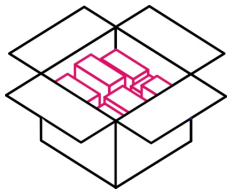
Eingabevalidierung in Spring

Hoai Viet Nguyen – TH Köln

Technology
Arts Sciences
TH Köln



Eingabevalidierung mit HTML

```
<label>Email</label>
<input name="email" minlength="5" maxlength="50" type="email"><br>
<label>Birthday</label>
<input type="date" name="birthday"><br>
<label>Gender</label>
<select name="gender">
    <option value="MALE">Male</option>
    <option value="FEMALE">Female</option>
    <option value="DIVERS">Divers</option>
</select><br>
<button>Create user</button>
```

Herausforderungen Eingabevalidierung im Client/Webbrowser

- Eingabevalidierung im Client nur für Gebrauchstauglichkeit gedacht
- Keine Kontrolle über den Client
- Angreifer können beliebigen Client verwenden und Validierung umgehen

Jakarta Bean Validation API in Spring

- Serverseitige Eingabevalidierung durch Annotationen bzw. Constraints
- Validierungen werden zur Laufzeit durchgeführt
- Es gibt ein Satz von vordefinierte Constraints
- Entwickler können selber Constraints definieren

Jakarta Bean Constraints (Auszug)

- `@NotNull`: **Wert darf nicht null werden**
- `@AssertTrue`: **Wert dart nur true sein**
- `@Size`: **Definiert einen maximal und minimal Größe des Wertes (String, Collection, Listen, Map)**
- `@Min`: **Definiert minimal Größe**
- `@Max`: **Definiert maximal Größe**
- `@Email`: **Validiert, ob es sich, um eine gültige Email handelt**
- `@NotEmpty`: **Wert darf nicht null und leer sein**
- `@NotBlank`: **Wert darf nicht nur aus Leerzeichen bestehen**
- `@Positive`, `@PositiveOrZero`: **Wert muss positive bzw. mindestens größer gleich null sein**
- `@Negative`, `@NegativeOrZero`: **Wert muss negative bzw. mindestens kleiner gleich null sein**
- `@Past`, `@PastOrPresent`: **Datum darf in der Vergangenheit sein**
- `@Future`, `@FutureOrPresent`: **Datum muss in der Zukunft sein**

Whitelisting in Spring mit Enum-Klassen

```
class User {
    …
    var gender: Gender? = null
}
```

```
enum class Gender {
    MALE, FEMALE, DIVERS
}
```

Serverseitige Eingabevalidierung: Welche Variante ist am sinnvollsten?

A

```
@PostMapping("/users")
@ResponseStatus(HttpStatus.CREATED)
fun saveUser(
    @Size(min=5, max=50) @Email email: String,
    gender: Gender,
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @PastOrPresent birthday: Date): User {
    var user = User()
    user.email = email
    user.gender = gender
    user.birthday = birthday
    usersService.saveUser(user)
    return user
}
```

B

```
class User {
    @Id
    val id : UUID = UUID.randomUUID()
    @Size(min=5, max=50) @Email
    var email: String = ""
    var gender: Gender? = null
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @PastOrPresent
    var birthday: Date? = null
}
```

```
@PostMapping("/users")
@ResponseStatus(HttpStatus.CREATED)
fun saveUser(@Valid @RequestBody user: User)
: User {
    usersService.saveUser(user)
    return user
}
```

C

```
class UserDto {
    @Size(min = 5, max = 50)
    @Email
    var email: String = ""
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    @PastOrPresent
    var birthday: Date? = null
    var gender: Gender? = null
}
```

```
@PostMapping("/users")
@ResponseStatus(HttpStatus.CREATED)
fun saveUser(@Valid @RequestBody userDto: UserDto)
: User {
    var user = User()
    user.birthday = userDto.birthday
    user.email = userDto.email
    user.gender = userDto.gender
    usersService.saveUser(user)
    return user
}
```

DTO-Ansatz

- Vorteile
  - Trennung von Model/Entity und Validierung (Single Responsibility)
  - Wiederverwendbarkeit in anderen Controller-Methoden
  - Client kann nur die Eigenschaften im DTO beeinflussen
- Nachteile
  - DTO-Klasse muss zusätzlich erstellt werden
  - Doppelter Wartungsaufwand

Lernzielkontrolle

- Was sind die Herausforderungen bei Eingabevalidierung auf dem Webbrowser/Client?
- Was ist die Jakarta Bean Validation API?
- Wie lässt sich Whitelisting in Spring implementieren?
- Warum ist die Eingabevalidierung im einem DTO sinnvoll?

Zusammenfassung

- Eingabevalidierung im Webbrowser kann einfach umgangen werden, indem Angreifer einen eigenen Client verwenden
- Jakarta Bean Validation API ermöglich Server-seitige Eingabevalidierung
- Whitelisting in Spring lässt sich mit Enum-Klassen realisieren
- Eingabevalidierung sollte im DTO durchgeführt, um die Verantwortlichkeiten zu trennen und zu definieren welche Eigenschaften in einem Entity/Model geändert werden dürfen