 / 13

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.Collecions.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);
```

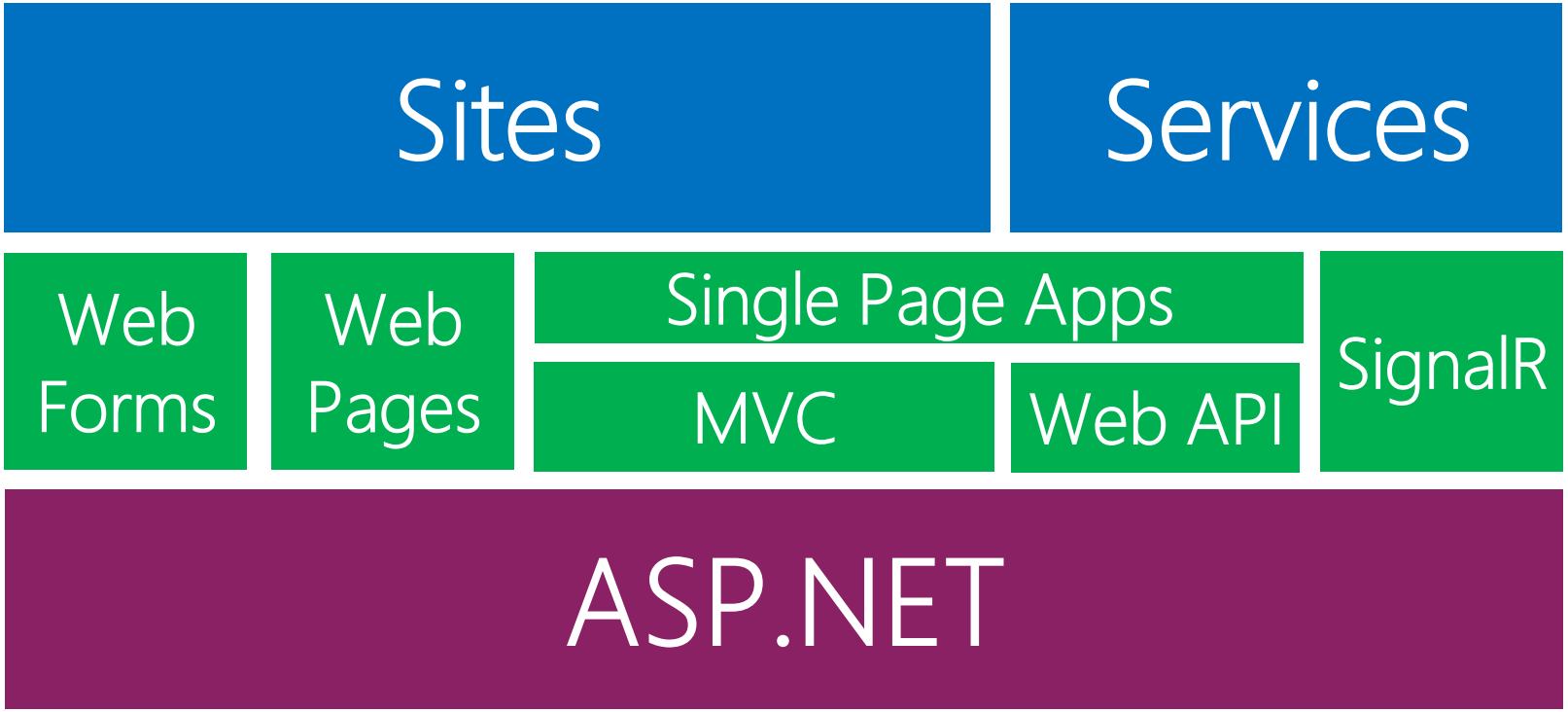
Params Collection C# 13

- Der „params“-Parameter ist nicht mehr nur limitiert auf Array Typen sondern mit jedem Sammlungstyp nutzbar

```
// Vorher
public void MeineMethode(params string[] werte)


