KB

It's Your Life

with

HTTP

알아보기

그란데 말입니다

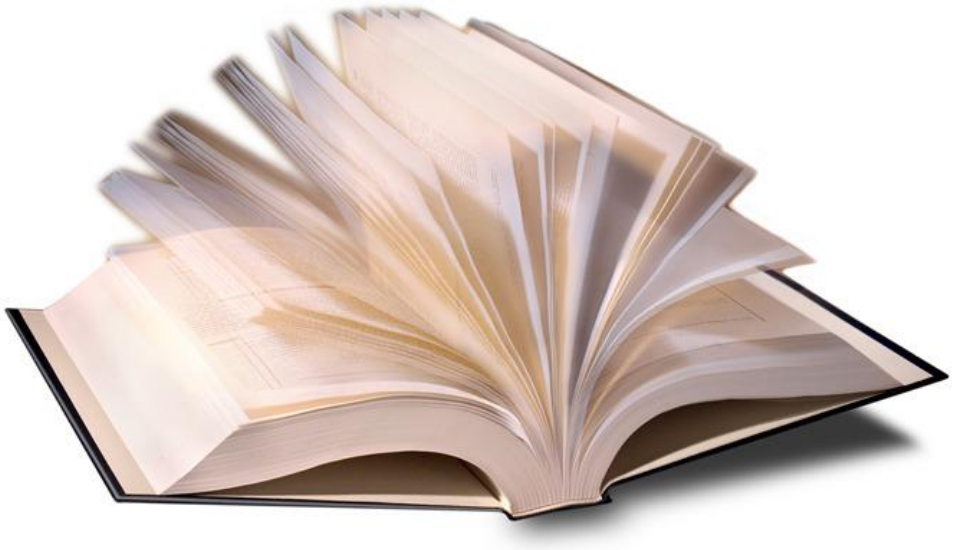
모든 웹 통신에 사용되는
HTTP 가 무엇을 의미 하는지 아시는 분!?

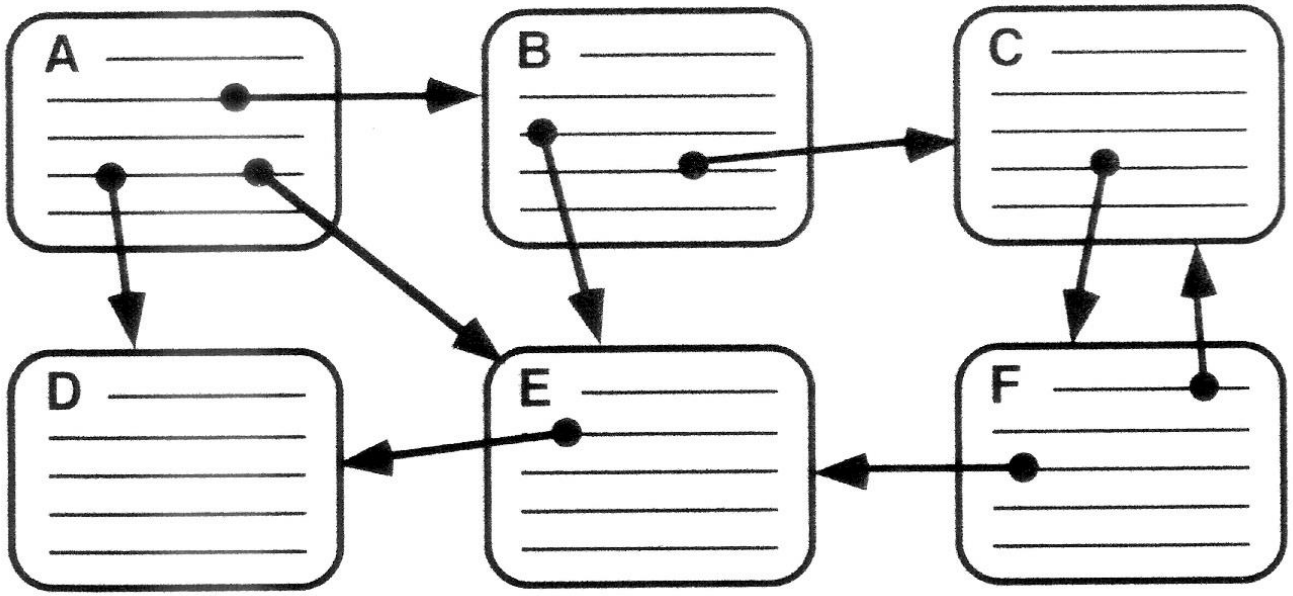
HTTP

HyperText Transfer Protocol

HyperText

- HyperText?





