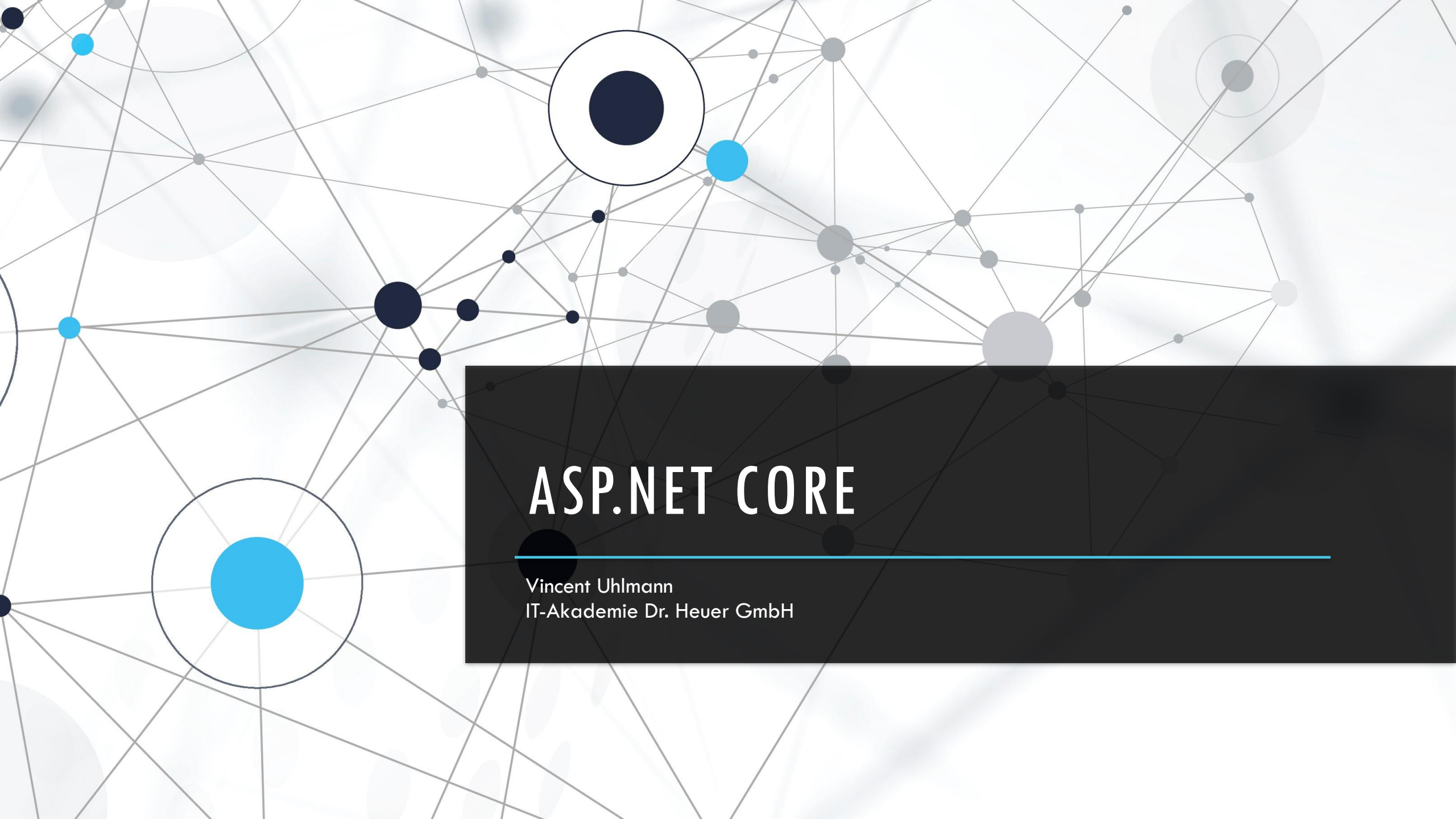

ASP.NET CORE

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

WAS IST ASP.NET CORE?

- ASP.NET Core ist ein plattformübergreifendes, hochleistungsfähiges Open-Source-Framework zum Erstellen moderner, cloudbasierter Web-Anwendungen
- Kann für Web-Apis, Web-Apps, IoT-Apps und Backends für mobile Geräte verwendet werden
- Läuft auf Windows, macOS und Linux
- Bietet eine modulare Architektur mit Middleware-Komponenten
- Entwickelt für hohe Leistung und Skalierbarkeit
- Entwickelt und gepflegt von Microsoft und der Community

MVC (MODEL-VIEW-CONTROLLER)

- Framework zum Erstellen von Webanwendungen nach dem MVC-Muster, das eine klare Trennung von Anwendungslogik, UI und Daten fördert
- **Model**: Repräsentiert die Daten und die Geschäftslogik
- **View**: Präsentationslogik (HTML und UI)
- **Controller**: Verarbeitet Benutzeranfragen, arbeitet mit dem Model und gibt die Ergebnisse an die View weiter

