



개발자가 만드는 예외처리 - 2

객체가 맡은 역할

선생님이 정리해준 각 객체 총 정리 (정리하기)

GetMapping (users/{id})

오류발생

해결

로직 오류 상황 및 문제점 발생

문제점을 해결하기 위한 로직

- 1 개발자가 정한 [에러 메시지](#)로 예외처리
- 2 개발자가 만든 [오류코드](#)로 에러내용 보내기
- 3 문자열 뿐만 아니라 [에러코드와 시간도 추가해서 응답](#)
- 4 [에러창고](#)를 활용하여 오류에 대한 내용 응답

결론

PutMapping(users)

기초작업

데이터 유효성 검사 추가

요청한 값을 외부에서 확인할 수 있도록 처리

DB 조작 기능(수정) 추가

기본작업

[예외처리를 통해 정상적인 로직으로 수정](#)

[sql 파일을 통해 DB에 값 미리 넣기](#)

PutMapping 진행

객체가 맡은 역할

Controller -> 주소 맵핑

Service -> 기능구현..

regist() .. 해당하는 USER 검색 by email
USER 저장..

Repository -> JPA

insert select update delete

Entity

-> DB Table 설계

Dto

-> 프론트쪽에서 넘어오는 파라메타 설계

```
no usages -> @JsonIgnore
@PostMapping("users")
public ResponseEntity<User> addUser(@RequestBody @Valid UserDTO userDTO) {
    // 수정된 네번째 방식 ( 코드가 더욱 간결해짐 )
    userDTO.setWdate(LocalDateTime.now());

    ModelMapper mapper = new ModelMapper();

    User user = mapper.map(userDTO, User.class);
    User dbUser = service.regist(user);

    return ResponseEntity.status(HttpStatus.CREATED).body(dbUser);
}
```

ModelMapper.map(매개변수1, 매개변수2)
→ 매개변수1의 값을 매개변수2로 전달한다는 의미

▼ 선생님이 정리해준 각 객체 총 정리 (정리하기)

MVC 기본설정

Controller -> 주소 매핑

Service -> 기능구현..

regist() .. 해당하는 USER 검색 by email

USER 저장..

Repository -> JPA

insert select update delete

Entity

-> DB Table 설계

Dto

-> 프론트쪽에서 넘어오는 파라메타 설계

-> DTO -> ENTITY

Builder of 함수를 호출해서

BeanUtils.copyproperties() 속성값복사

ModelMapper속성값 복사

<dependency>

<groupId>org.modelmapper</groupId>

<artifactId>modelmapper</artifactId>

<version>3.2.0</version>

</dependency>

프론트 쪽에서 데이터인 DTO 클래스를 넘겨줄때... 유효성검사를 도와주는 클래스

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

DTO 클래스 안에... NotNull Max Min

=====

프론트 쪽에서 데이터인 DTO 클래스를 넘겨줄때... 유효성검사를 도와주는 클래스

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-validation</artifactId>
```

```
</dependency>
```

DTO 클래스 안에... NotNull Max Min

=====

@ControllerAdvice

AOP 활용한...

예외 발생했을때에 프론트 쪽에 어떤 결과를 보내는 역할..

ErrorResponse 객체 생성

ErrorCode 있는 내용으로 설정해서 보냅니다.

=====

CorsConfig --> 다른 504호 PC 에서 404호 서버에 요청시에

크롬브라우저 허용하지않는데... CORS

설정&서버응답하기

```
@Override
```

```
public void addCorsMappings(CorsRegistry registry) {
```

```
    registry.addMapping("/**")
```

```
    .allowedOrigins("*")
```

```
    .allowedMethods("GET", "POST", "PUT", "DELETE")
```

```
    .allowedHeaders("*");
```

```
}
```



MVC 기본설정



이해...(듣는거) 외우기...(적어보기) 응용...(적어보기)



=====

Controller -> 주소 매핑
Service -> 기능구현..
regist() .. 해당하는 USER 검색 by email
USER 저장..
Repository -> JPA
insert select update delete
Entity
-> DB Table 설계
Dto
-> 프론트쪽에서 넘어오는 파라메타 설계



> DTO -> ENTITY
Builder of 함수를 호출해서
BeanUtils.copyproperties() 속성값복사
ModelMapper속성값 복사
<dependency>
<groupId>org.modelmapper</groupId>
<artifactId>modelmapper</artifactId>
<version>3.2.0</version>
</dependency>



프론트 쪽에서 데이터인 DTO 클래스를 넘겨줄때... 유효성검사를 도와주는 클래스
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
DTO 클래스 안에... NotNull Max Min



@ControllerAdvice
AOP 활용한..
예외 발생했을때에 프론트 쪽에 어떤 결과를 보내는 역할..
ErrorResponse 객체 생성
ErrorCode 있는 내용으로 설정해서 보냅니다.



=====
CorsConfig --> 다른 504호 PC 에서 404호 서버에 요청시에
크롬브라우저 허용하지않는데... CORS

```
@Override
public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
        .allowedOrigins("
        ")
        .allowedMethods("GET","POST","PUT","DELETE")
        .allowedHeaders("
        ");
}
```

GetMapping (users/{id})

```
@GetMapping("users/{id}")
public ResponseEntity<String> getUserById(@PathVariable Long id) {
    System.out.println(id);
    return ResponseEntity.status(HttpStatus.OK).body("SUCCESS");
}
```


오류발생