//Nachher
public void MeineMethode(params IEnumerable<string>
werte)
```

ASP.NET Core 9

Überblick ASP.NET 4.5



Überblick ASP.NET Core 9

ASP.NET Core 9
(MVC + WebAPI + MinimalApi + Razor Pages + SignalR + gRPC + Blazor (SSR, Server, Blazor WebAssembly)

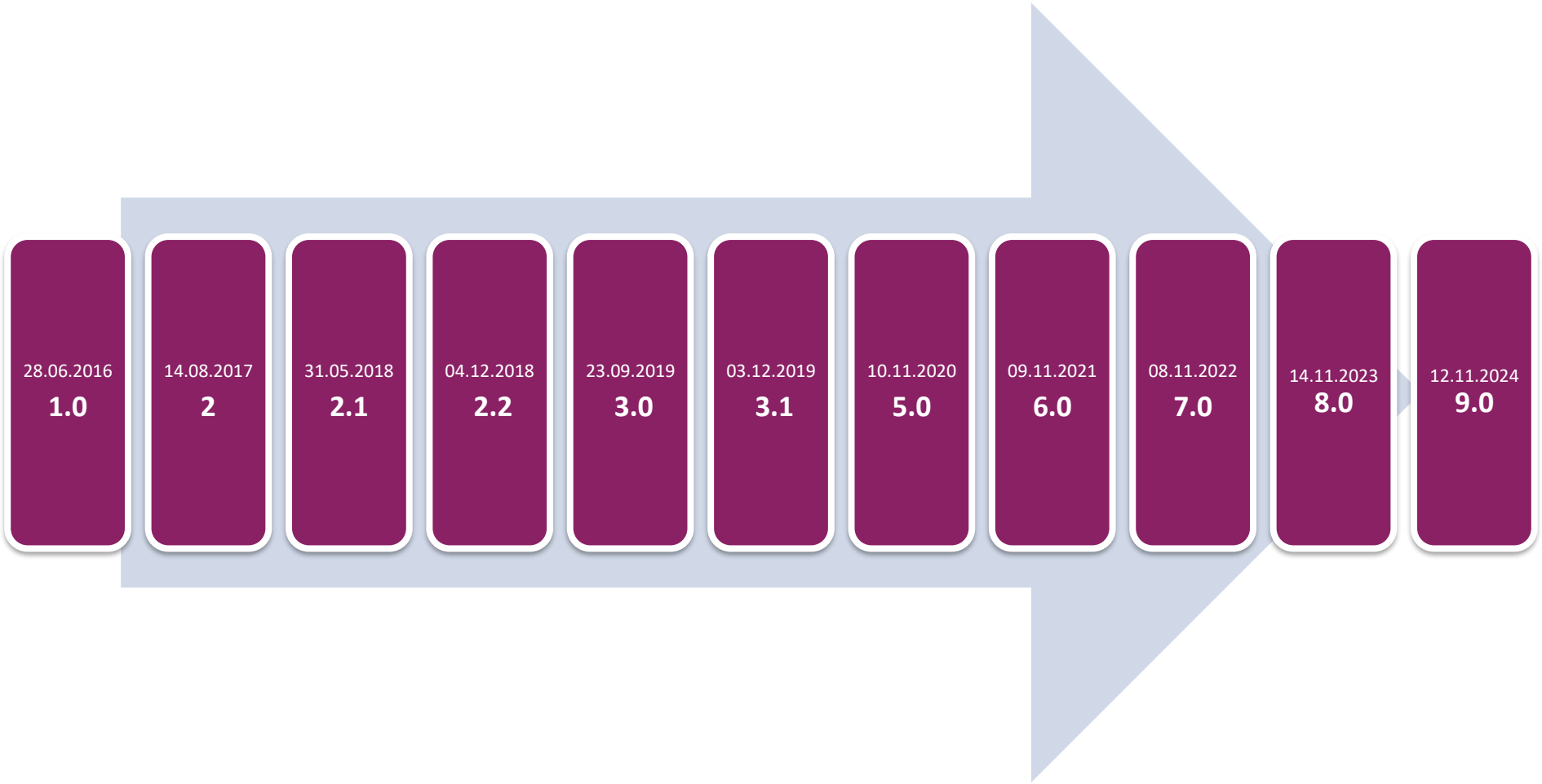
C# 13

.NET 9 Base Class library

.NET Core
Windows & Linux & MacOS

ASP.NET Core 9 Api

Versionshistorie ASP.NET Core Web Api



Entwicklung zu ASP.NET Core 8 WebApi

- 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 9 erfordert .NET Core 9

Features

