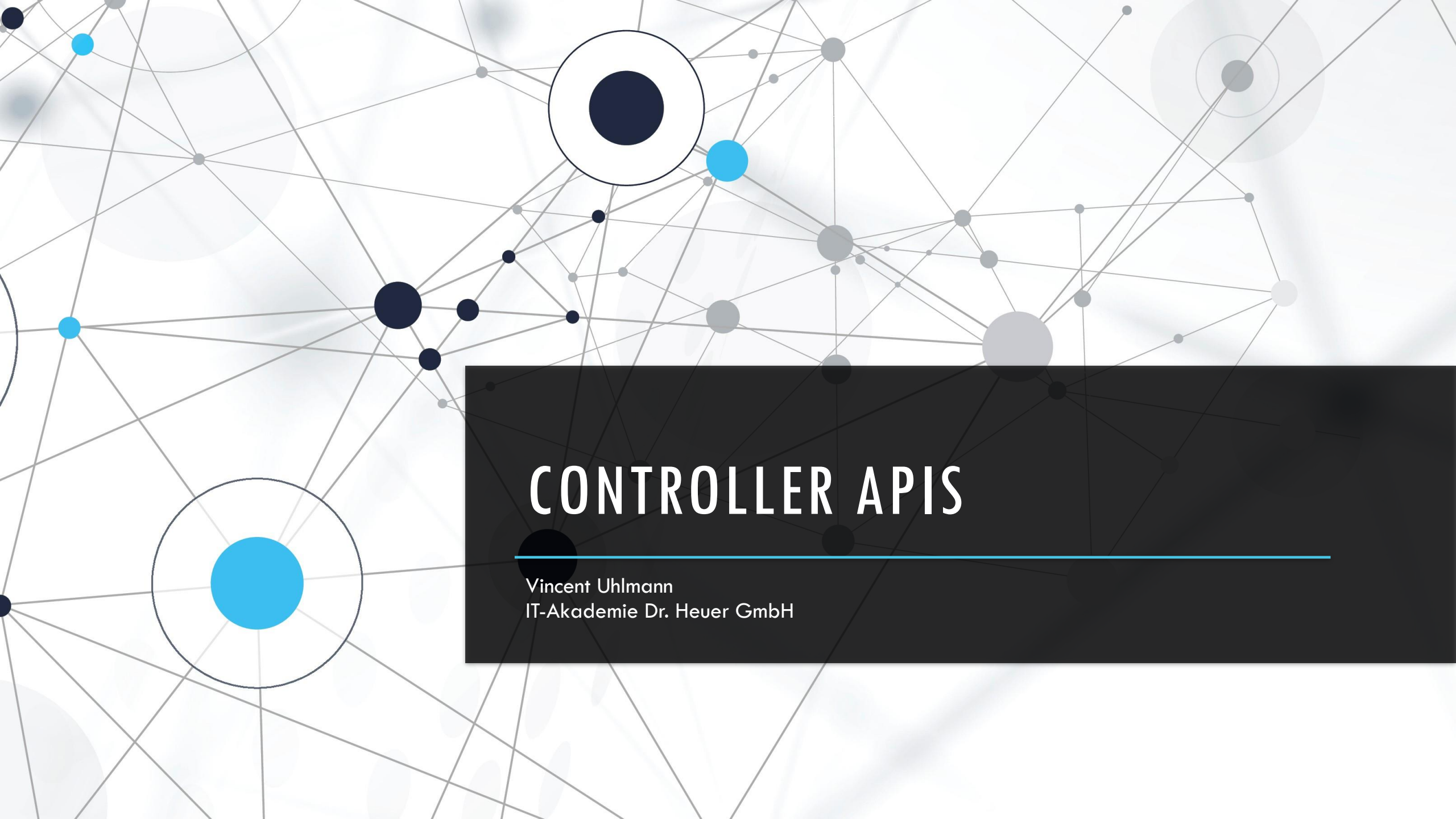

CONTROLLER APIS

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

CONTROLLER APIS

- Controller-basierte APIs in ASP.NET Core sind eine strukturierte Art und Weise, HTTP-Anfragen zu verarbeiten und darauf zu antworten
- Sie basieren auf dem MVC-Pattern (Model-View-Controller)

- **Vorteile**
- Klare Trennung von Verantwortlichkeiten
- Integrierte Unterstützung für Filter, Model Binding und Validierung
- Gute Testbarkeit

