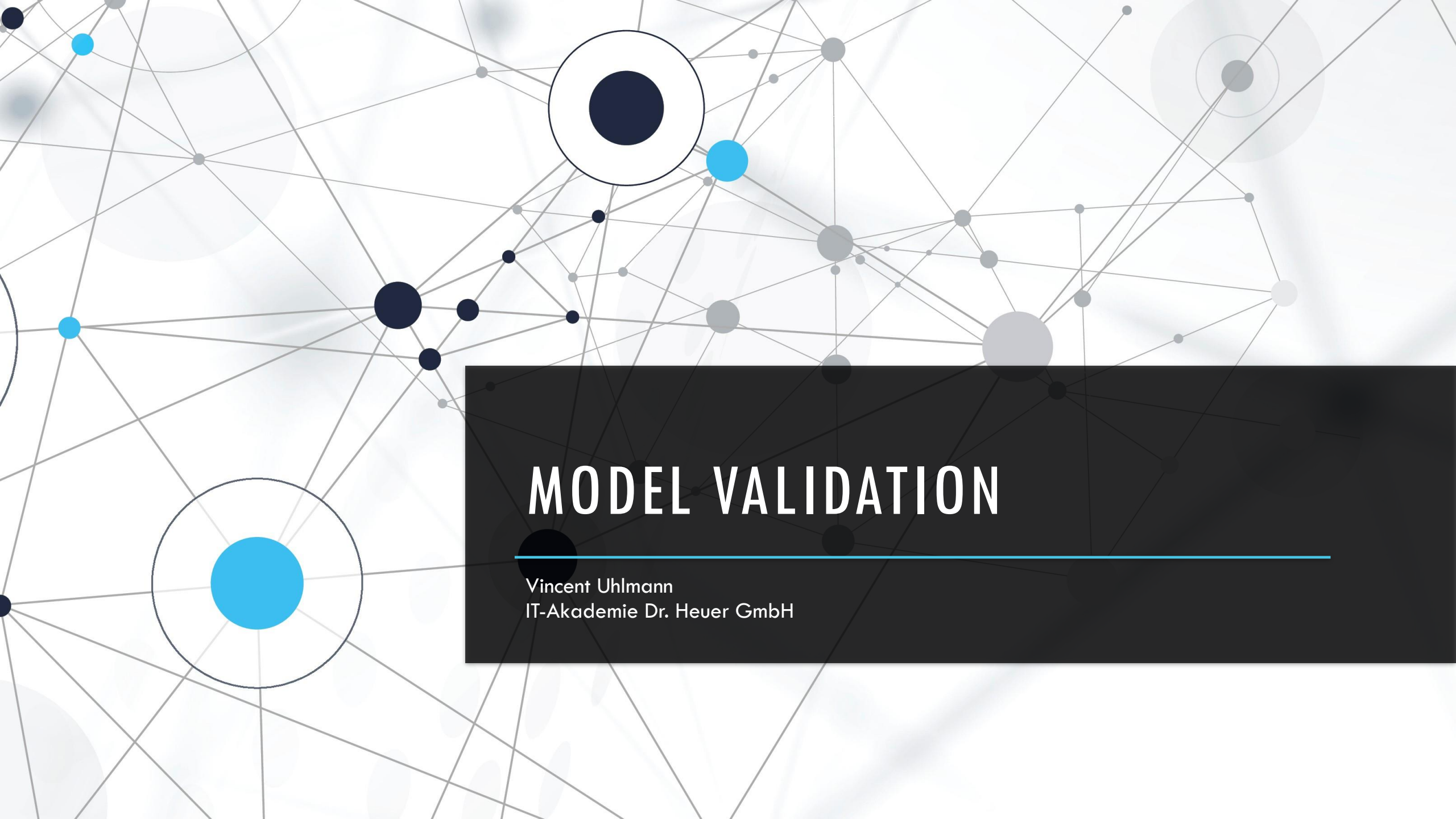

MODEL VALIDATION

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

MODEL VALIDATION

- Model Validation ist der Prozess, bei dem überprüft wird, ob die Daten eines Modells den festgelegten Regeln und Anforderungen entsprechen
- Es soll sicherstellen, dass die von Benutzern bereitgestellten Daten korrekt und vollständig sind, bevor sie weiterverarbeitet werden
- Verbessert die Datenintegrität
- Vermeidet Laufzeitfehler durch unzureichende oder ungültige Daten
- Gilt für Minimal-Apis, MVC-Controller, API-Controller und Razor Pages