```
Caused by: java.security.PublicKeyRetrievalException: Public Key Retrieval is not allowed
    at java.security.PublicKeyRetrievalException.SQLException(java:111) ~[mysql-connecto
    at java.sql.SQLException.translateException(SQLExceptionsMapping.java:111) ~[mysql-connecto
```

해결

[Mysql] Public key retrieval is not allowed 에러 해결

Public key retrieval is not allowed 에러가 발생하는 이유와 해결하는 방법에
대해서 소개해드립니다.

 <https://deeplify.dev/database/troubleshoot/public-key-retrieval-is-not-allowed>



- properties에서 `spring.datasource.url` 파라미터 수정 해줌

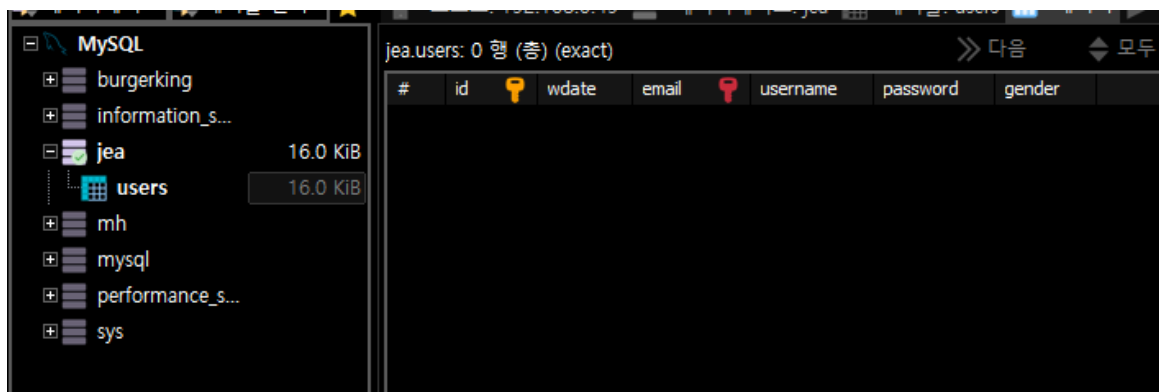
```
spring.datasource.url=
    jdbc:mysql://192.168.0.49:3306/jea?useSSL=false&
```

로직 오류 상황 및 문제점 발생

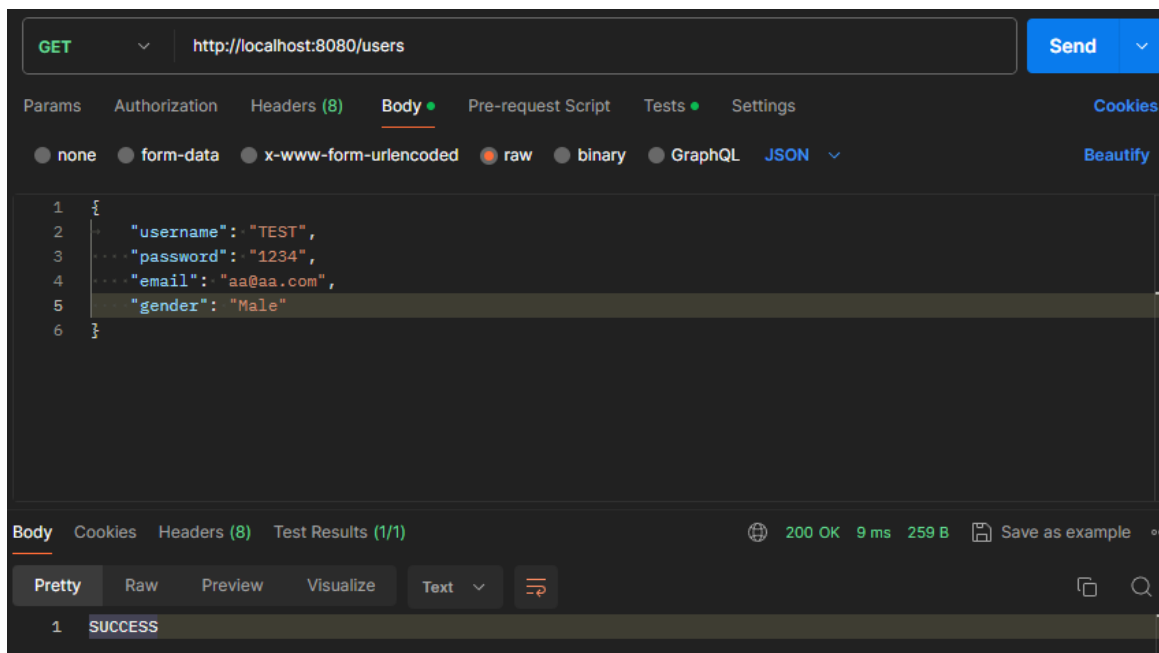
- 반환 값으로 200 코드를 보내는 HttpStatus.OK을 사용해주었고, 그에 대한 예러 메시지는 SUCCESS이다.

```
@GetMapping("users/{id}")
public ResponseEntity<String> getUserById(@PathVariable Long id)
    System.out.println("id >>>> " + id);
    return ResponseEntity.status(HttpStatus.OK).body("SUCCESS");
}
```

- 실행 결과
 - 실행됨과 동시에 users 테이블이 생성된다. 단, 데이터는 없는 상태이다.



- 반환값을 무조건 200 에러코드와 SUCCESS 문자열을 보내고 있어서 **예외처리가 정상적으로 되어있지 않은 로직이라는 점을 확인할 수 있다.**



문제점을 해결하기 위한 로직

1 개발자가 정한 에러 메시지로 예외처리

오류코드 키워드를 정해준 후,
내가 정한 문자열로 에러 내용을 만들어 응답해줄 수 있다.

- C 수정

```
@GetMapping("users/{id}")
public ResponseEntity<User> getUserById(@PathVariable Long id) {
    System.out.println("id >>>> " + id);
    User user = service.getUserById(id);
    return ResponseEntity.status(HttpStatus.OK).body(user);
}
```

- M 생성

```
// 특정 id 조회
public User getUserById(Long id) {
    // id를 기준으로 데이터를 조회한다.
    Optional<User> user = userRepository.findById(id);
```


- 실행 결과
 - 내가 만든 에러에 대한 문자열은 출력되고 있다.

! NOT USER 뒤에 있는 내용들은 프로그램 실행에 있어 있어선 안되는 내용들이다.



내가 예외처리에 대한 설정을 직접 만들어서 사용하겠다!

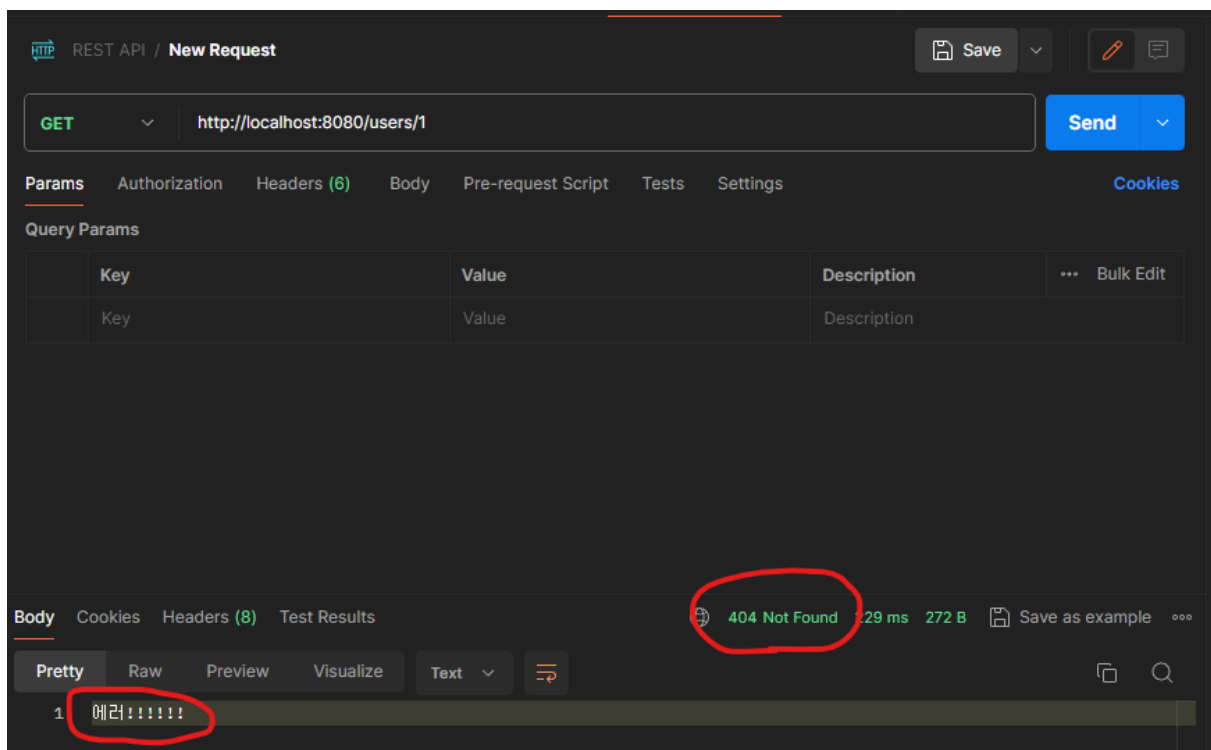
예를 들어, 404

NOT_FOUND 예외처리가 발생할 때 ERROR라는 문자열을 응답해줄 수 있는 방식.

- GlobalExceptionHandler → handlerLogException 생성
 - 클래스명 변경 : UserExceptionHandler → GlobalExceptionHandler
 - Exception을 통해 모든 예외를 받을 수 있다.
 - 예외코드 중 404인 NOT_FOUND가 나올 경우 에러!!!!!!라는 문자열을 응답해줄 것이다.

