스프링부트로 RestFulAPI 구현하기

7장 에러코드관리하기

- @ControllerAdvice
- ErrorCode
- ResponseEntity

박명회

7장 HTTP CODE

https://developer.mozilla.org/ko/docs/Web/HTTP/Status

1xx (Informational): 요청이 받았으며 처리 중입니다. 일반적으로 사용되지 않습니다.
2xx (Success): 요청이 성공적으로 처리되었습니다.
200 (OK): 요청이 성공했으며 데이터가 반환되었습니다.
201 (Created): 요청이 성공적으로 처리되었고 새로운 리소스가 생성되었습니다.
204 (No Content): 요청이 성공했지만 반환할 콘텐츠가 없습니다.
3xx (Redirection): 추가 조치가 필요합니다. 클라이언트는 추가 작업을 수행해야 합니다.
4xx (Client Error): 클라이언트의 요청이 잘못되었습니다.
400 (Bad Request): 서버가 요청을 이해하지 못했습니다.
401 (Unauthorized): 인증이 필요합니다.
404 (Not Found): 요청한 리소스를 찾을 수 없습니다.
403 (Forbidden): 요청이 서버에서 거부되었습니다.
5xx (Server Error): 서버에서 오류가 발생하여 요청을 처리할 수 없습니다.
500 (Internal Server Error): 서버에서 요청을 처리하는 중에 오류가 발생했습니다.

HTTP 상태 코드는 클라이언트에게 요청 상태에 대한 정보를 전달하여 적절한 조치를 취할 수 있도록 합니다. 예를 들어, 성공적인 요청에 대한 응답으로는 200 상태 코드를 받으며, 오류가 발생한 경우에는 해당 오류를 나타내는 상태 코드를 받습니다.

7장 ResponseEntity

```
@GetMapping("select/{id}")
public ResponseEntity<User> select(@PathVariable Long id){
    User user = userService.findById(id);
    if (user != null) {
        return ResponseEntity.ok(user);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

notFound 사용시
HTTP 404 상태코드

```
@GetMapping("selectall")
public ResponseEntity<List<User>> selectall(){
    List<User> list = userService.findAll();
    if (list != null && !list.isEmpty()) {
        return ResponseEntity.ok(list);
    } else {
        return ResponseEntity.noContent().build();
    }
}
```

noContent 사용시
HTTP 204 상태 코드

7장 ResponseEntity

```
@PostMapping("insert")
public ResponseEntity<Void> insert(@RequestBody UserReq userReq){
    userService.saveUser(userReq);
    return ResponseEntity.status(HttpStatus.CREATED).build();
}
```

CREATED 사용시
HTTP 201 상태코드

```
@PutMapping("update")
public ResponseEntity<Void> update(@RequestBody UserReq userReq){
    userService.saveUser(userReq);
    return ResponseEntity.ok().build();
}
```

수정시에는 필요에 따라서
수정된 유저**entity**를
보여준다.

```
@PutMapping("update")
public ResponseEntity<User> update(@RequestBody UserReq userReq){
    User updatedUser = userService.saveUser(userReq);
    return ResponseEntity.ok(updatedUser);
}
```

7장 ResponseEntity

```
@DeleteMapping("delete/{id}")
public ResponseEntity<Void> delete(@PathVariable Long id){
    userService.deleteUser(id);
    return ResponseEntity.noContent().build();
}
```

noContent 사용시
HTTP 204 상태코드

7장 @ControllerAdvice

```
@ControllerAdvice
public class GlobalExceptionHandler {


    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<?> handleUserNotFoundException(UserNotFoundException ex, WebRequest request) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
    }


}
```

7장 UserNotFoundException 정의하기

```
public class UserNotFoundException extends RuntimeException {
    public UserNotFoundException(Long id) {
        super("User not found with id: " + id);
    }
}
```

7장 유저 삭제시에 Exception 발생

```
public void deleteUser(Long id) {
    User user = userRepository.findById(id).orElseThrow(() -> new UserNotFoundException(id));
    userRepository.delete(user);
}
```

7장 에러코드 enum