- 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

Hands On

Projekt Quote of the Day

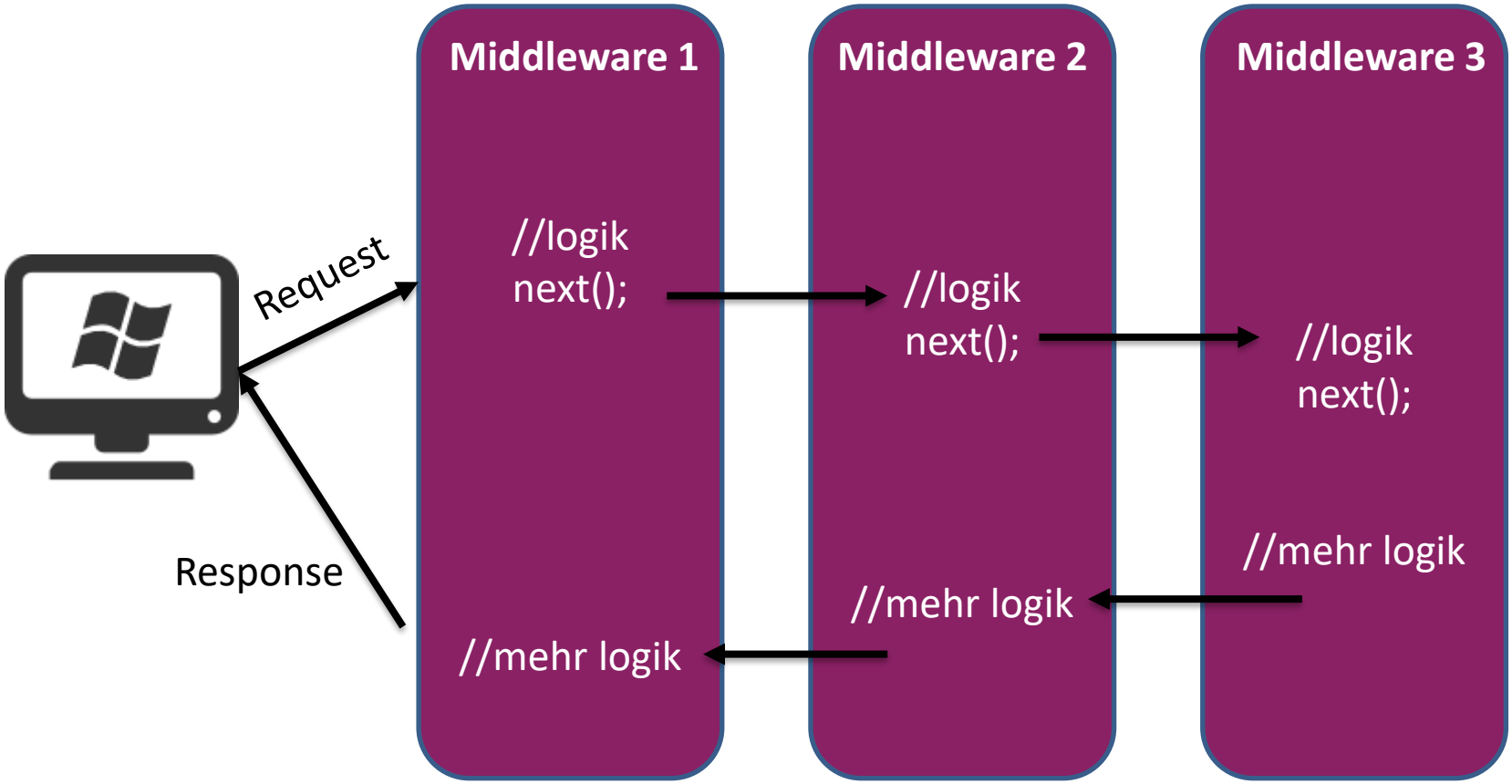
Middleware

ASP.NET Core

- **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

ASP.NET Core Request Pipeline & Middleware

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



Middleware

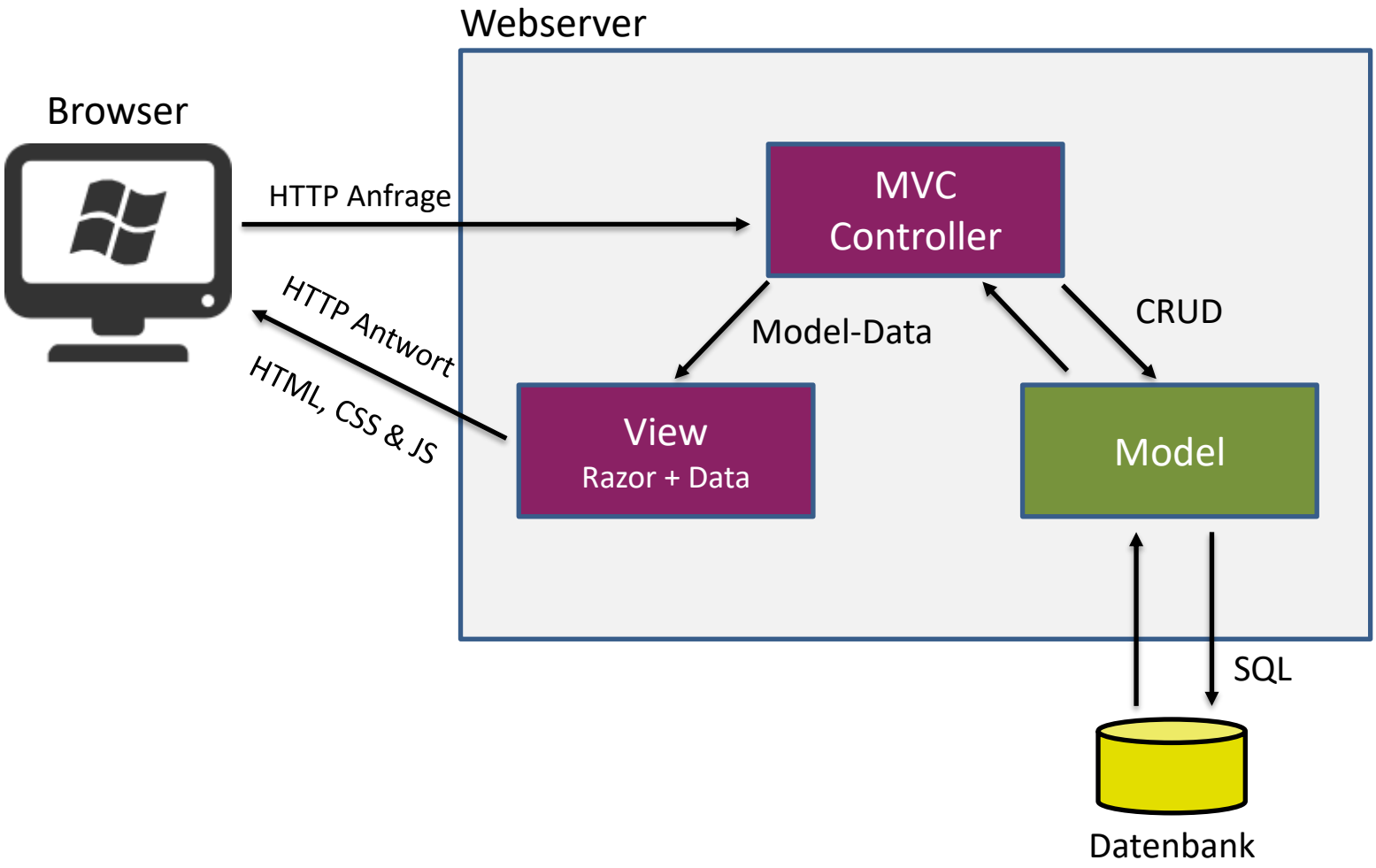
- Software, die zu einer Anwendungspipeline zusammengesetzt wird (Delegate)
- Anwendung durch Extension-Methoden des **WebApplication** Objekts
- Die **WebApplication** stellt den Mechanismus bereit die Request-Pipeline zu konfigurieren