```
@ExceptionHandler(Exception.class)
public final ResponseEntity<String> handlerLogException(Exception e) {
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("에러!!!");
}
```

- 실행 결과



정상 실행

3 문자열 뿐만 아니라 에러코드와 시간도 추가해서 응답

솔직히 에러코드만 보내면 어떤 부분이 오류가 났는지 모른다.
외부에게 어떤 에러가 생겼는지
정확하게 안내해주기 위한 작업을 진행하자!

- GlobalExceptionHandler → `handlerLogException` 수정

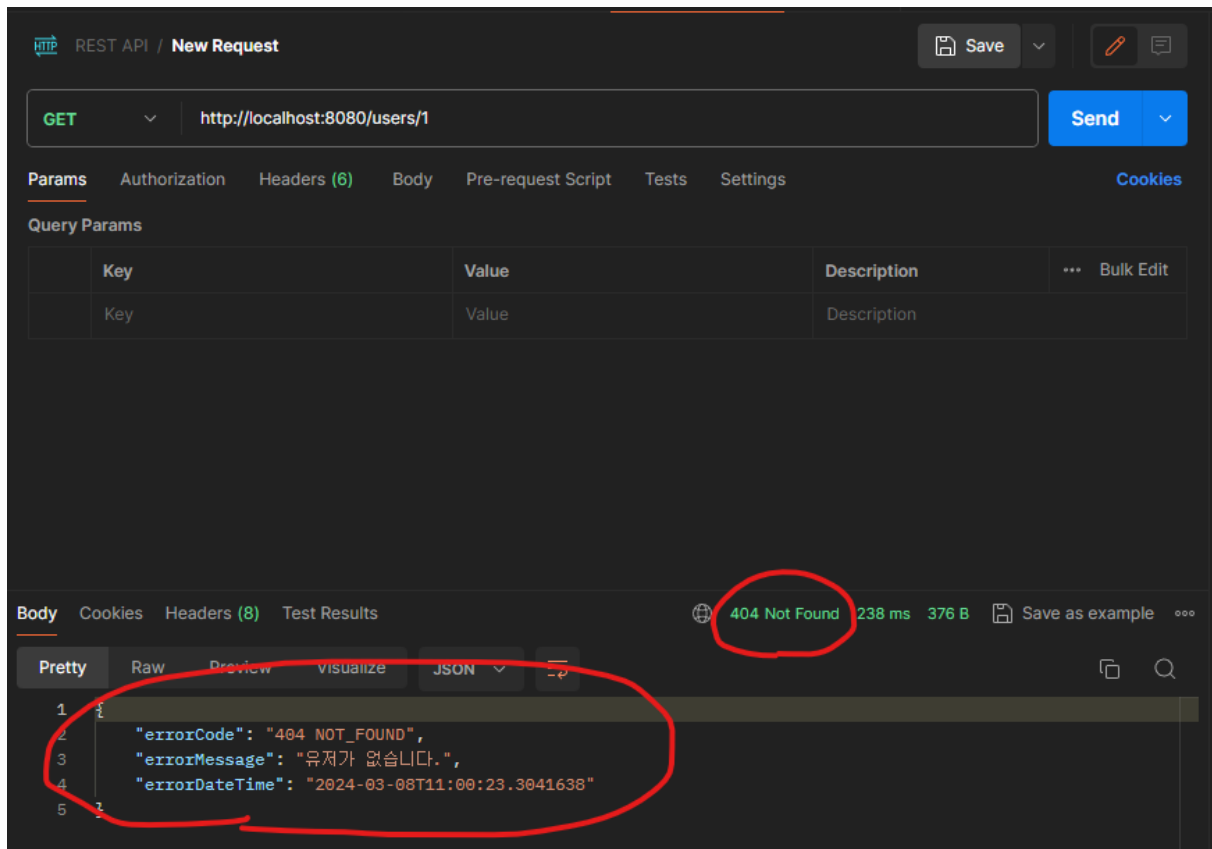
▼ ErrorResponse ? (열람 가능)

```
@Getter
@Builder
public class ErrorResponse {
    private String errorCode;
    private String errorMessage;
    private LocalDateTime errorDateTime;
}
```

```
@ExceptionHandler(Exception.class)
public final ResponseEntity<ErrorResponse> handlerLogException(Exception e) {
    ErrorResponse errorResponse = ErrorResponse.builder()
        .errorCode(HttpStatus.NOT_FOUND)
        .errorMessage("유저가 요청한 리소스를 찾을 수 없습니다.")
        .errorDateTime(LocalDateTime.now())
        .build();

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorResponse);
}
```

- 실행 결과



✅ ErrorResponse 참조 변수에 담은 값들이 정상적으로 출력되는 점 확인할 수 있다.

4 예러창고를 활용하여 오류에 대한 내용 응답

예러창고를 사용해서 예외처리 내용을 응답해주자!
위의 방법들로 예외처리를 진행하면 불필요한 로직이 많아진다.

👉 내가 말하는 예러창고는 `enum` 타입의 `ErrorCode` 객체이다.

- 객체 생성

👋 잠깐! 한 객체로 예외처리를 관리하면 로직이 많아지고 복잡해진다.

User를 담당하는 객체를 만들어서 따로 따로 관리하도록 하자!

- UserException

