04

유효성 검증

01. 유효성 검사/데이터 검증

- 서비스의 비즈니스 로직을 올바르게 동작하게 하기 위해 사용되는 데이터에 대한 사전 검증을 하는 작업
- 유효성 검사 또는 데이터 검증(Validation)이라고 함
- 데이터 검증은 여러 Layer에서 발생
- 들어오는 데이터에 대해 의도한 형식의 값이 제대로 들어오는지 체크하는 과정
- Dependency를 추가해야함
- implementation 'org.springframework.boot:spring-boot-starter-validation'

02. Validation Annotaion

- @Size : 문자의 길이 조건
  - @Size(min=10, max=20)
- @NotNull : Null 값 불가
- @NotEmpty : @NotNull + “” 불가
- @NotBlank : @NotEmpty + “ “불가
- @Max : 최대값 조건 설정
  - @Max(value=100)
- @Min : 최소값 조건 설정
- @Past /@PastOrPresent : 과거 날짜/ 현재포함과거날짜
- @Future/@FutureOrPresent

03. DTO 클래스에 검증 Annotation 설정

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ProductDTO {
    private Integer id;
    @NotEmpty(message="이름은  비어있을수  없습니다")
    private String title;
    private String imgsrc;
    private int price;
}
```

04. Controller에서 검증 설정

```
@PostMapping(value="/new-product")
public ResponseEntity<ProductDTO> newProduct(@Valid
@RequestBody ProductDTO productDTO){
    ProductDTO
product=this.productService.saveProduct(productDTO);
    return
ResponseEntity.status(HttpStatus.OK).body(product);
}
```

05

예외처리

01. Thymeleaf 설치 시

error.html

- 서버에서 에러가 발생하면 그 페이지로 자동으로 이동

(error.html)

```
<div>에러페이지임</div>
<p th:text="${status}"></p>
<p th:text="${error}"></p>
<p th:text="${path}"></p>
<p th:text="${message}"></p>
<p th:text="${exception}"></p>
```

02. ResponseEntity<T>

REST API 설계 시 상태 코드와 메시지를 Client에 전송

상태 코드

- Client Error : 400번대
- Server Error : 500번대
- 정상 : 200번대

```
@GetMapping("/detail/{id}")
ResponseEntity<String> detail() {
    try {
        throw new Exception("이런저런에러");
    } catch(Exception e){
        return ResponseEntity.status(에러코드).body("에러이유");
    }
}
```

ResponseEntity.status(HttpStatus.NOT_FOUND).body("에러남");

03. @ControllerAdvice & @RestControllerAdvice

@ControllerAdvice

- 전역 예외 처리를 구현
- 컨트롤러에서 발생하는 예외를 한 곳에서 처리
- 특정 Controller를 지정할 수도 있음
  - @ControllerAdvice(basePackages = "com.example.controller")
- @ResponseBody와 함께 사용하여 Json데이터를 오류데이터로 전달 가능

@RestControllerAdvice

- @ControllerAdvice 와 @ResponseBody를 합친 annotaion

03. @ControllerAdvice & @RestControllerAdvice

```
@ControllerAdvice
public class MyExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("에러남");
    }

    @ExceptionHandler(MethodArgumentTypeMismatchException.class)
    public ResponseEntity<String> handleException3(Exception e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("에러남");
    }
}
```

04. MethodArgumentNotValidException

- 유효성 검사 실패 시 발생하는 예외
  - @Valid를 사용하여 요청 데이터의 유효성을 검사할 때 유효하지 않은 값이 전달되면 발생.
- 주로 요청 데이터의 필드 오류를 반환

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<Map<String, String>>
handleValidationExceptions(MethodArgumentNotValidException ex) {
    Map<String, String> errors = new HashMap<>();
    ex.getBindingResult().getFieldErrors().forEach(error ->
            errors.put(error.getField(), error.getDefaultMessage())
    );
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
}
```

05. HttpMessageNotReadableException

JSON 형식 오류나 요청 본문 파싱 오류를 처리

```
@ExceptionHandler(HttpMessageNotReadableException.class)
public ResponseEntity<String>
handleHttpMessageNotReadableException(HttpMessageNotReadableExcept
ion ex) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST)
            .body("Invalid request body: " + ex.getMessage());
}
```

06. EntityNotFoundException

- JPA 엔티티를 찾지 못했을 때 발생하는 예외를 처리

```
@ExceptionHandler(EntityNotFoundException.class)
public ResponseEntity<String>
handleEntityNotFoundException(EntityNotFoundException ex) {
    return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .body("Resource not found: " + ex.getMessage());
}
```

- JPA의 find(), findById(), deleteById() 메소드는 기본적으로 데이터가 없을 경우 *발생하는 것이 아님*
- repository.findById().get()을 했을때 repository.findById()가 Entity객체를 찾지 못했는데 get()을 호출했을 때 발생

07. MethodArgumentTypeMismatchException

- 컨트롤러 메소드의 매개변수 타입과 클라이언트가 전달한 요청 값의 타입이 일치하지 않을 때 발생

```
@ExceptionHandler(MethodArgumentTypeMismatchException.class)
public ResponseEntity<String>
handleMethodArgumentTypeMismatch(MethodArgumentTypeMismatchException
ex) {
    String errorMessage = String.format(
            "Invalid argument: '%s'. Expected type: '%s'.",
            ex.getValue(),
            ex.getRequiredType().getSimpleName()
    );
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errorMessage);
}
```