HTTP

Request 구조

Start Line --------- 요청라인 또는 상태라인

Header --------- 헤더는 생략 가능

Blank Line --------- 헤더의 끝을 빈 줄로 식별

Body --------- 바디는 생략 가능


Start Line
요청라인 또는 상태라인
Header
헤더는 생략 가능
Blank Line
헤더의 끝을 빈 줄로 식별
Body
바디는 생략 가능
GET /test.html HTTP/1.1
Host: google.com
Accept: text/html
Accept-Encoding: gzip, deflate
Connection: keep-alive
hl=ko&ogbl=0&page=99
start line
headers
blank line
body

일단 메서드를 명시 합니다!

```
GET /test.html HTTP/1.1
```

start line

마지막으로 HTTP 의 버전 표기

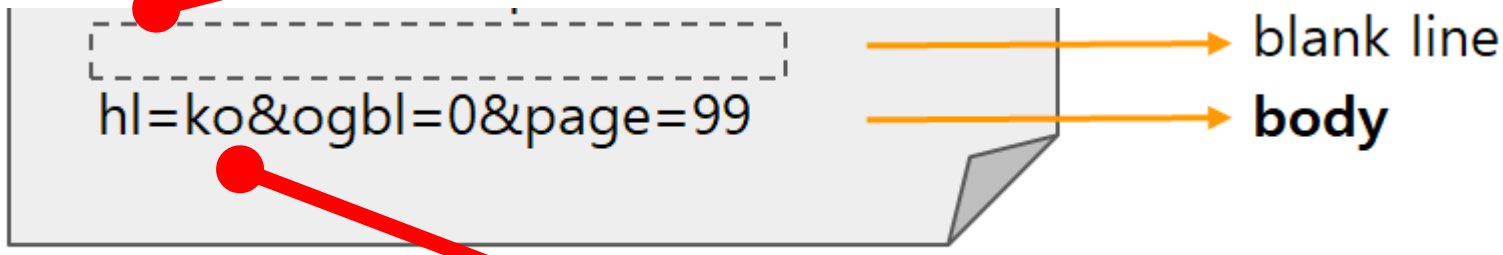
어디다 요청을 해야하는지도 알아야겠죠?
요청 주소가 다음에 따라 옵니다!

요청을 해석하기 위한 정보를 포함 합니다!

Host: google.com
Accept: text/html
Accept-Encoding: gzip, deflate
Connection: keep-alive

headers

Header 와 Body 를 구분하기 위한 공백



데이터 전달을 위한 body

브라우저에 의해 자동으로 생성되는 header 와 달리
사용자가 원하는 데이터 전달을 위해 사용 됩니다!

그란데 말입니다

Request 의 body 는 어떤 메서드에서만
사용이 가능할까요!?

보통 POST, PUT 에서 사용 됩니다!

GET 방식은
쿼리스트링 형태로 보내기 때문에
Body 를 사용하지 않습니다!

HTTP

Response 구조

당신도 똑같어!!

