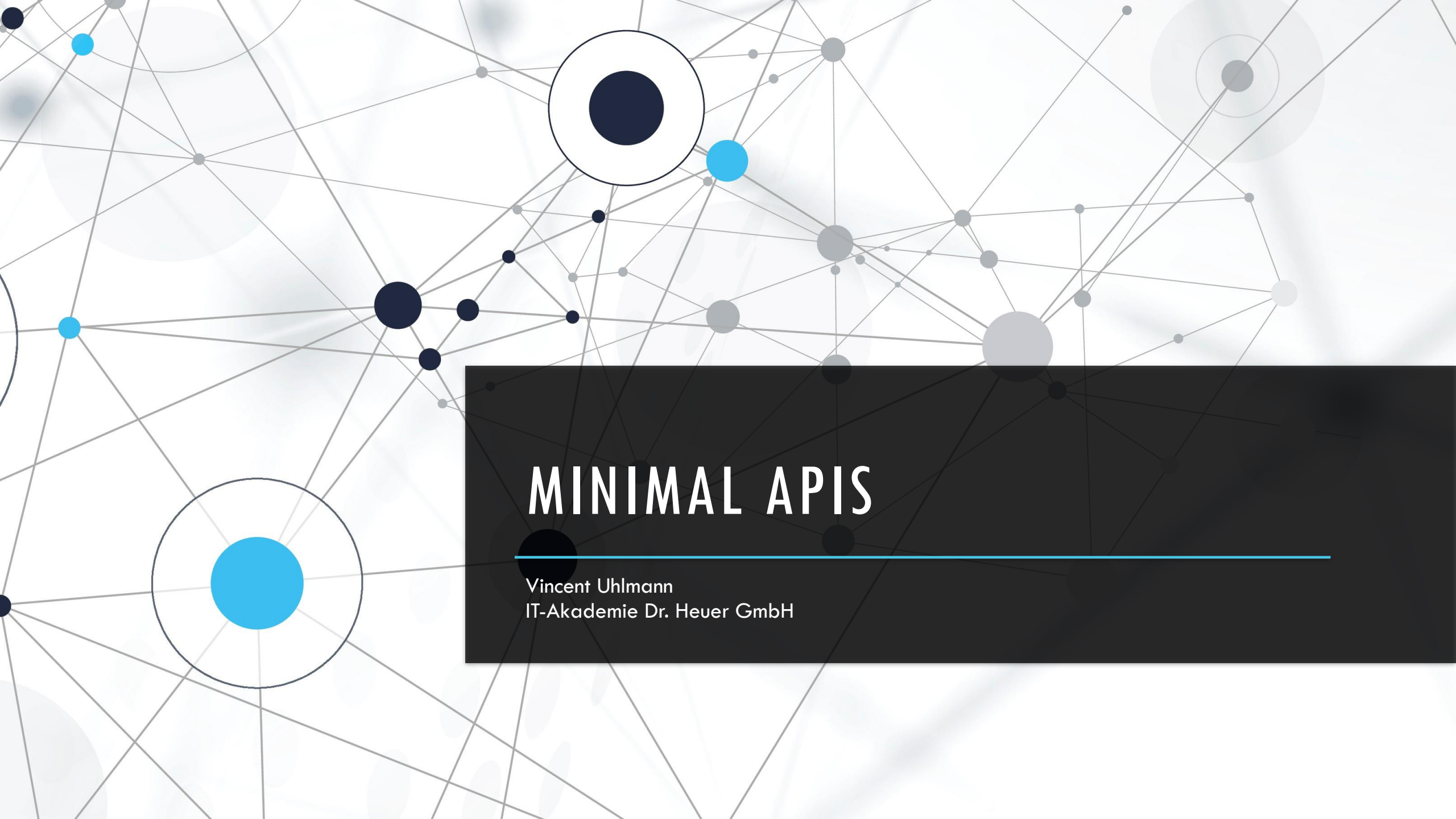

MINIMAL APIS

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

MINIMAL API

- **Minimal API** ist ein leichtgewichtiger Ansatz, um HTTP-APIs in ASP.NET Core zu erstellen
- Entwickelt für einfache und schnelle API-Entwicklung ohne den Overhead von Controllern und Aktionen in traditionellen MVC-Architekturen