- Built-In-Middleware
  - Mvc, StaticFiles, CORS, Routing, DeveloperExceptionPage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

Hands On

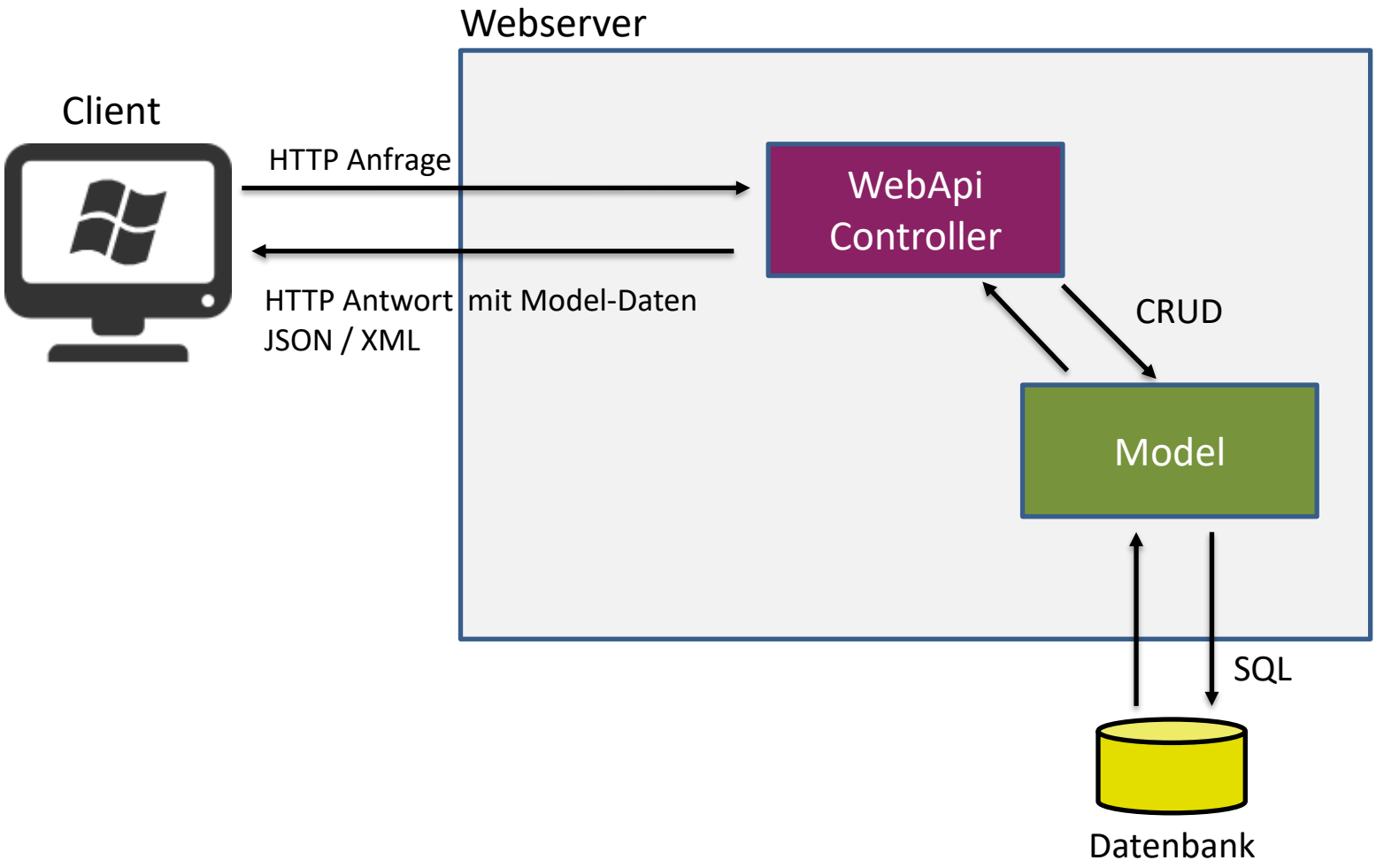
Demo Middleware

Controller & Actions

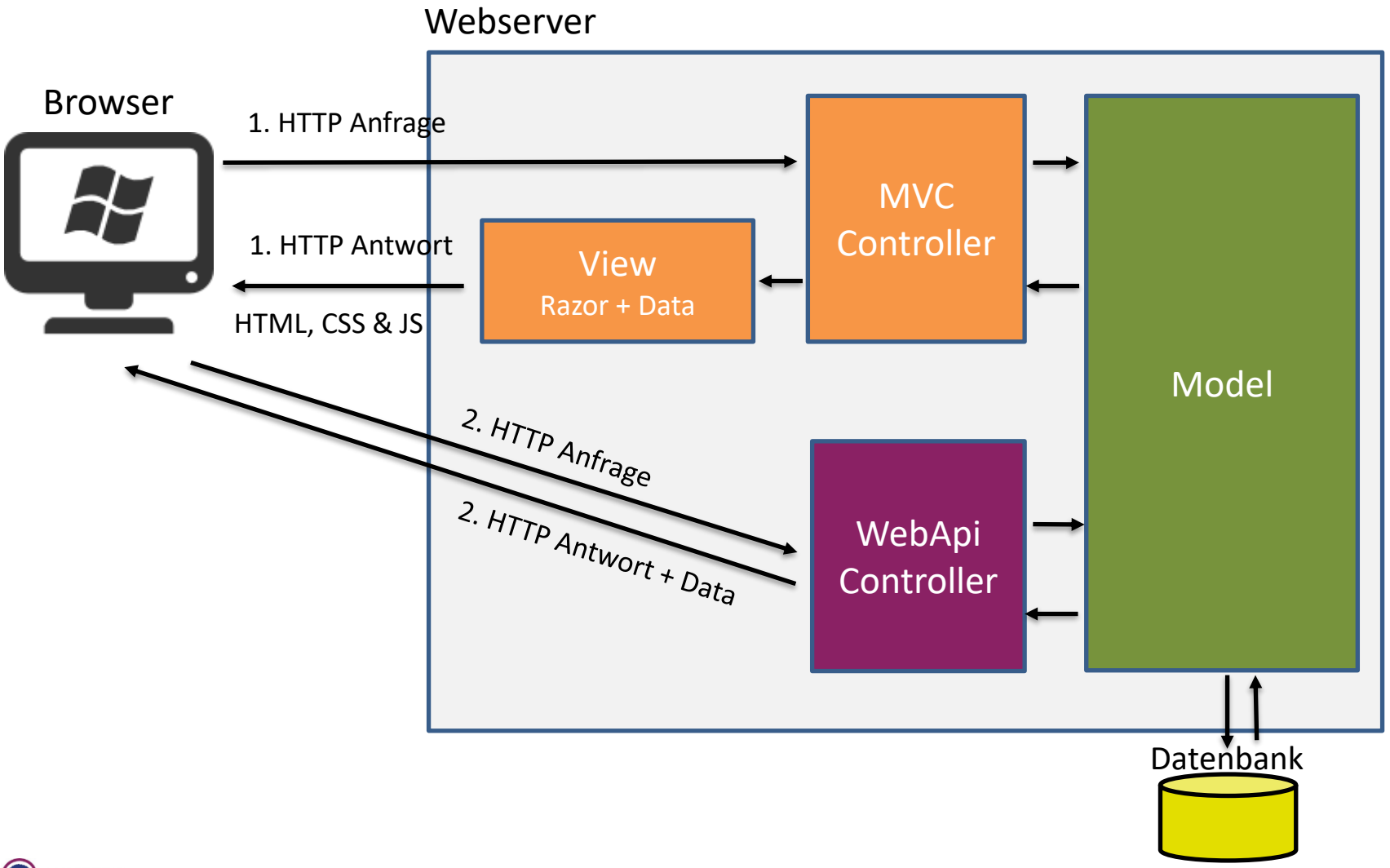
Request-Response Ablauf MVC



Request-Response-Ablauf ASP.NET Core 9 Api



Request-Response Single Page Application



Action Results

- 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
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);
  }
}
```