```

@Getter
public class UserException extends RuntimeException{
    private ErrorCode errorCode;

    public UserException(ErrorCode errorCode) {
        super(errorCode.getMessage());
        this.errorCode = errorCode;
    }
}

```

- 객체 수정

- M → `getUserById` 수정

```

public User getUserById(Long id) {
    Optional<User> user = userRepository.findById(id);

    if(user.isEmpty()) {
        throw new UserException(ErrorCode.NOTFOUND);
    } else {
        return user.get();
    }
}

```

- `GlobalExceptionHandler` → `handlerLogException` 수정

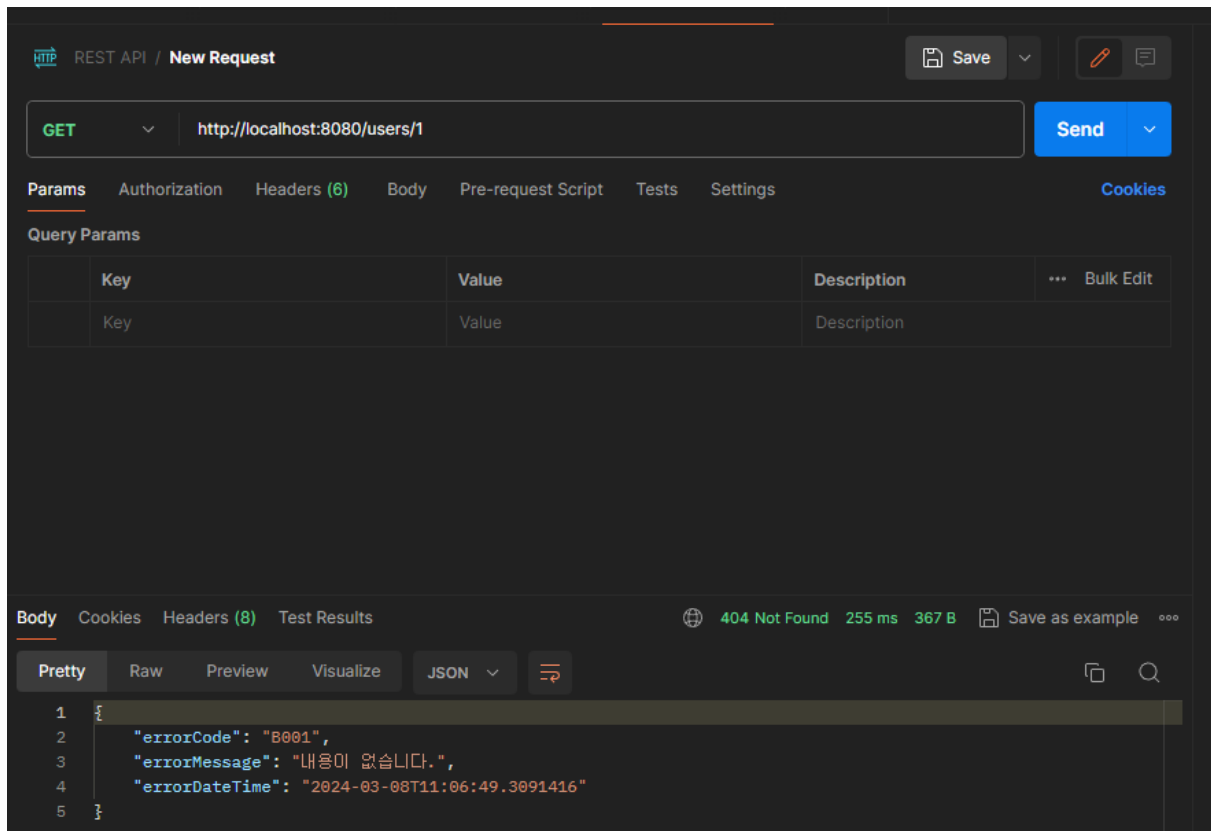
```

@ExceptionHandler(Exception.class)
public final ResponseEntity<ErrorResponse> handlerLogException(U
    ErrorResponse ErrorResponse = ErrorResponse.builder()
                                                .errorCode(e.get
                                                .errorMessage(e.
                                                .errorDateTime(L
                                                .build());

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(erro
}

```

- 실행 결과



✅ 에러창고에 있는 Code와 Message가 출력되는 점 확인할 수 있다.

- **UserException 활용**
 - 유저 목록을 조회할 때 유저가 한명도 없으면 **NOTFOUND** 예외를 던져준다.

```

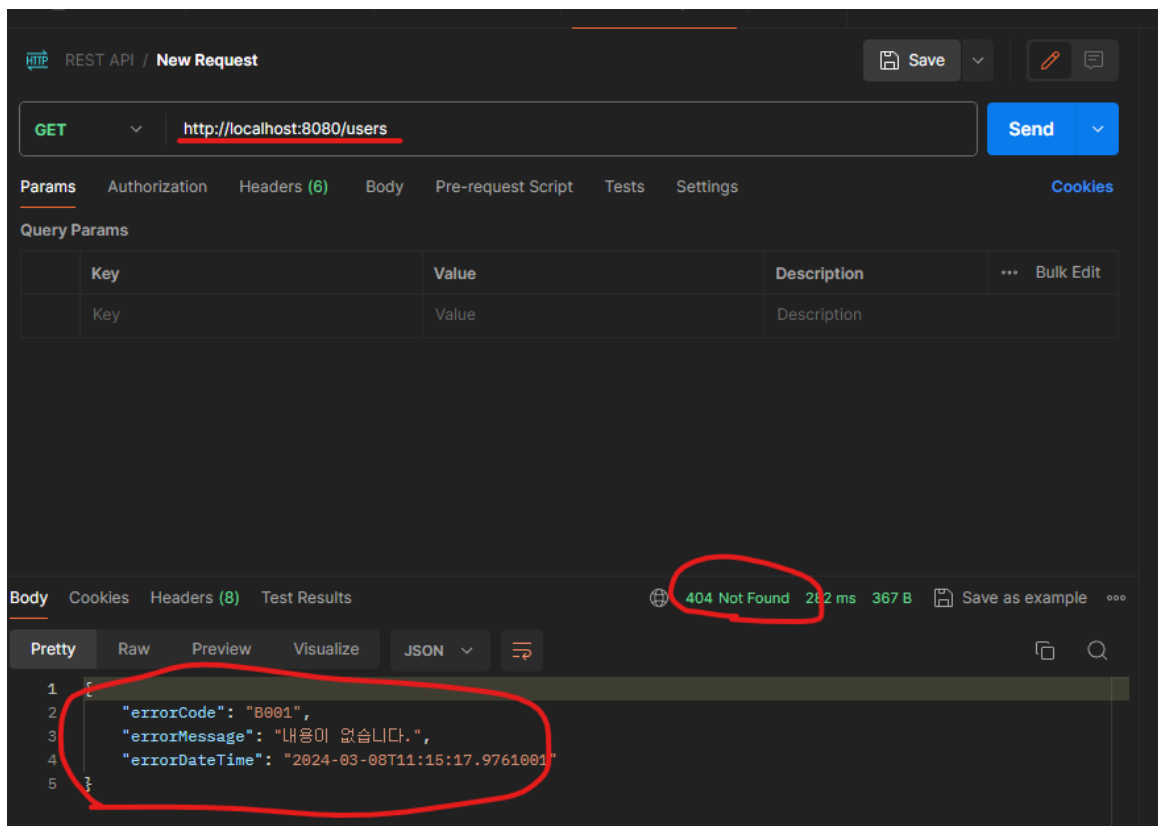
@GetMapping("users")
public ResponseEntity<List<User>> getAllUsers() {
    List<User> list = service.getAllUsers();