Start Line --------- 요청라인 또는 상태라인

Header --------- 헤더는 생략 가능

Blank Line --------- 헤더의 끝을 빈 줄로 식별

Body --------- 바디는 생략 가능

Start Line --------- 요청라인 또는 상태라인

Header --------- 헤더는 생략 가능

Blank Line --------- 헤더의 끝을 빈 줄로 식별

Body --------- 바디는 생략 가능

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Status Line

Response Headers

Response Message Header

A blank line separates header & body

Response Message Body

응답은 당연히 보낸 클라이언트로 보내면 되므로
주소가 없습니다!

```
HTTP/1.1 200 OK
```

→ Status Line

클라이언트 입장에서는 빠르게 판단을 내리고
거기에 맞는 화면을 그려야 하므로
응답 코드를 제일 위에 배치 합니다

그리고 바로 HTTP 버전을 표기

응답을 해석하기 위한 정보를 포함 합니다!

```
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html
```

Response Headers

Response Message Header

Header 와 Body 를 구분하기 위한 공백

```
<h1>My Home page</h1>
```

A blank line separates header & body

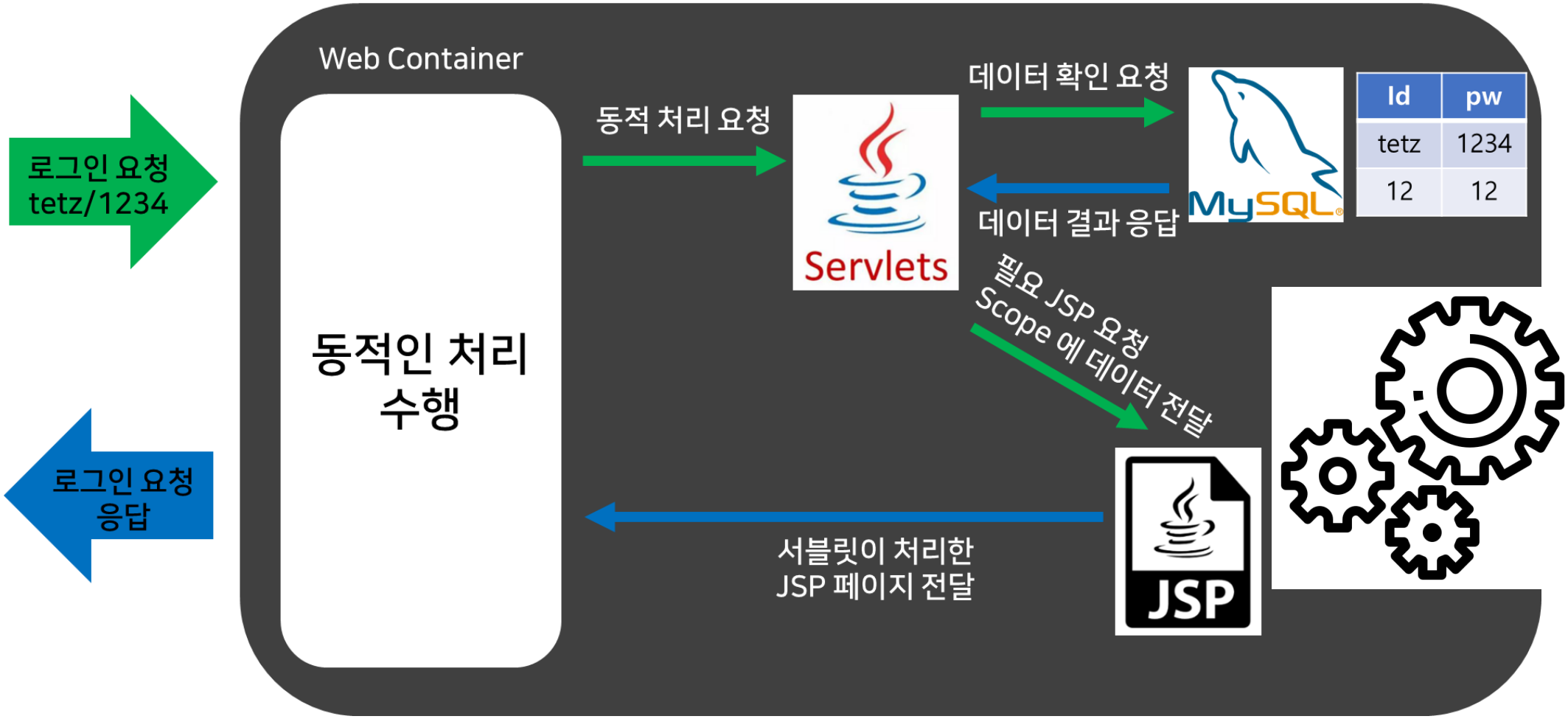
Response Message Body

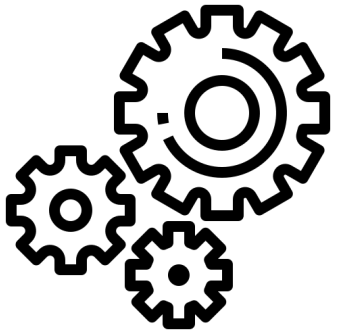
요청에 의한 응답을 Body 에 담아 보냅니다!

응답은 보통 HTML 혹은 JSON 이 포함되어
클라이언트의 요청 응답 결과를 전달 합니다!

REST API 와 HTTP 상태 코드

Tomcat Server





Client

React

Vue.js

ANGULAR

다양한 요청

요청에 대한 응답

Tomcat Server

spring

Servlets

MySQL

응답의 전부를 읽고 처리하면 늦습니다!