VALIDIERUNGSATTRIBUTE

- **[Required]:** Kennzeichnet, dass ein Feld einen Wert haben muss
- **[Range]:** Definiert den zulässigen Wertebereich für numerische und Datumsfelder
- **[StringLength]:** Bestimmt die maximale und minimale Länge einer Zeichenkette
- **[Min/MaxLength] :** Legt die maximale und minimale Anzahl von z.B. Arrays/Listen fest
- **[RegularExpression]:** Validiert ein Feld anhand eines regulären Ausdrucks
- **[EmailAddress]:** Überprüft, ob ein Feld eine gültige E-Mail-Adresse enthält
- **[Phone]:** Überprüft, ob ein Feld eine gültige Telefonnummer enthält
- **[CreditCard]:** Validiert, ob ein Feld eine gültige Kreditkartennummer enthält
- **[Compare]:** Vergleicht den Wert eines Feldes mit dem eines anderen Feldes im Modell

VALIDIERUNGSATTRIBUTE

```
public class CreateUserRequest
{
    [Required]
    public string Name { get; set; }

    [Range(18, 99)]
    public int Age { get; set; }
}
```

VALIDIERUNGSATTRIBUTE

- Die Fehlermeldung, die im Falle eines Validierungsfehlers ausgegeben wird, kann mit dem Parameter ErrorMessage innerhalb der Validierungsattribute angepasst werden

```
public class CreateCustomerRequest
{
    [Required(ErrorMessage = "Name is required")]
    public string Name { get; set; }

    [Range(0, 100, ErrorMessage = "Age must be between 0 and 100")]
    public int Age { get; set; }

    [EmailAddress(ErrorMessage = "Invalid email address")]
    public string Email { get; set; }

    [Phone(ErrorMessage = "Invalid phone number")]
    public string PhoneNumber { get; set; }
}
```

AUTOMATISCHE VALIDIERUNG

- In API-Controllern die das [ApiController] Attribut haben, werden Modelle automatisch validiert
- Bei ungültigen Modellen wird automatisch ein BadRequest zurückgegeben
- Die Antwort besteht aus einer einheitlichen Fehlermeldung in Form eines JSON strings

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "Name": [
      "Name is required"
    ]
  },
  "traceId": "00-bfa88e4e6bf4267029bae1ae00dba1f2-f7a8471299fef8ee-00"
}
```

AUTOMATISCHE VALIDIERUNG

```
[ApiController]
[Route("[controller]")]
public class TestController : ControllerBase
{
    [HttpPost]
    public IActionResult Create([FromBody] Test test)
    {
        return Ok(test);
    }
}

public class Test
{
    public int Id { get; set; }

    [Required(ErrorMessage = "Name is required")]
    public string Name { get; set; }
}
```

CUSTOM VALIDATION

- Custom Validation kann genutzt werden, wenn Standard-Validierungsattribute nicht ausreichen
- Dafür wird eine benutzerdefinierte Validationsattribut-Klasse erstellt, die von ValidationAttribute erbt
- In dieser Klasse wird die IsValid Methode überschrieben

```
public class DateInTheFutureAttribute : ValidationAttribute
{
    public override bool IsValid(object? value)
    {
        if (value is DateTime dateTime) {
            return dateTime > DateTime.UtcNow;
        }
        return false;
    }
}

[DateInTheFuture(ErrorMessage = "The event date must be in the future")]
public DateTime EventDate { get; set; }
```

CUSTOM VALIDATION

- Wenn eine Validierung erforderlich ist, die z.B. mehrere Felder im Modell betrifft, oder wenn ein Service aus dem Dependency Injection Container zur Validierung benötigt wird, kann das IValidateObject implementiert werden
- IValidatableObject ermöglicht die Validierung auf Modellebene
- Die Test-Klasse implementiert IValidatableObject, um benutzerdefinierte Validierungslogik zu integrieren, die sicherstellt, dass Name und EventDate valide Werte haben

CUSTOM VALIDATION

```
public class Test : IValidatableObject
{
    public int Id { get; set; }

    public string Name { get; set; }

    public DateTime EventDate { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if(string.IsNullOrWhiteSpace(Name)) {
            yield return new ValidationResult("Name is required", [nameof(Name)]);
        }

        if (EventDate < DateTime.UtcNow)
        {
            yield return new ValidationResult("The event date must be in the future", [nameof(EventDate)]);
        }
    }
}
```

REIHENFOLGE DER VALIDIERUNGEN

- **Attributbasierte Validierung**
- Die Validierung basierend auf den Attributen in den Modellklassen wird zuerst ausgeführt
- Dies umfasst Validierungsattribute wie [Required], [StringLength], [Range] usw.

- **IValidatableObject-Validierung**
- Nachdem die Attributvalidierung abgeschlossen ist, wird die Validate-Methode des IValidatableObject-Interfaces aufgerufen, falls das Modell dieses Interface implementiert

- Diese Reihenfolge gewährleistet, dass zunächst die einfachen Validierungsregeln auf Eigenschaftsebene durchgeführt werden (Attributvalidierung), gefolgt von komplexeren Validierungen auf Modell- oder Objektebene (IValidatableObject)