RAZOR PAGES

- Framework für serverseitiges Rendering von Web-UI
- Webseiten und Formulare, bei denen eine enge Kopplung zwischen UI und Anwendungslogik gewünscht ist

- **MVVM-ähnliches Muster**: Trennung von Markup und Logik
- **Seitenzentriert**: Jede Seite hat ein eigenes Page-Model, das die Logik enthält
- **Integriert mit Razor-View-Engine**: Verwendung von Razor-Syntax für HTML-Rendering
- **Einfach zu verwenden**: Weniger Boilerplate-Code im Vergleich zu MVC

BLAZOR

- **Blazor**
- Framework zum Erstellen interaktiver Web-UI mit C# statt JavaScript
- Wiederverwendbare Komponenten fürs UI

- **Blazor Server**
- **Ausführung**: Code läuft auf dem Server
- **Kommunikation**: Echtzeit-Kommunikation über SignalR
- **Vorteile**: Schnelle Ladezeiten, Zugriff auf .NET-APIs und –Ressourcen
- **Nachteile**: Höhere Latenz bei langsamer Internetverbindung, Abhängigkeit von der Serververfügbarkeit

BLAZOR

- **Blazor WebAssembly**
- **Ausführung:** Code läuft direkt im Browser
- **Kommunikation:** Direkter Zugriff auf Browser-APIs
- **Vorteile:** Unabhängig von der Serververfügbarkeit nach dem Laden, Offline-Betrieb möglich
- **Nachteile:** Größere Ladezeiten bei der ersten Anfrage

CLIENTSEITIGE VS. SERVERSEITIGE RENDERING

- **Serverseitiges Rendering (SSR)**
- **Definition:** Rendering der HTML-Seiten auf dem Server
- **Schnellere Ladezeiten:** Erste Seite wird schneller geladen, da HTML vom Server kommt
- **Bessere SEO-Optimierung:** Suchmaschinen können die gerenderten Seiten besser indexieren
- **Weniger Abhängigkeit vom Client:** Funktioniert auch auf älteren oder leistungsschwachen Geräten
- **Beispiele:** Razor Pages, MVC, Blazor Server

CLIENTSEITIGE VS. SERVERSEITIGE RENDERING

- **Clientseitiges Rendering (CSR)**
- **Definition:** Rendering der HTML-Seiten im Browser
- **Bessere Benutzererfahrung:** Schnellere und interaktive UI, da keine kompletten Seiten neu geladen werden
- **Reduzierte Serverlast:** Server muss weniger Rendering-Aufgaben übernehmen
- **Offline-Fähigkeit:** Anwendungen können teilweise offline funktionieren
- **Beispiele:** Blazor WebAssembly, SPAs mit Angular, React

BLAZOR: SERVER VS. WEBASSEMBLY

- **Blazor Server**
- **Ausführung:** Code läuft auf dem Server
- **Kommunikation:** Echtzeit-Kommunikation über SignalR

- **Vorteile**
- **Schnelle Ladezeiten:** Keine großen Downloads nötig, da der Code auf dem Server läuft
- **Zugriff auf .NET-APIs:** Direkter Zugriff auf Server-Ressourcen und APIs#

- **Nachteile**
- **Latenz:** Höhere Latenz bei langsamer Internetverbindung
- **Serverabhängigkeit:** Anwendung funktioniert nur, wenn der Server verfügbar ist

BLAZOR: SERVER VS. WEBASSEMBLY

- **Blazor WebAssembly**
- **Ausführung:** Code läuft direkt im Browser
- **Kommunikation:** Direkter Zugriff auf Browser-APIs

- **Vorteile**
- **Unabhängigkeit:** Läuft unabhängig vom Server, nachdem die Anwendung geladen wurde
- **Offline-Fähigkeit:** Kann auch ohne Internetverbindung funktionieren

- **Nachteile**
- **Ladezeit:** Größere initiale Ladezeit, da die Anwendung im Browser heruntergeladen wird

WEB API

- **Web API**
- Framework zum Erstellen von HTTP-Diensten, die von verschiedenen Clients (Browser, mobile Geräte) genutzt werden können
- Ideal für RESTful APIs und Microservices
- Unterstützung für JSON, XML und andere Formate
- Integration mit Entity Framework Core für Datenzugriff
- Minimaler Overhead, ideal für skalierbare Dienste
- Unterstützung für Swagger (OpenAPI), Routing, Model Binding und Validierung

WEB API

- **Anwendungsbeispiele**
- Backend-Services für Single Page Applications (SPAs) und mobile Apps
- Microservices in einer verteilten Architektur
- Öffentliche APIs für externe Entwickler