    if(list.size() == 0) {
        throw new UserException(ErrorCode.NOTFOUND);
    }

    return ResponseEntity.ok(list);
}

```

- 실행 결과



결론

- Service에서 에러가 발생하면 Handler로 이동한다고 이해함.

```

UserController -> UserService
(
에러 발생시에
Throw new LogException
Throw new UserException
->
GlobalExceptionHandler
LogException -> handleLogException (Exception e)
Exception -> handleException (Exception e)

```

Red circles and arrows highlight the flow from the exception types in the controller to the corresponding handlers in the GlobalExceptionHandler.

PutMapping(users)



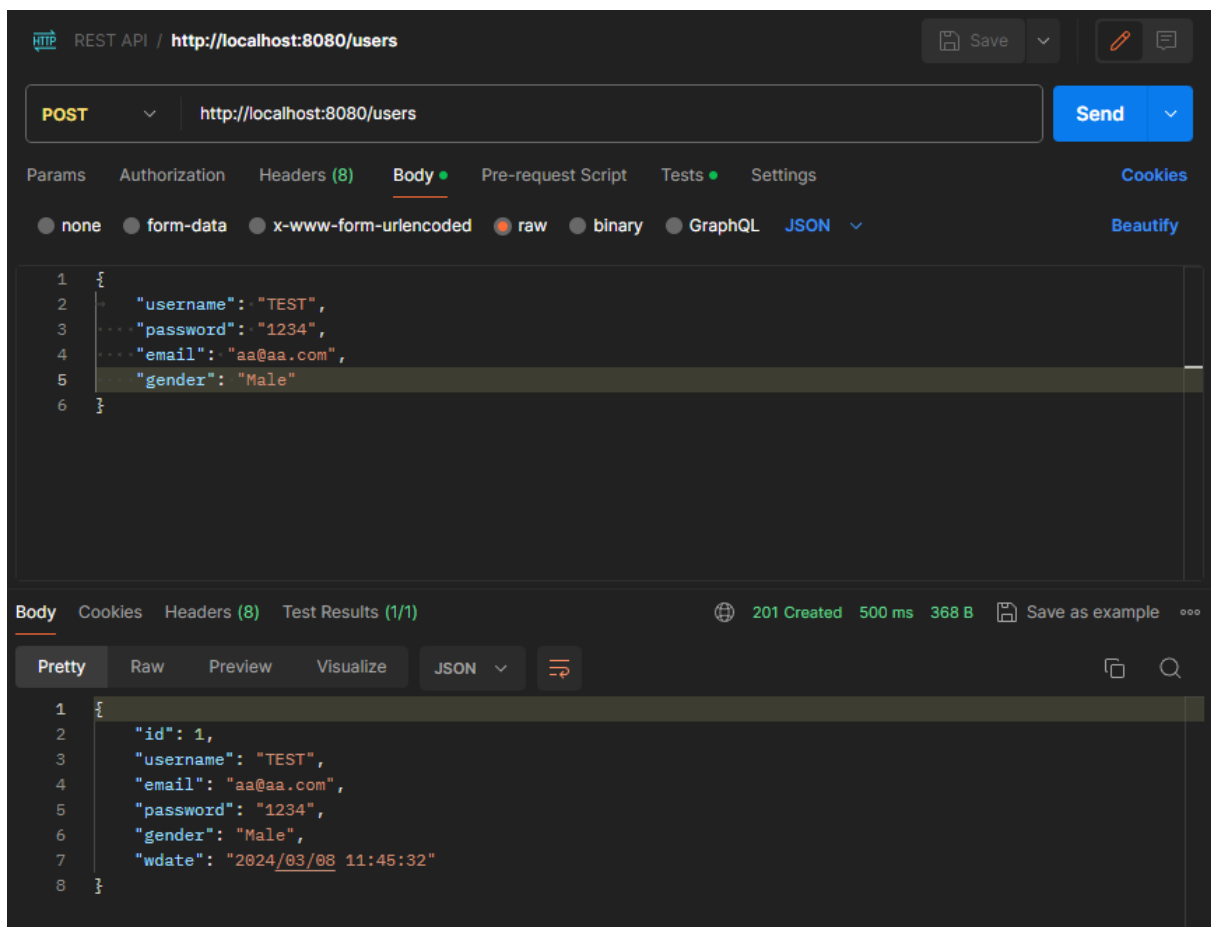
REST API 중 PUT은 수정을 뜻한다.

기초작업

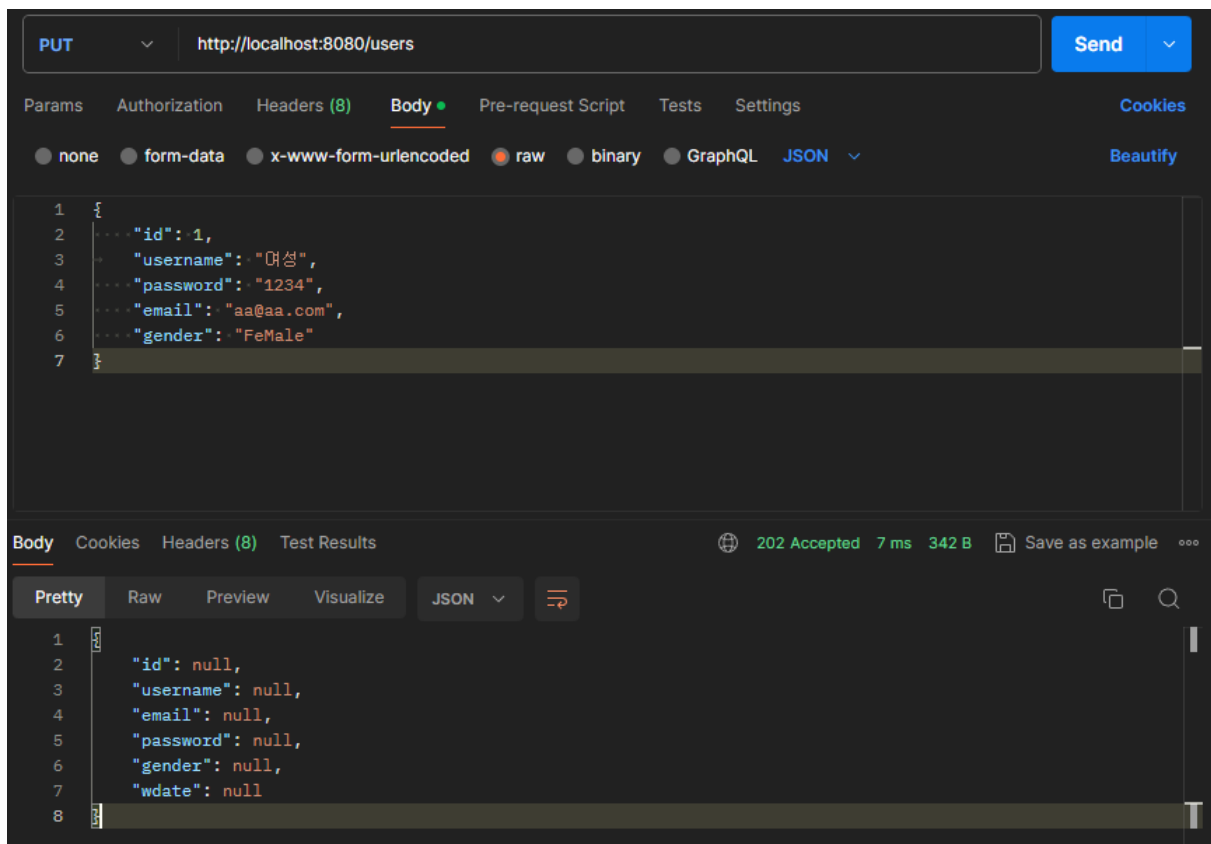
- C → `modifyUser` 생성
 - UserDTO를 통해 값을 요청받아서 console을 통해 값을 제대로 가져오는지 확인해보기