Asynchrone Controller

- 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

Überblick

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

Was ist ein Model?

Eine Klasse

Model

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

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

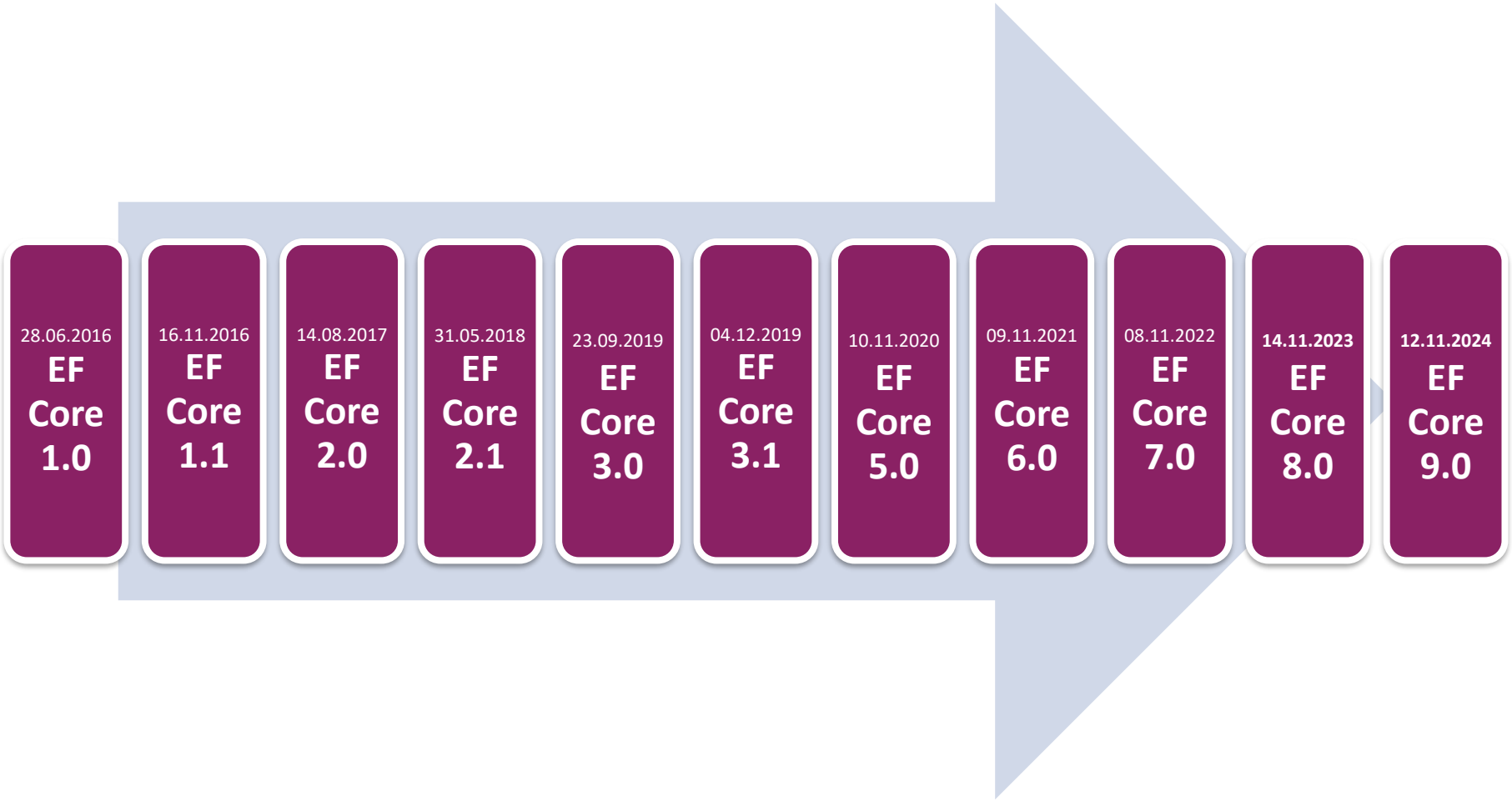
Exkurs: Entity Framework Core 9

ADO.NET Entity Framework

- 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



Ohne Entity Framework

- Fehleranfällig, Unsicher, Kompliziert

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

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 AuthorContext())
{
    db.Authors.Add(new Author { Name = „Mark Twain" });
    db.SaveChanges();
}
```

EF Begrifflichkeiten

- 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

Aufbau Entity Framework

- 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

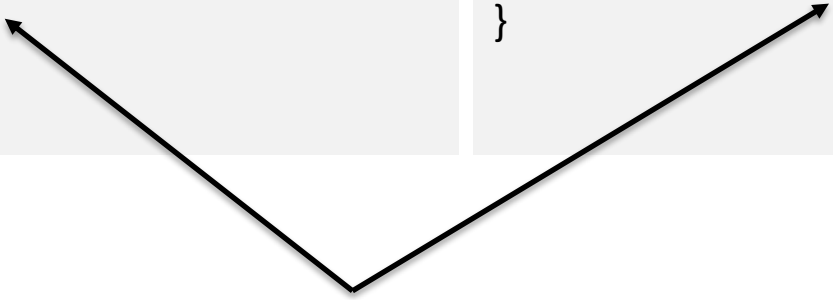
Vorgehensweise	Datenbank vorhanden	Beschreibung
Code First	**Nein**	Objekte und deren Beziehungen werden in Klassen beschrieben. Daraus wird dann mit dem EF die DB erstellt
Reverse Engineering Code First	**Ja**	Generierung von Entity Framework Klassen aus der DB

Code First Entitäten

- Für jede Entity eine Klasse erstellen

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

```
public class Quote
{
   public Guid QuoteId { 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 instanziieren
- Gewünschte Entitäten abfragen (Filtern möglich)
- Als Model an View übergeben

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

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

Database Migrations

- 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

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
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

Validierung

Model - Validierungsmöglichkeiten

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

Built-In-Validierungsattribute (Auszug)

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
Range	Prüft, ob sich der zu validierende Wert in einem bestimmten Wertebereich befindet
RegularExpression	Validiert die Eigenschaft mit regulären Ausdrücken
Required	Markiert die Eigenschaft als Pflichtfeld

Model mit Validierungsattribute

Author
- AuthorId : Guid
- Name : string
- Description : string
- BirthDate: DateOnly

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

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

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

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

Benutzerdefinierte Validierungsattribute

- 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

Author
- AuthorId : Guid
- Name : string
- Description : string
- BirthDate: DateOnly

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

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

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

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

Serverseitige deklarative Validierung

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

```
public IActionResult Post([FromBody] Author author)
{
    if(!ModelState.IsValid)
    {
      ModelState.AddModelError("Name","So ein Pech auch");
         return BadRequest(ModelState);
    }
    return Ok(author);
}
```