- **Vorteile**
- Weniger Boilerplate-Code
- Einfacher Einstieg
- Schneller für einfache APIs
- Keine Controller oder Actions erforderlich
- Direktes Mapping von HTTP-Verben zu Methoden

MINIMAL API

- Ein Minimal API-Projekt in ASP.NET Core wird in der Program.cs Datei definiert
- Dieser Code erstellt einen einfachen GET-Endpunkt, der den Text "Hello World!" zurückgibt
- app.MapGet registriert einen GET-Endpunkt
- Der Pfad /hello spezifiziert die URL des Endpunkts

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// builder.Services.AddSwaggerGen();

var app = builder.Build();

app.MapGet("/hello", () =>
{
    return Results.Ok("Hello World!");
});

app.Run();
```

GET METHODE

- **app.MapGet**: Definiert eine Route, die auf eine GET-Anfrage reagiert
- **Results.Ok**: Gibt eine 200 OK Antwort mit den gewünschten Daten zurück

```
app.MapGet("/items", (HttpContext context) =>
{
    string[] items = { "Item1", "Item2", "Item3" };
    return Results.Ok(items);
});
```

POST METHODE

- **app.MapPost**: Definiert eine Route, die auf eine POST-Anfrage reagiert
- **Results.Created**: Gibt eine 201 Created Antwort zurück mit dem Speicherort der neuen Ressource

```
app.MapPost("/items", async (HttpContext context) =>
{
    using var reader = new StreamReader(context.Request.Body);
    string bodyContent = await reader.ReadToEndAsync();

    // Verarbeitung der empfangenen Daten z.B. mit EF Core
    return Results.Created($"/items/{1}", new { Id = 1, Content = bodyContent });
});
```

PUT METHODE

- **app.MapPut**: Definiert eine Route, die auf eine PUT-Anfrage reagiert
- **Results.NoContent**: Gibt eine 204 No Content Antwort zurück, um anzuzeigen, dass die Aktualisierung erfolgreich war

```
app.MapPut("/items/{id}", async (HttpContext context, int id) =>
{
    using var reader = new StreamReader(context.Request.Body);
    var bodyContent = await reader.ReadToEndAsync();
    // Aktualisierung der Ressource mit der angegebenen ID
    return Results.NoContent();
});
```

DELETE METHODE

- **app.MapDelete**: Definiert eine Route, die auf eine DELETE-Anfrage reagiert
- **Results.NoContent**: Gibt eine 204 No Content Antwort zurück, um anzuzeigen, dass die Löschung erfolgreich war

```
app.MapDelete("/items/{id}", (HttpContext context, int id) =>
{
    // Löschen der Ressource mit der angegebenen ID
    return Results.NoContent();
});
```

RESULTS

- Results bietet eine einfache Möglichkeit, HTTP-Antworten zurückzugeben
- Kann verschiedene Arten von HTTP-Antworten zurückgeben, einschließlich Ok, Created, BadRequest, NoContent, etc.
- Erlaubt die direkte Manipulation von HTTP-Statuscodes und –Nachrichten

```
app.MapDelete("/items/{id}", (HttpContext context, int id) =>
{
    // Löschen der Ressource mit der angegebenen ID
    return Results.NoContent();
});
```

TYPED-RESULTS

- **TypedResults** bietet stark typisierte Rückgabewerte, die eine bessere Integration und Darstellung in Swagger ermöglichen
- **TypedResults** erleichtert die automatische Generierung von präzisen und detaillierten Swagger-Dokumentationen
- Klarere API-Spezifikationen, da die Rückgabetypen genau definiert sind

```
app.MapPut("/items/{id}", async (HttpContext context, int id) =>
{
    using var reader = new StreamReader(context.Request.Body);
    var bodyContent = await reader.ReadToEndAsync();
    // Aktualisierung der Ressource mit der angegebenen ID
    return TypedResults.NoContent();
});
```

DATEN AUS QUERY PARAMETERN LESEN

- **HttpContext**: Ermöglicht den Zugriff auf HTTP-Anforderungs- und Antwortdaten
- **context.Request.Query**: Zugriff auf die Query-Parameter der Anfrage