응답의 첫 줄만 읽고 어떤 상태인지
파악할 수 있어야 합니다!

HTTP Status Codes

Level 200

200: OK
201: Created
202: Accepted
203: Non-Authoritative Information
204: No content

Level 400

400: Bad Request
401: Unauthorized
403: Forbidden
404: Not Found
409: Conflict

Level 500

500: Internal Server Error
501: Not Implemented
502: Bad Gateway
503: Service Unavailable
504: Gateway Timeout
599: Network Timeout

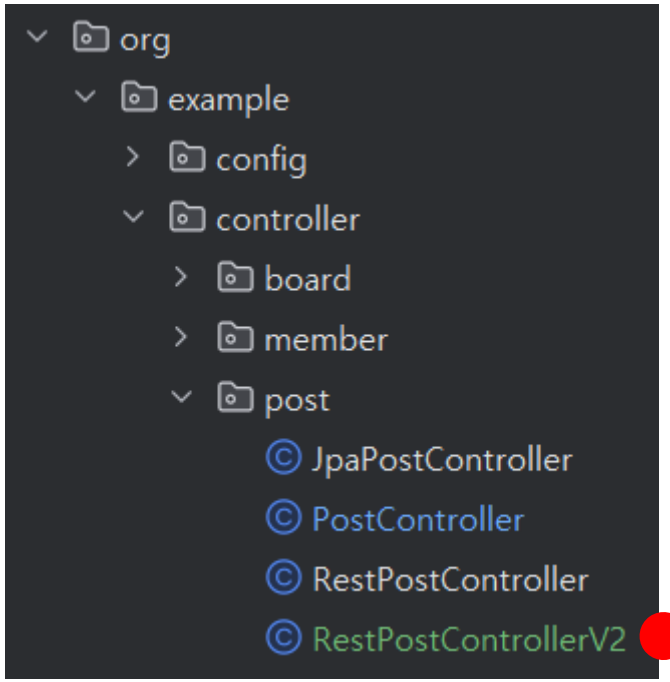
ResponseEntity 로 응답 만들어 보내기!

```java
// 게시글 목록
@GetMapping("/show")  Tetz
public List<PostDto> postList(HttpServletRequest request, Model model) {
    log.info("================> 게시글 목록 페이지 호출, " + request.getRequestURI());

    return postRepository.findAll();
}
```

기존의 RestController 는
응답의 Body 에 들어갈 데이터만 전달하는 구조를
가지고 있었습니다!

→ 즉, 우리가 원하는 대로 응답을 만들 수 없었습니다!

ResponseEntity



ResponseEntity 적용을 위해
RestPostControllerV2 를 만들어 봅시다

```java
@RestController  new *
@Slf4j
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:5173")
@RequestMapping("/post/v2/rest")
public class RestPostControllerV2 {
```

일단 요청 주소를 v2 로 변경!

이제 REST API 에 대한 응답은
ResponseEntity 로 만들어 보내므로
리턴의 타입은 ResponseEntity 가 됩니다!