```
@PutMapping("users")
public ResponseEntity<User> modifyUser(@RequestBody UserDTO user) {
    System.out.println(userDTO.toString());
    return ResponseEntity.status(HttpStatus.ACCEPTED).body(new U
}
```

- PUT 기능을 확인하기 위해 먼저 POST로 user INSERT를 먼저 진행해준다.



- 실행 결과



✅ body로 빈 값의 객체값을 보내서 모든 내용이 null인 점 확인할 수 있다.



✅ 입력한 값이 UserDTO에 정상적으로 들어가는 점 확인할 수 있다.

데이터 유효성 검사 추가

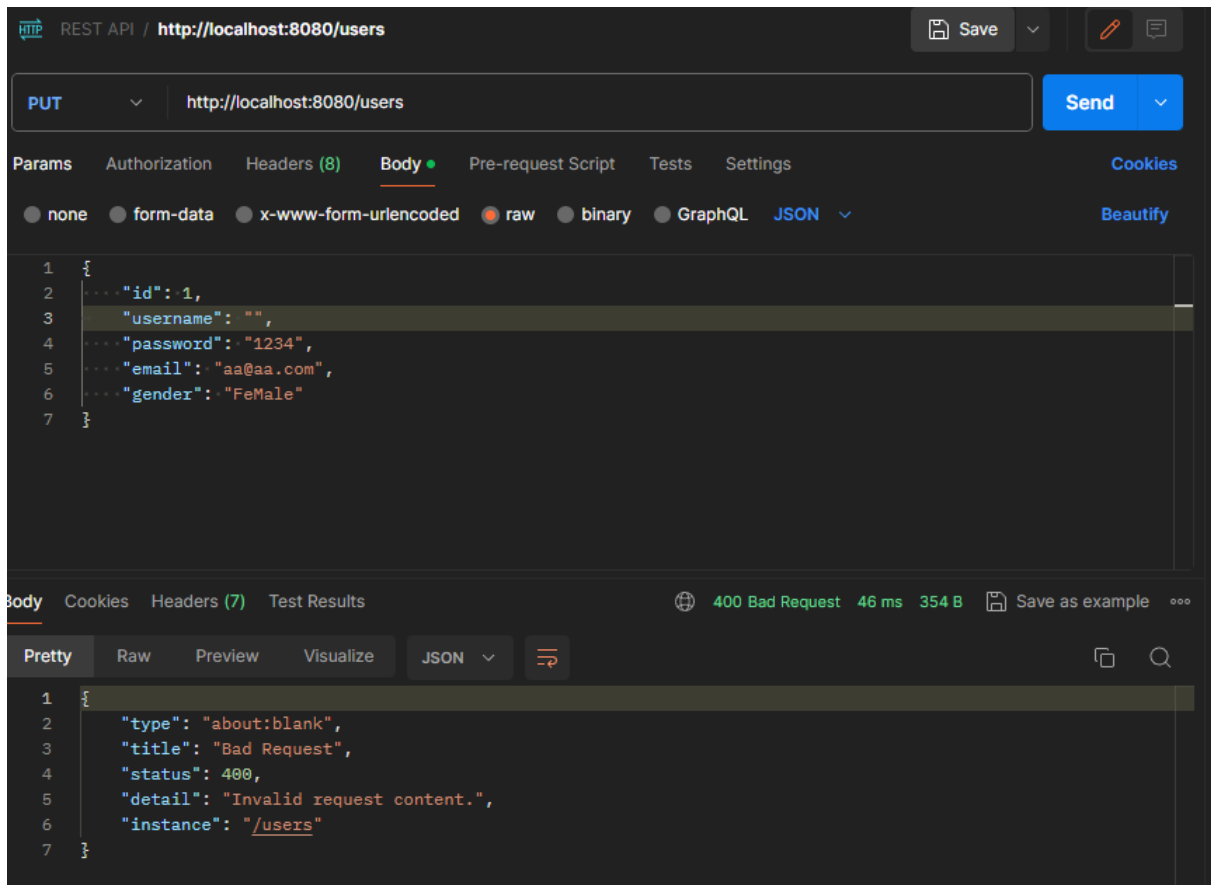
- C → `modifyUser` 수정
 - `@Valid` 어노테이션 추가

```

@PutMapping("users")
public ResponseEntity<User> modifyUser(@RequestBody @Valid UserDTO userDTO) {
    System.out.println(userDTO.toString());
    return ResponseEntity.status(HttpStatus.ACCEPTED).body(new UserDTO());
}

```

- 실행 결과



✅ 400 에러가 나온다. 이는 클라이언트가 서버로 잘못된 요청을 보냈다는 뜻이다.