```
app.MapGet("/queryTest", (HttpContext context) =>
{
    var param = context.Request.Query["search"];
    return Results.Ok($"Query Parameter: {param}");
});
```

DATEN AUS BODY LESEN

- **HttpContext**: Ermöglicht den Zugriff auf HTTP-Anforderungs- und Antwortdaten
- **StreamReader**: Zum Lesen des Inhalts des Anforderungsbodys
- **context.Request.Body**: Zugriff auf den Anforderungsbody

```
app.MapPost("/create", async (HttpContext context) =>
{
    using var reader = new StreamReader(context.Request.Body);
    var bodyContent = await reader.ReadToEndAsync();
    return Results.Ok(bodyContent);
});
```

DATEN AUS HEADER LESEN

- **HttpContext**: Ermöglicht den Zugriff auf HTTP-Anforderungs- und Antwortdaten
- **context.Request.Headers**: Zugriff auf die Header der Anfrage

```
app.MapGet("/headerTest", (HttpContext context) =>
{
    var headerValue = context.Request.Headers["Custom-Header"];
    return Results.Ok($"Header Value: {headerValue}");
});
```

ÜBERPRÜFUNG DES ACCEPT-HEADERS

- **Accept-Header:** Wird vom Client gesendet, um die bevorzugte Antwortmedien-Typen anzugeben (z.B. application/json, application/xml)
- Die API kann die Antwort basierend auf dem Accept-Header anpassen, um den Anforderungen des Clients zu entsprechen

- **Flexibilität:** Ermöglicht der API, verschiedene Antwortformate zu unterstützen
- **Spezifische Antworten:** Liefert passende Antworten für verschiedene Clientanforderungen

UMGANG MIT NICHT UNTERSTÜTZTEN HEADERN

```
app.MapGet("/content", (HttpContext context) =>
{
    var acceptHeader = context.Request.Headers["Accept"].ToString();

    if (acceptHeader.Contains("application/xml"))
    {
        var xmlResponse = "<message>Hello, World!</message>";
        context.Response.ContentType = "application/xml";
        return Results.Ok(xmlResponse);
    }
    else if (acceptHeader.Contains("application/json"))
    {
        var jsonResponse = "{ \"message\": \"Hello, World\"}";
        context.Response.ContentType = "application/json";
        return Results.Ok(jsonResponse);
    }
    else
    {
        return Results.StatusCode(StatusCodes.Status406NotAcceptable);
    }
});
```

SERVICES IN MINIMAL API INJIZIEREN

- **context.RequestServices.GetRequiredService**: Ermöglicht das Abrufen von Diensten aus dem DI-Container

```
app.MapGet("/serviceTest", (HttpContext context) =>
{
    var myService = context.RequestServices.GetRequiredService<IDataService>();
    return Results.Ok(myService.GetMessage());
});
```

ROUTENPARAMETER

- Im Endpunkt wird eine Route mit einem Platzhalter {id} definiert
- Die Methode des Endpunkts hat einen Parameter int id, der dem Routenplatzhalter {id} entspricht
- ASP.NET Core erkennt, dass der Wert des {id} Platzhalters aus der Route genommen und dem Parameter id der Methode zugewiesen werden soll

```
app.MapPut("/test/{id}", (int id) => {
    return Results.Created();
});
```

NAMED ENDPOINTS

- **Definition**: Named Endpoints sind Routen, denen ein Name zugewiesen wird
- **Vorteile**: Ermöglicht die einfache Link-Generierung und Verwaltung von Routen
- Der Endpunkt /test/{id} bekommt den Namen TestRoute

```
app.MapPut("/test/{id}", (int id) => {
    return Results.Created();
}).WithName("TestRoute");
```

LINK-GENERIERUNG

- **Definition:** Link-Generierung ermöglicht das Erstellen von URLs basierend auf benannten Endpunkten
- **Hypermedia:** Unterstützt HATEOAS (Hypermedia as the Engine of Application State) in RESTful APIs

```
app.MapGet("/items/{id}", (int id) => {
    return Results.Ok(new { Id = id, Content = "Some content" });
}).WithName("GetItemById");

app.MapPost("/items", (HttpContext context) =>
{
    var linkGenerator = context.RequestServices.GetRequiredService<LinkGenerator>();
    string? path = linkGenerator.GetPathByName("GetItemById", new { id = 1 });
    return Results.Created(path, new { Id = 1, Content = "Some content" });
});
```