그리고 Body 에 담아 전달한 데이터의 타입을
제네릭을 활용하여 전달 합니다!

```java
// 게시글 목록
@GetMapping("/show")
public ResponseEntity<List<PostDto>> postList(HttpServletRequest request, Model model) {
    log.info("=================> 게시글 목록 페이지 호출, " + request.getRequestURI());

    List<PostDto> allList = postRepository.findAll();

    return ResponseEntity.ok(allList);
}
```

```java
// 게시글 목록
@GetMapping("/show")  new *
public ResponseEntity<List<PostDto>> postList(HttpServletRequest request, Model model) {
    log.info("================> 게시글 목록 페이지 호출, " + request.getRequestURI());

    List<PostDto> allList = postRepository.findAll();

    return ResponseEntity.ok(allList);
}
```

요청에 대한 응답이 성공적이므로
ResponseEntity 에서 제공하는 빌더 패턴을 사용하여
ok 메서드를 이용하여 응답 코드를 200(성공)으로 만들고
메서드의 인자로 전달 하고자하는 데이터를 넣어 줍니다!

해당 데이터는 자동으로 JSON 형태로 변환되어
응답의 Body 에 담겨서 전달 됩니다!

PostMan으로 확인

http://localhost:8080/post/v2/rest/show

변경된 주소로 요청을 보내 봅니다!

Body Cookies (1) Headers (8) Test Results

Status: 200 OK

Pretty Raw Preview Visualize JSON

ok() 에 의해 생성 된 200번 응답 코드 확인!


```
[
    {
        "id": 1,
        "title": "첫 글 테스트",
        "content": "첫 글 테스트 입니다!"
    },
    {
        "id": 2,
        "title": "두번째 테스트",
        "content": "두번째 테스트 입니다!"
    },
    {
        "id": 3,
        "title": "12",
        "content": "12"
    },
```


JSON 으로 변환 되어 전달 된 데이터 확인!

ResponseEntity 로 다양한 응답 만들기!

테스트를 위해 /test 주소에 매핑

Body 에 담겨서 전달 되는
텍스트의 인코딩을 설정!

```java
// ResponseEntity 테스트
@GetMapping(value = "/test", produces = "text/plain;charset=UTF-8")  new *
public ResponseEntity<String> test() {
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("요청을 처리할 수 없습니다");
}
```

```java
// ResponseEntity 테스트
@GetMapping(value = "/test", produces = "text/plain;charset=UTF-8")  new *
public ResponseEntity<String> test() {
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body("요청을 처리할 수 없습니다");
}
```

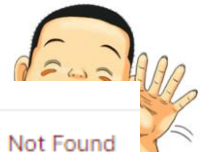
상태 코드를 status 메서드로 설정
HttpStatus 클래스에 정의된 필드를 사용하여
다양한 상태 코드를 명시적으로 사용 가능!

전달하고 싶은 메시지를 body()
메서드를 이용하여 body 에 담아서 전달하기

http://localhost:9080/post/v2/rest/test

GET http://localhost:9080/post/v2/rest/test

방금 만든 컨트롤러에 Request 보내기!



Body Cookies (1) Headers (8) Test Results

Status: 404 Not Found

Pretty Raw Preview Visualize Text

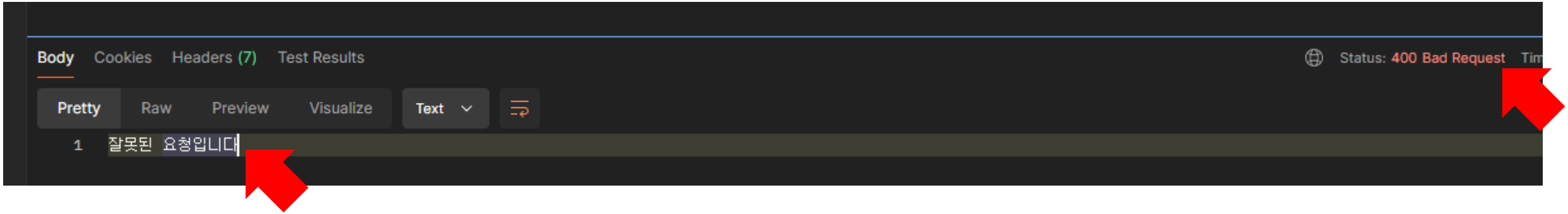
```
1  요청을 처리할 수 없습니다
```

설정한 상태 코드 확인!