CONTROLLER APIS

- Endpunkte werden in Controller gruppiert
- Alle zugehörigen Endpunkte sind in einem Controller zusammengefasst
- Jedem Endpunkt wird eine sogenannte Action-Methode zugewiesen

- **PersonsController**
- GET /persons
- POST /persons
- PUT /persons
- DELETE /persons/{id}
- GET /persons/{id}

CONTROLLER APIS

- **Aufbau eines Controllers**
- **[ApiController]:** Deklariert die Klasse als API-Controller und aktiviert automatische Verhaltensweisen wie Model Binding und Validierung
- **[Route]:** Definiert die Route für die Controller-Aktionen
- **[controller]** innerhalb der Route stellt sicher, dass der erste Teil des Controller-Namens für die Route verwendet wird (api/persons)

```
[ApiController]
[Route("api/[controller]")]
public class PersonsController : ControllerBase
{
    // Action-Methoden hier
}
```

CONTROLLERBASE

- ControllerBase ist eine Basisklasse für Controller ohne View-Support in ASP.NET Core
- Stellt eine Vielzahl von Methoden und Eigenschaften bereit, die in API-Controllern häufig verwendet werden
- Methoden wie Ok(), NotFound(), BadRequest() usw. erleichtern die Rückgabe von standardisierten HTTP-Antworten
- Zugriff auf Kontextinformationen wie HttpContext, ModelState, Request, Response usw.

```
[ApiController]
[Route("api/[controller]")]
public class PersonsController : ControllerBase
{
    // Action-Methoden hier
}
```

ACTION-METHODEN

- Action-Methoden sind die Endpunkte eines Controllers, die HTTP-Anfragen verarbeiten und Antworten zurückgeben
- **IActionResult:** Basistyp für viele Rückgabearten wie z.B. Ok(), NotFound(), BadRequest() ...
- **ActionResult<T>:** Generischer Typ, der es ermöglicht, sowohl konkrete Typen als auch IActionResult zurückzugeben
- **Concrete Types:** Direktes Zurückgeben von Daten (z.B. string, int, CustomModel)

```
[HttpGet]
public IActionResult GetFooMessage()
{
    return Ok(new { Message = "Hello World!" });
}
```

ACTION-METHODEN

- Asynchrone Methoden verbessern die Skalierbarkeit und Leistung von Anwendungen
- **Task<IActionResult>**
- **Task<ActionResult<T>>**
- **Task<T>**

```
[HttpGet]
public async Task<IActionResult> GetFooMessageAsync()
{
    // Do some async work
    await Task.Delay(1000);
    return Ok(new { Message = "Hello World!" });
}
```

HTTP-VERBEN

- **[HttpGet]:** Binden an GET-Anfragen
- **[HttpPost]:** Binden an POST-Anfragen
- **[HttpPut]:** Binden an PUT-Anfragen
- **[HttpDelete]:** Binden an DELETE-Anfragen
- …

```
[HttpGet]
public IActionResult GetFooMessage()
{
    return Ok(new { Message = "Hello World!" });
}
```

ACTION-METHODEN ATTRIBUTE

- **[Route]:** Definiert die Route für die Action-Methode

```
[HttpGet]
[Route("api/sample/{id}")]
public IActionResult Get()
{
    return Ok(new { Message = "Hello World!" });
}
```

- **[Produces]:** Definiert die MIME-Typen, die die Methode zurückgeben kann

```
[HttpGet]
[Produces("application/json")]
public IActionResult Get()
{
    return Ok(new { Message = "Hello World!" });
}
```

ACTION-METHODEN ATTRIBUTE

- **[ProducesResponseType]**: Dokumentiert die erwarteten Antworttypen

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public IActionResult Get()
{
    return Ok(new { Message = "Hello World!" });
}
```

- …

ACTION-METHODEN ATTRIBUTE

- Routenparameter können in die Route und als Parameter in die Methode geschrieben werden

```
[HttpGet("{id}")]
public ActionResult<TaskItem> Get(int id)
{
    var model = _repository.GetById(id);
    if (model == null)
    {
        return NotFound();
    }
    return Ok(model);
}
```

KONFIGURATION

- Um Controller nutzen zu können müssen in der Program.cs einige Einstellung vorgenommen werden
- Nutzen Sie builder.Services.AddControllers(), um die notwendigen Dienste für Controller-basierte APIs hinzuzufügen
- Verwenden Sie app.MapControllers(), um die Routen der Controller in der Anwendung zu registrieren

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

builder.Services.AddControllers();

var app = builder.Build();

app.MapControllers();

app.Run();
```