Валидация сущностей

- Описание валидации
- Автоматическая валидация

Валидация – это механизм проверки данных объекта на корректность перед их сохранением в базу данных. Валидация реализуется набором аннотаций Jakarta Bean Validation, которые обычно применяются к сущностям и DTO. В этом уроке мы поговорим о том, как добавить правила валидации и как выполнить саму валидацию.

Описание валидации

Возьмем для примера модель `User` с добавленными аннотациями для валидации:

```
package io.hexlet.spring.model;

// Остальные импорты
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

@Entity
@Getter
@Setter
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Long id;

    @Column(unique = true)
    @Email
    private String email;

    @NotBlank
    private String firstName;

    @NotNull
```

```
    @Size(min = 8)
    private String password;
}
```

В этом примере:

- `@Email` проверяет, что `email` содержит корректный адрес
- `@NotBlank` проверяет, что `firstName` содержит хотя бы один цифро-буквенный символ
- `@NotNull` проверяет, что `password` не пустой
- `@Size` проверяет, что минимальная длина пароля составляет восемь символов

Есть и множество других аннотаций. Заранее знать их не нужно, но имеет смысл периодически просматривать их список в спецификации.

Автоматическая валидация

Валидация выполняется с помощью аннотации `@Valid`, которая применяется в контроллере:

```
package io.hexlet.spring.controller.api;

// Остальные импорты
import jakarta.validation.Valid;

@RestController
@RequestMapping("/api")
public class UsersController {
    @Autowired
    private UserRepository repository;

    @Autowired
    private UserMapper userMapper;

    @PostMapping("/users")
    @ResponseStatus(HttpStatus.CREATED)
    // Валидация происходит до вызова метода
    public UserDTO create(@Valid @RequestBody User user) {
        // Логика создания
    }
}
```

Аннотация `@Valid` идет в паре с `@RequestBody`. Сама валидация вызывается уже на получившемся объекте, в нашем примере — это `user`. При успешной валидации вызывается метод контроллера, при неуспешной — возникает исключение `MethodArgumentNotValidException`. Spring Boot обрабатывает это исключение автоматически и возвращает ошибку *400 Bad Request*:


```
{
    "timestamp": 1695940603767,
    "status": 400,
    "error": "Bad Request",
    "message": "Validation failed for object='user'. Error count: 2",
    "errors": [
        {
            "codes": [
                "NotNull.user.firstName",
                "NotNull.firstName",
                "NotNull"
            ],
            "arguments": [
                {
                    "codes": [
                        "user.firstName",
                        "firstName"
                    ],
                    "defaultMessage": "firstName",
                    "code": "firstName"
                }
            ],
            "defaultMessage": "must not be null",
            "objectName": "user",
            "field": "firstName",
            "bindingFailure": false,
            "code": "NotNull"
        },
        {
            "codes": [
                "NotNull.user.slug",
                "NotNull.slug",
                "NotNull"
            ],
            "arguments": [
```

```
                {
                    "codes": [
                        "user.email",
                        "email"
                    ],
                    "defaultMessage": "email",
                    "code": "email"
                }
            ],
            "defaultMessage": "must not be null",
            "objectName": "user",
            "field": "email",
            "bindingFailure": false,
            "code": "NotNull"
        }
    ],
    "path": "/api/users/1"
}
```


В примере выше обратите внимание на тело запроса. Здесь мы используем для него саму сущность, но в реальном коде там почти наверняка будет использоваться DTO. В этом случае нам придется дополнительно повесить валидацию на DTO:

```
package io.hexlet.spring.dto;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class UserCreateDTO {
    @Email
    private String email;

    @NotBlank
    private String firstName;
```

```
    @NotNull
    @Size(min = 8)
    private String password;
}
```

Теперь мы можем заменить сущность на DTO:

```
public UserDTO create(@Valid @RequestBody UserCreateDTO user) {
    // Код
}
```

На выходе мы получаем аннотации, добавленные в саму сущность и в ее DTO, используемые для создания или обновления. Это неизбежно приводит к дублированию аннотаций, что придется регулярно делать в реальном коде.

Далее →