Selbstvalidierendes Model

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

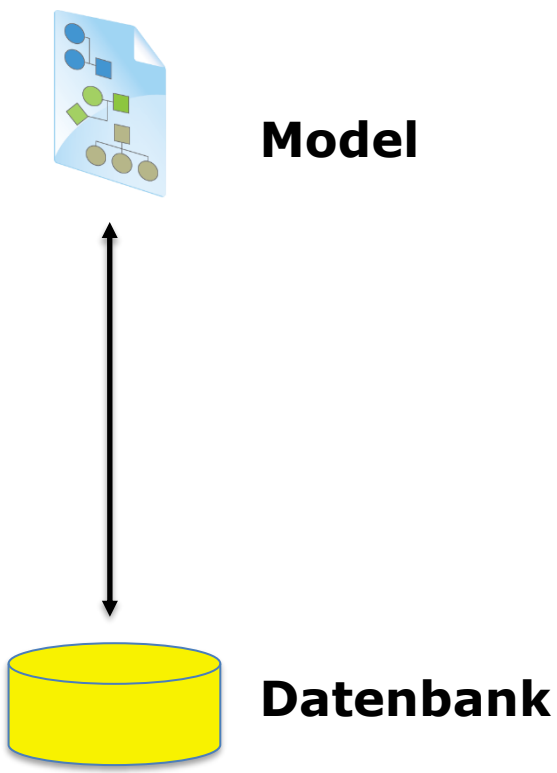
Selbstvalidierendes Model

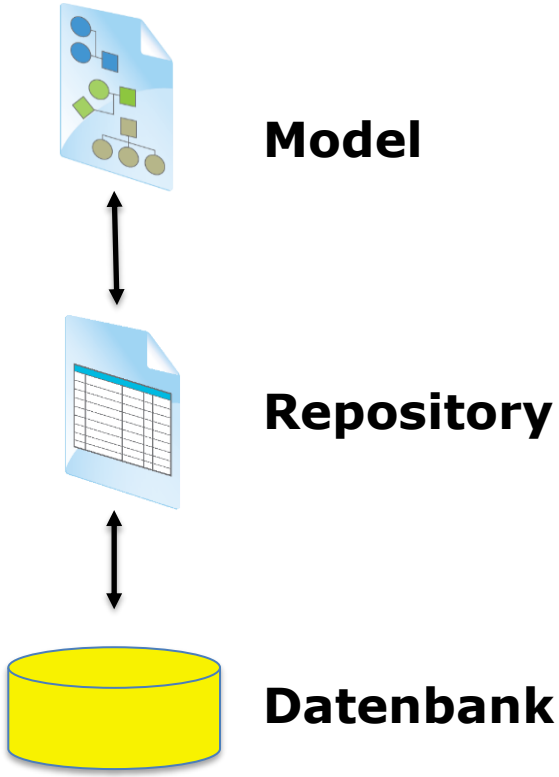
```
public class Teilnehmer : IValidateObject
{
      …
      [StringLength(50)]
      //[NoAdmin]
      public string Name { get; set; }
      …
}

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

Repository

Repository

- Separation of Concerns
- Lose Kopplung





Dependency Injection

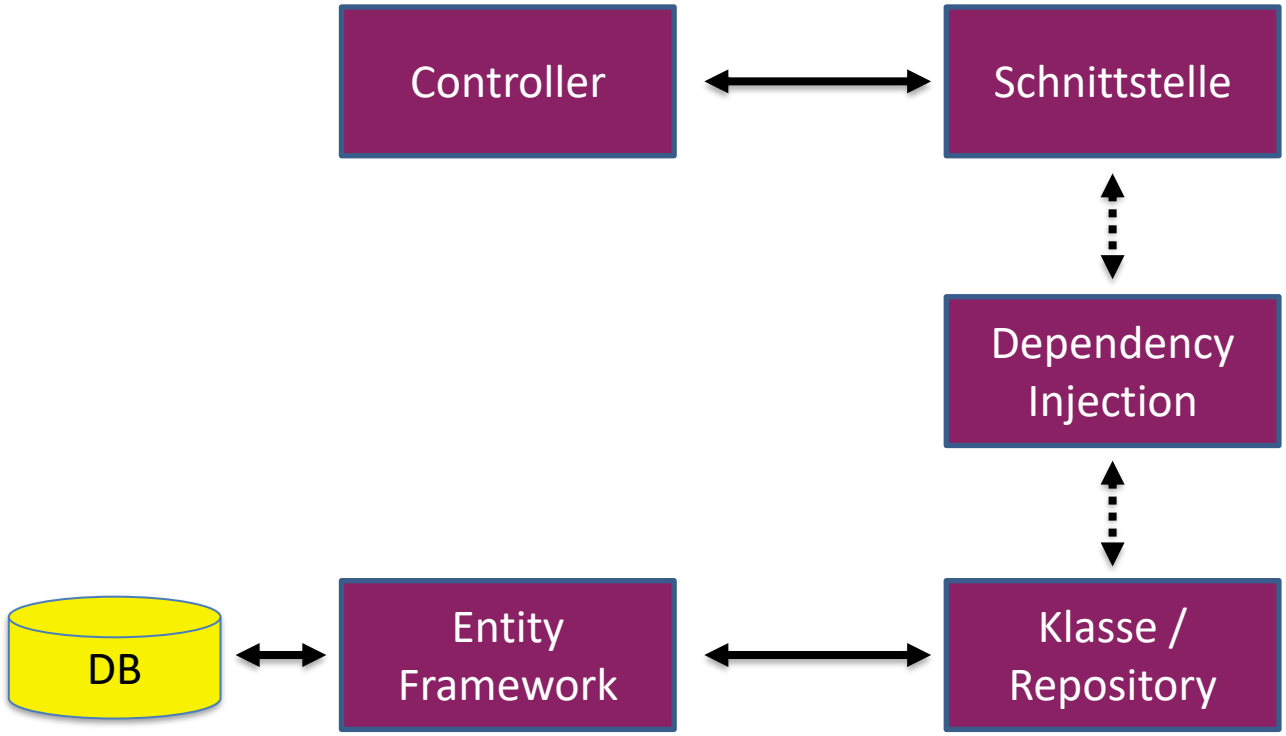
IoC

IoC / Dependency Injection

- ***I**nversion **o**f **C**ontrol* 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 Dependeny Injection



Dependency Injection Beispiel

```
public class AuthorController : ControllerBase
{
  private readonly IAuthorRepository _authorRepository;

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

Interface, *keine* konkrete Implementierung

Konstruktor Injection

Dependency 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

DI Scopes

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

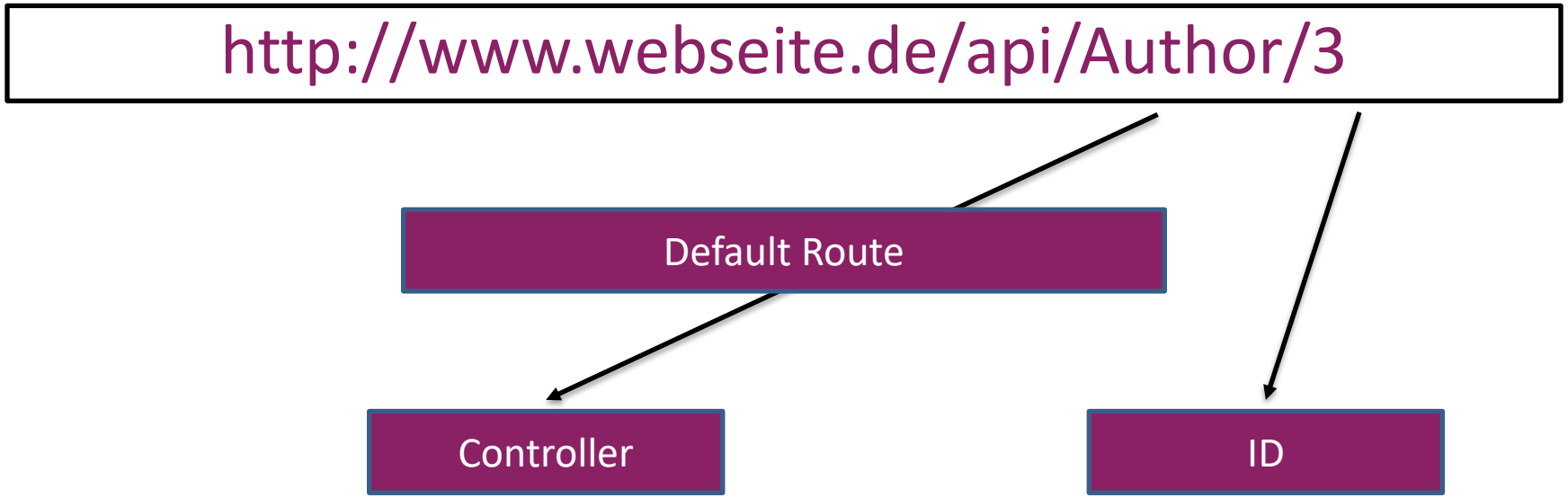
Scope	Bedeutung	Erläuterung
Transient	**vorübergehend**	Eine neue Instanz des Typs wird bei jeder Anforderung des Typs verwendet.
Scoped	**bereichsbezogen**	Eine neue Instanz des Typs wird bei seiner erstmaligen Anforderung in einer bestimmten HTTP-Anforderung erstellt und anschließend für alle nachfolgenden 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.

Routing

Routing

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

Die Standardroute



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

Routing-Konfiguration

- Routing wird in **`Program.cs`** 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