```
public enum ErrorCode {
    USER_NOT_FOUND("USER_NOT_FOUND", "User not found with id: %d"),
    INVALID_REQUEST("INVALID_REQUEST", "Invalid request");

    private final String code;
    private final String message;

    ErrorCode(String code, String message) {
        this.code = code;
        this.message = message;
    }

    public String getCode() {
        return code;
    }

    public String getMessage(Object... args) {
        return String.format(message, args);
    }
}
```

7장 CustomException

```
public class CustomException extends RuntimeException {
    private final ErrorCode errorCode;

    public CustomException(ErrorCode errorCode, Object... args) {
        super(errorCode.getMessage(args));
        this.errorCode = errorCode;
    }

    public ErrorCode getErrorCode() {
        return errorCode;
    }
}
```

7장 GlobalExceptionHandler CustomException 추가하기

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

import java.time.LocalDateTime;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(CustomException.class)
    public ResponseEntity<ErrorResponse> handleCustomException(CustomException ex,
WebRequest request) {
        ErrorResponse errorResponse = new ErrorResponse(ex.getErrorCode(), ex.getMessage(),
LocalDateTime.now());
        return new ResponseEntity<>(errorResponse, HttpStatus.NOT_FOUND);
    }
}
```

7장 에러발생시간 추가하기

```
@Getter
@Setter
@ToString
public class ErrorResponse {
    private String errorCode;
    private String message;
    private LocalDateTime timestamp;

    public ErrorResponse(ErrorCode errorCode, String message, LocalDateTime timestamp) {
        this.errorCode = errorCode.getCode();
        this.message = message;
        this.timestamp = timestamp;
    }

 }
```

7장 GlobalExceptionHandler CustomException 시간추가하기

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

import java.time.LocalDateTime;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(CustomException.class)
    public ResponseEntity<ErrorResponse> handleCustomException(CustomException ex,
WebRequest request) {
        ErrorResponse errorResponse = new ErrorResponse(ex.getErrorCode(), ex.getMessage(),
LocalDateTime.now());
        return new ResponseEntity<>(errorResponse, HttpStatus.NOT_FOUND);
    }
}
```

```
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;

public class UserReqDto {

    @NotBlank(message = "Username is mandatory")
    @Size(min = 3, max = 20, message = "Username must be between 3 and 20 characters")
    private String username;

    @NotBlank(message = "Email is mandatory")
    @Size(max = 50, message = "Email must be less than 50 characters")
    private String email;

    // Other fields and getters/setters
}
```

```
public enum ErrorCode {
    USER_NOT_FOUND("USER_NOT_FOUND", "User not found with id: %d"),
    INVALID_REQUEST("INVALID_REQUEST", "Invalid request"),
    VALIDATION_ERROR("VALIDATION_ERROR", "Validation error: %s");

    private final String code;
    private final String message;

    ErrorCode(String code, String message) {
        this.code = code;
        this.message = message;
    }

    public String getCode() {
        return code;
    }

    public String getMessage(Object... args) {
        return String.format(message, args);
    }
}
```

7장 유효성검사

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<ErrorResponse>
        handleValidationException(MethodArgumentNotValidException ex) {
        String errors = ex.getBindingResult().getAllErrors().stream()
                        .map(error -> error.getDefaultMessage())
                        .collect(Collectors.joining(", "));

        ErrorResponse errorResponse = new ErrorResponse(ErrorCode.VALIDATION_ERROR, errors,
LocalDateTime.now());
        return new ResponseEntity<>(errorResponse, HttpStatus.BAD_REQUEST);
}
```

7장 @**Valid** 추가하기

```
@PostMapping("insert")
public ResponseEntity<Void> insert(@Valid @RequestBody UserReqDto userReqDto){
    userService.saveUser(userReqDto);
    return ResponseEntity.status(HttpStatus.CREATED).build();
}

@PutMapping("update")
public ResponseEntity<Void> update(@Valid @RequestBody UserReqDto userReqDto){
    userService.saveUser(userReqDto);
    return ResponseEntity.ok().build();
}

@DeleteMapping("delete/{id}")
public ResponseEntity<Void> delete(@PathVariable Long id){
    userService.deleteUser(id);
    return ResponseEntity.noContent().build();
}
```

감사합니다