```
[Field error in object 'userDTO' on field 'username': rejected value [];
```

✅ 400 에러가 뜬 이유는 데이터 유효성 검사 어노테이션을 통해 빈 값을 확인하여 오류가 발생하였기 때문이다.

요청한 값을 외부에서 확인할 수 있도록 처리

- C → `modifyUser` 수정
 - ModelMapper 사용하여 userDTO에 담긴 값을 User 객체로 전달.

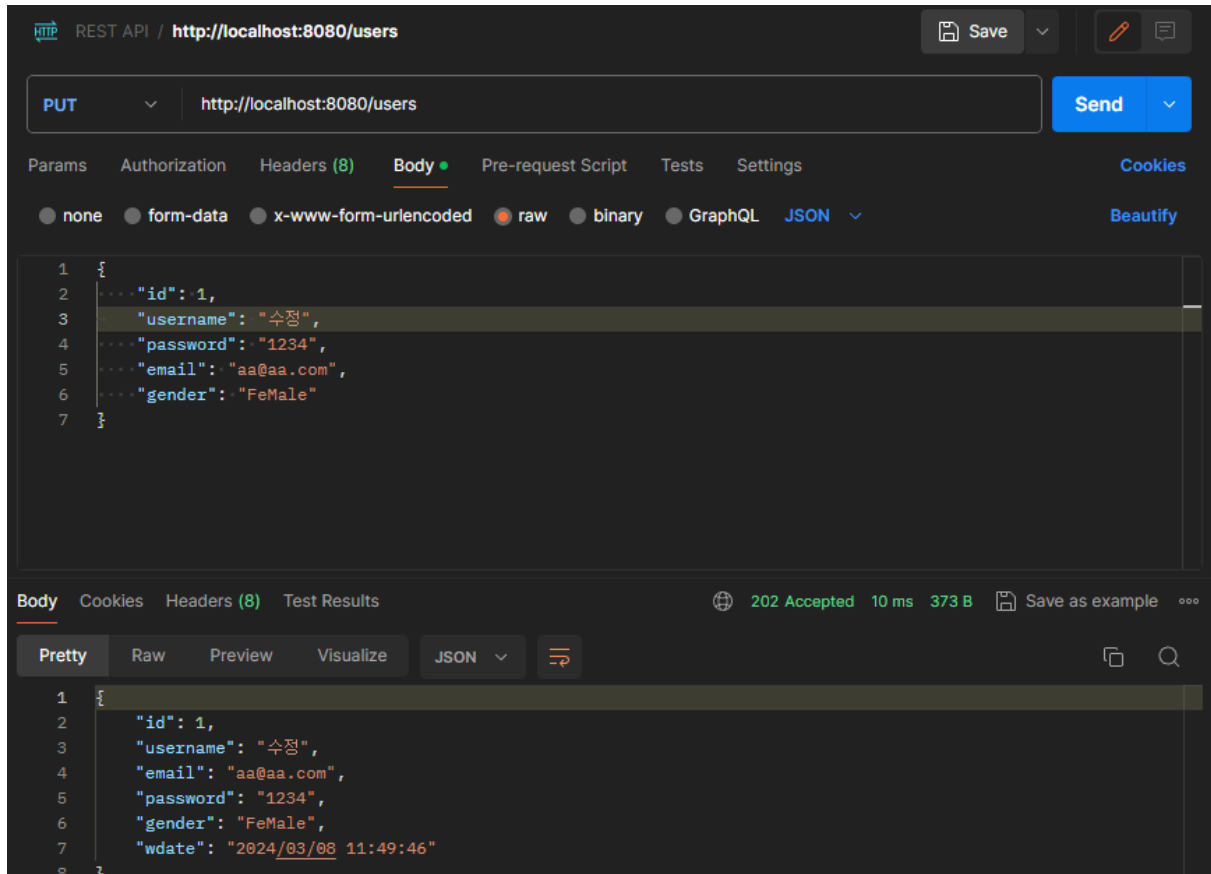
```
@PutMapping("users")
public ResponseEntity<User> modifyUser(@RequestBody @Valid UserD
    ModelMapper mapper = new ModelMapper();

    User user = mapper.map(userDTO, User.class);
    user.setWdate(LocalDateTime.now());

    System.out.println(user);
```

```
    return ResponseEntity.status(HttpStatus.ACCEPTED).body(user)
}
```

- 실행 결과



호스트: 192.168.0.49 | 데이터베이스: jea | 테이블: users | 데이터

jea.users: 1 행 (총) (exact) >> 다음 << 모두 보기 | 정렬

#	id	wdate	email	username	password	gender
1	1	2024-03-08 02:49:32.772567	aa@aa.com	TEST	1234	Male

DB 조작 기능(수정) 추가

기본작업

- C → `modifyUser` 수정

```
@PutMapping("users")
public ResponseEntity<User> modifyUser(@RequestBody @Valid UserDTO
    ModelMapper mapper = new ModelMapper();
```

```

    User user = mapper.map(userDTO, User.class);
    user.setWdate(LocalDateTime.now());

    User dbUser = service.modifyUser(user);
    return ResponseEntity.status(HttpStatus.ACCEPTED).body(dbUser);
}

```

- M → `modifyUser` 생성

```

public User modifyUser(User user) {
    User dbUser = userRepository.save(user);
    return dbUser;
}

```



잠깐! JPA 메소드 중 `save`는 INSERT와 UPDATE가 실행되는 메소드이다.
 우리는 `user`가 없으면 INSERT,
`user`가 존재하면 UPDATE가 진행될 수 있도록 로직을 짜주어야 한다.

예외처리를 통해 정상적인 로직으로 수정

- 참고에 하나 더 추가

```
NOTUPDATE(HttpStatus.NOT_FOUND, "D001", "수정할 이메일이 없습니다.")
```

- C → `modifyUser` 전체적으로 수정

```

public User modifyUser(User user) {
    // findByEmail을 통해 입력한 email에 대한 정보를 가져옴
    User emailUser = userRepository.findByEmail(user.getEmail());

    // 만약 입력한 email이 DB에 없으면
    if(emailUser == null) {
        // 예외처리를 진행.
        throw new UserException(ErrorCode.NOTUPDATE);
    }

    // email에 해당되는 정보를 수정
    emailUser.setWdate(user.getWdate());
    emailUser.setUsername(user.getUsername());
    emailUser.setPassword(user.getPassword());
}

```