Attribut Routing

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

Attribut Routing Beispiel

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

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

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

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}$)}

Custom Route Constraints

- 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 Contraints 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)
  {
  if (values.TryGetValue(routeKey,out var value) && value != null)
     {
     return _validAuthorDescription.Contains(value.ToString(),StringComparer.OrdinalIgnoreCase);
     }
       return false;
  }
}
```

Custom Routing Contraints Registrierung

2. Registrieren

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

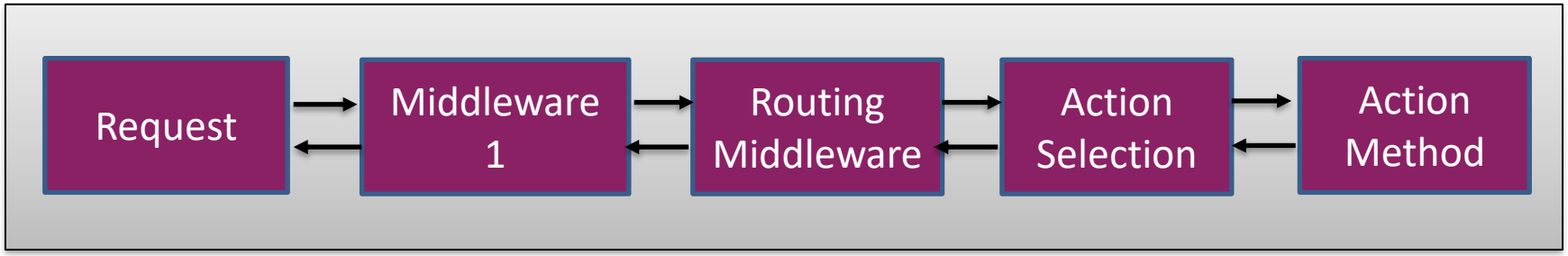
3. Action Attribute

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

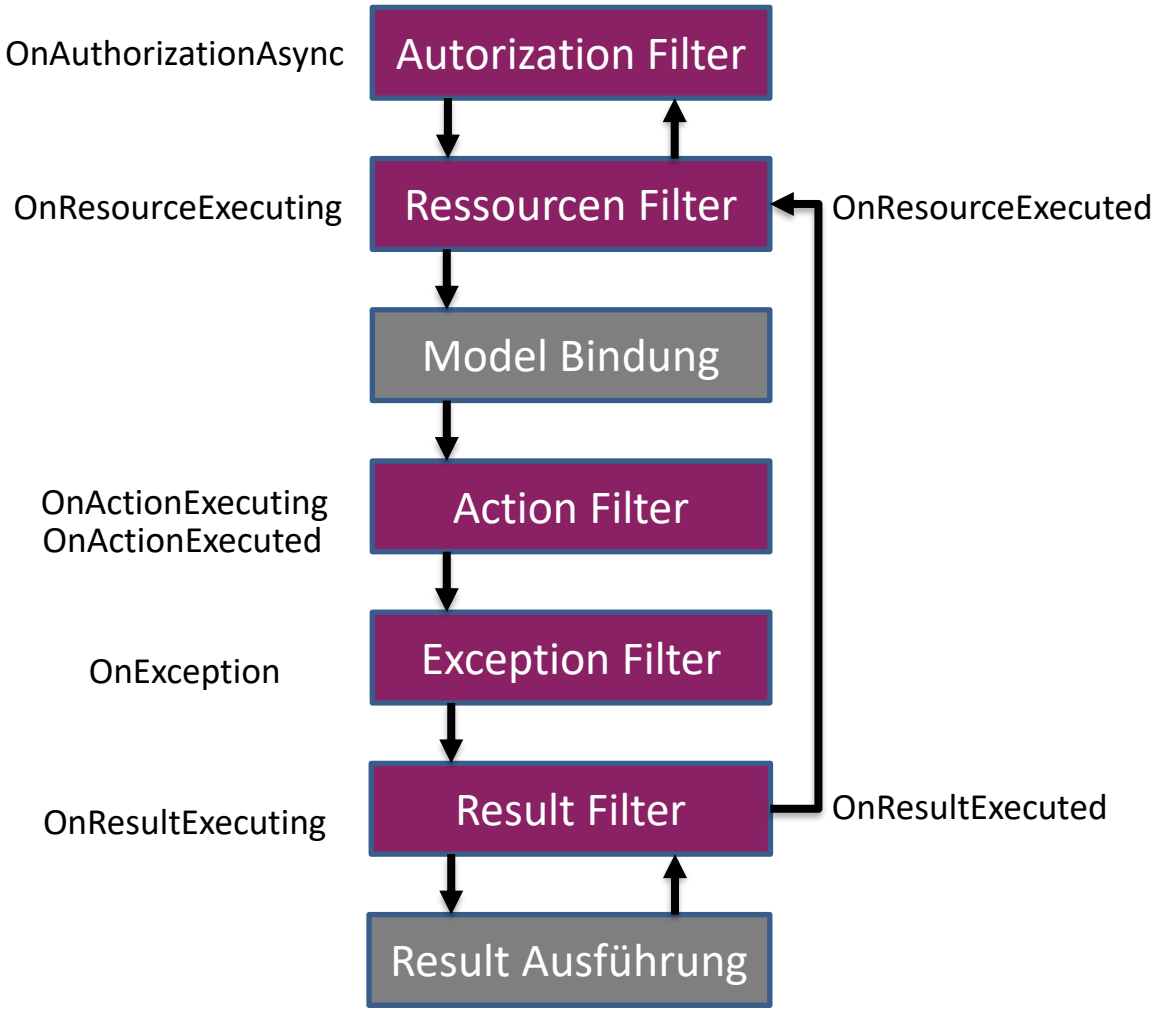
Filter

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	IAsyncAuthorization Filter	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	IExceptionFilter/ IAsyncExceptionFilter	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");
      }
   }
```

DEMO

„Filter TimeAttribute, TimeReloaded“

HttpClient

Http-Client

- 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/qotd";
 var client = new HttpClient();
 var response = await client.GetAsync(url);

 response.EnsureSuccessStatusCode();

 var content = await response.Content.ReadAsStringAsync();
 var quoteDto = JsonSerializer.Deserialize<QotdDto>(content);

 System.Console.WriteLine($"Spruch Rückgabe (Auszug) ->
 Name: {quoteDto.AuthorName} ## Beschreibung:
{quoteDto.AuthorBeschreibung} ##  Spruch:
quoteDto.QuoteText}");

  Console.ReadLine()
}
```

DEMO

„Http-Client“

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/ XmlDataContractSerialzerFormatter	Xml (nutzt DataContractSerializer)

Benutzerdefinierter Media Type Formatter

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

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

DEMO

„CSV Formatter für SpruchDesTages“

CORS

Cross **O**rigin **R**esource **S**haring

- 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

Tracing / Logging

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

Fehlerbehandlung

- Verschiedene Fehlerbehandlungsmöglichkeiten
  - Action-Ebene (**`ExceptionFilterAttribute`**)
  - Controller-Ebene **`(ExceptionFilterAttribute)`**
  - Global durch **`IExceptionHandler oder ExceptionFilterAttribute`**
  - `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(StatusCodes.Status410Gone);
   }
  }
}
```

```
[ResourceRemoved]
public IActionResult Delete(int id)
{
   … auslösen der ResourceRemovedException
}
```