```

    User dbUser = userRepository.save(emailUser);

    return dbUser;
}

```

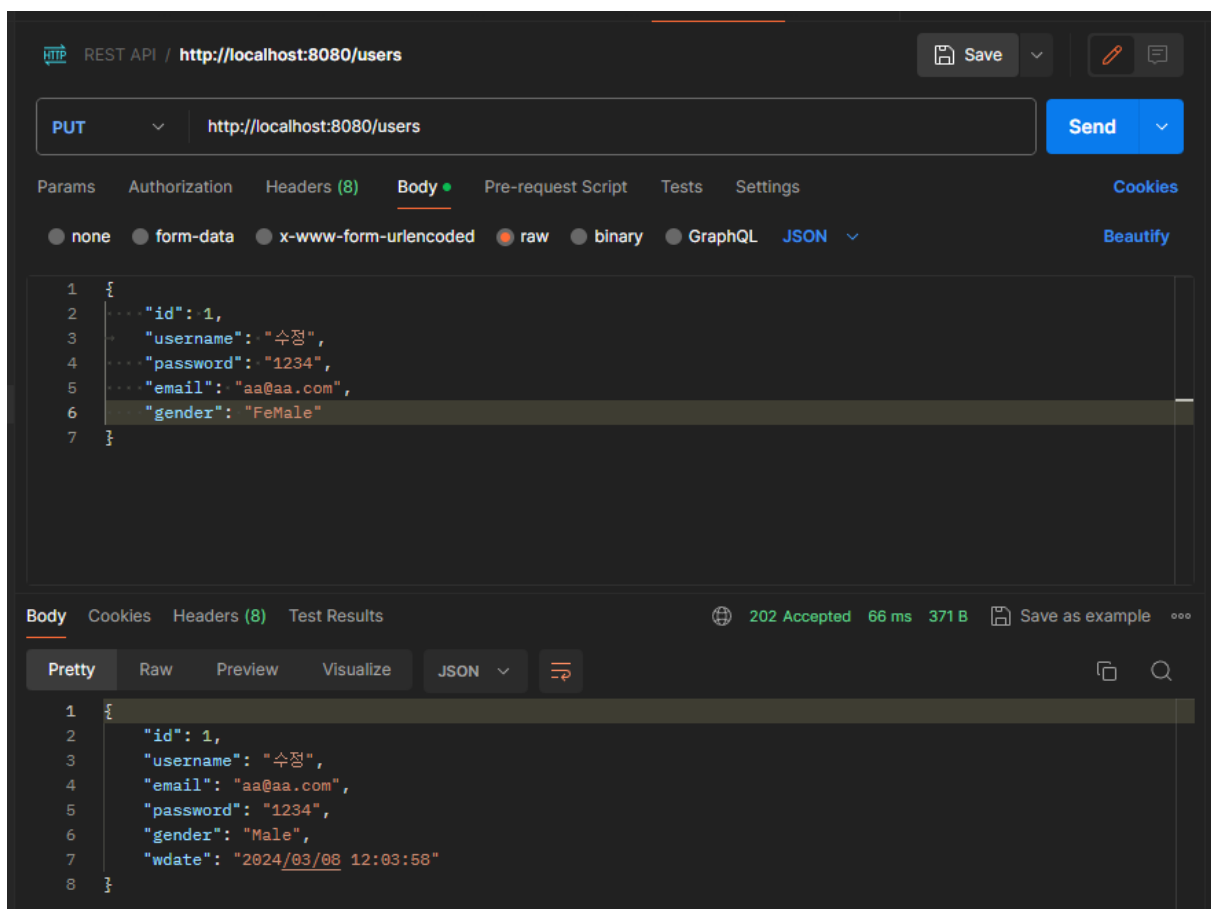
- 실행결과

```

Hibernate:
    update
        users
    set
        email=?,
        gender=?,
        password=?,
        username=?,
        wdate=?
    where
        id=?

```

- 정상 실행



호스트: 192.168.0.49 데이터베이스: jea 테이블: users 데이터 쿼리*

jea.users: 1 행 (총) (exact) >> 다음 << 모두 보기 | 정렬

#	id	wdate	email	username	password	gender
1	1	2024-03-08 03:03:58.496210	aa@aa.com	수정	1234	Male

- 비정상 실행
 - 입력한 email이 DB에 없어서 오류가 발생함.

REST API / http://localhost:8080/users Save

PUT http://localhost:8080/users Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautif

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON

```

1 {
2   "id": 1,
3   "username": "수정",
4   "password": "1234",
5   "email": "abb@aa.com",
6   "gender": "Female"
7 }

```

body Cookies Headers (8) Test Results 404 Not Found 14 ms 380 B Save as example

Pretty Raw Preview Visualize JSON

```

1 {
2   "errorCode": "D001",
3   "errorMessage": "수정할 이메일이 없습니다.",
4   "errorDateTime": "2024-03-08T12:05:05.0810123"
5 }

```

sql 파일을 통해 DB에 값 미리 넣기

- data.sql
 - 무조건 세명이 들어갈 수 있도록 진행

```

insert into users (email, gender, password, username, wdate, id) va
insert into users (email, gender, password, username, wdate, id) va
insert into users (email, gender, password, username, wdate, id) va

```

- 실행 결과

jea.users: 3 행 (총) (exact) >> 다음 <> 모두 보기 | ▼ 정렬 ▼ 열 (6/6)

#	id	wdate	email	username	password	gender
1	1	2024-03-08 03:08:30.000000	aaa@naver.com	홍길동	12341234	Male
2	2	2024-03-08 03:08:30.000000	bbb@naver.com	김길동	12341234	Male
3	3	2024-03-08 03:08:30.000000	ccc@naver.com	이길동	12341234	Male

PutMapping 진행

- 실행 결과

REST API / http://localhost:8080/users

PUT http://localhost:8080/users Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1 {
2   "username": "수정",
3   "password": "1234",
4   "email": "aaa@naver.com",
5   "gender": "FeMale"
6 }

```

Body Cookies Headers (8) Test Results 202 Accepted 462 ms 377 B Save as example

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 1,
3   "username": "수정",
4   "email": "aaa@naver.com",
5   "password": "1234",
6   "gender": "FeMale",
7   "wdate": "2024/03/08 12:09:12"
8 }

```

jea.users: 3 행 (총) (exact) >> 다음 <> 모두 보기 | ▼ 정렬 ▼ 열 (6/6)

#	id	wdate	email	username	password	gender
1	1	2024-03-08 03:09:12.622224	aaa@naver.com	수정	1234	FeMale
2	2	2024-03-08 03:08:30.000000	bbb@naver.com	김길동	12341234	Male
3	3	2024-03-08 03:08:30.000000	ccc@naver.com	이길동	12341234	Male