Authentifizierung / Authorisierung

Authentifizierung/Autorisierung

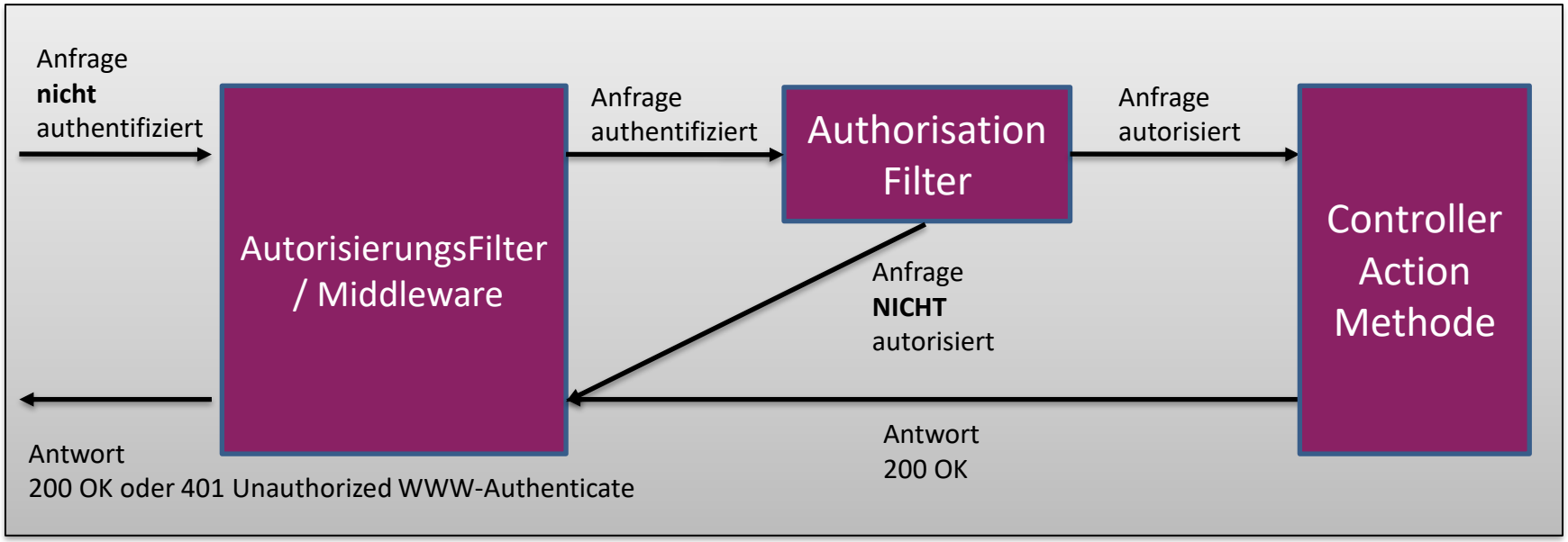
- Authentifikation klärt die **WER**-Frage
  - Ist der Benutzer jener Benutzer, der er vorgibt zu sein?
- Autorisierung klärt die **WAS**-Frage
  - Was darf der Benutzer tun?

- 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 Autorisations-Header mit Base64 codiertem Username und Passwort getrennt durch Doppelpunkt
  - Authorization: Basic YWxpQGV4YW1wbGUuY29tOlNhb
- 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

„Authentifizierung“

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 schützbare Daten
  - **Resource server** = Server, der die Resource hostet
  - **Resource owner** = Die Entität, das den Zugriff auf die Resource erlauben kann (User)
  - **Client** = Die Anwendung, die auf die Resource zugreifen möchte (Webbrowser)
  - **Access token** = Ein Token, dass 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

JWT

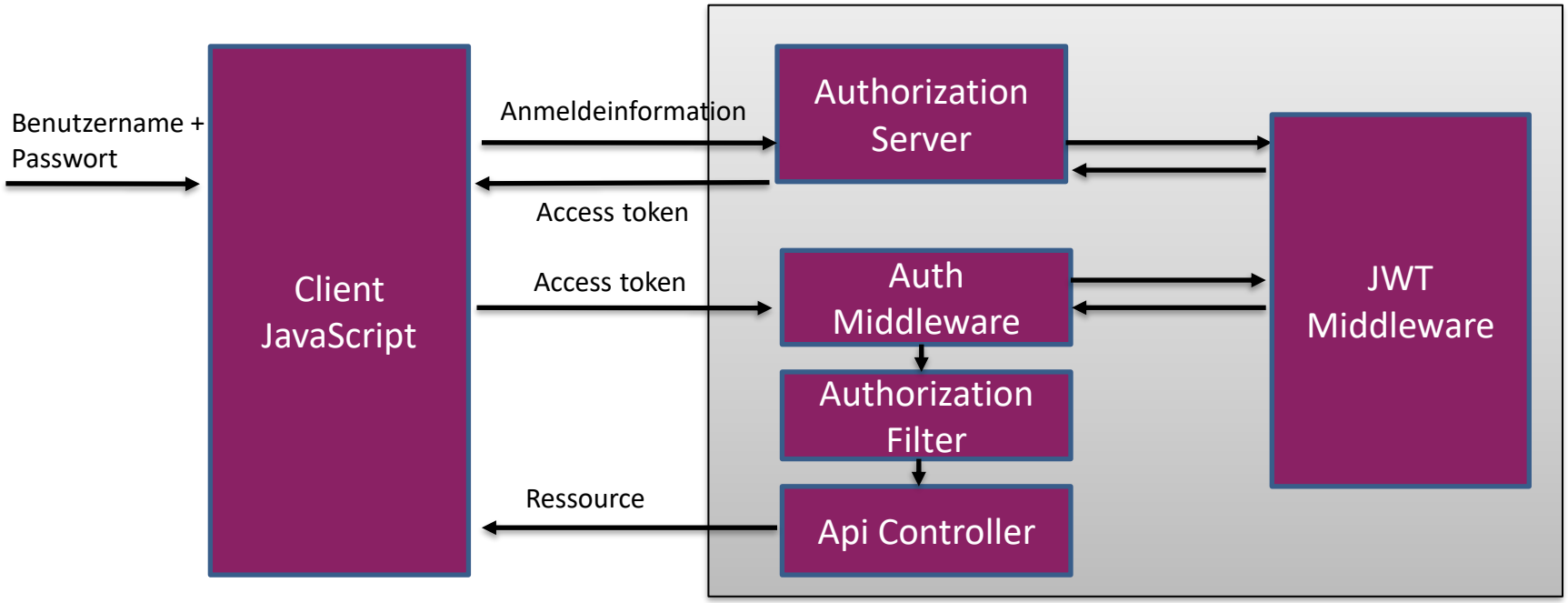
- JSON Web Token (JWT) ist ein Webstandard, der eine Methode zum Übertragen von Claims als JSON-Objekt so definiert, dass sie kryptografisch signiert oder verschlüsselt werden können

- 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)



Security Filters

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

DEMO

„Bearer-Authentifizierung Web- und Client Anwendung“

Versionierung

Versionierung

- Wie werden verschiedene Versionen der Web API verwaltet?
- Philosophische Frage: Sollen Web API versioniert werden?

- Vorgehensweise:
  - NuGet-Paket **`AspVersioning.Mvc`**
  - `ConfigureServices -> AddApiVersioning()`
  - `Attributierung der Controller mit [ApiVersion()]`

- `Verschiedene Möglichkeiten der Versionierung`
  - **`QueryString`** `Parameter`
  - **`Url Path`** `Segment`
  - **`Header`**

JavaScript & JQuery

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

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!)
- Juni 2016 Version 3

JQuery Zugriff auf HTML-Elemente

- 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

AJAX

- **A**synchronous **J**avaScript **a**nd **X**ML / Stichwort Web 2.0
- Erlaubt das Aktualisieren nur von gewissen Bereichen einer Webseite