Body 에 담아서 전달한 메시지 확인!

```java
// ResponseEntity 테스트2
@GetMapping(value = "/test2", produces = "text/plain;charset=UTF-8")  new *
public ResponseEntity<String> test2() {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("잘못된 요청입니다");
}
```

위와 같은 컨트롤러 메서드를 추가해서 결과를 확인!



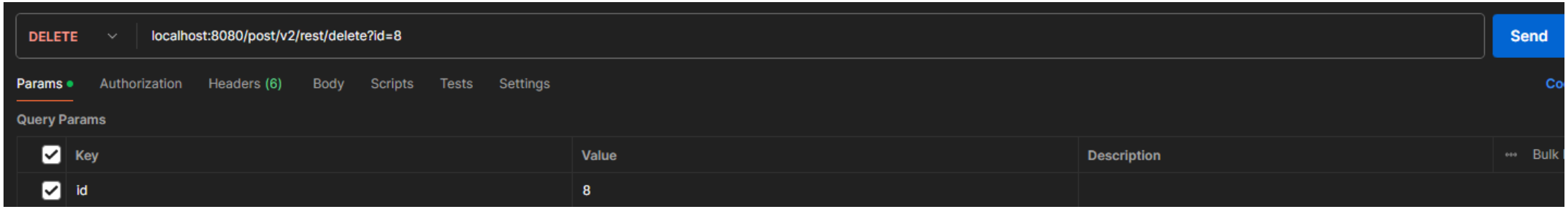
실습, /delete 요청 수정하기!

- 아래의 /delete 요청에 대한 응답을 기존의 MVC 패턴이 아닌 REST API 형태의 응답으로 수정해 주세요!

```
@DeleteMapping("/delete") Tetz
public String postDelete(@RequestParam("id") String id, HttpServletRequest request) {
    log.info("================> 게시글 삭제 기능 호출, " + request.getRequestURI());

    long postId = Long.parseLong(id);
    int affectedRows = postRepository.delete(postId);

    if (affectedRows > 0) log.info("삭제 성공");

    return "redirect:/post/v1/show";
}
```

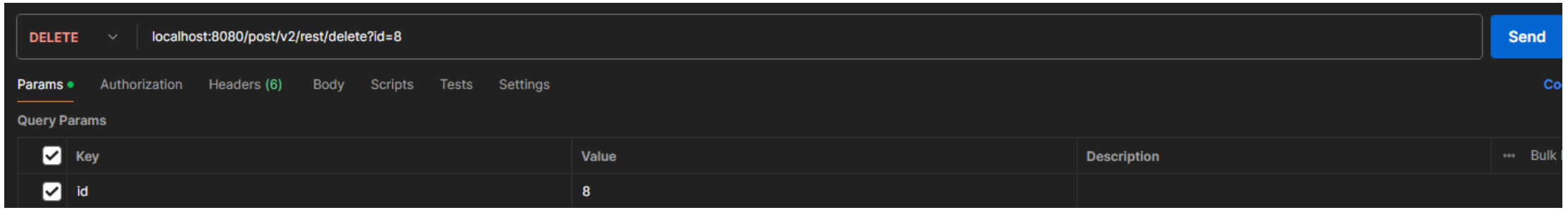
실습, /delete 요청 수정하기!

- 기존 메서드와 동일하게 삭제하려는 게시글의 id 를 쿼리스트링으로 받습니다
- 삭제가 성공하면 아래의 스크린 샷 처럼 200 Status 코드와 함께,
  "삭제 성공" 이라는 메시지를 띄웁니다



실습, /delete 요청 수정하기!

- 삭제가 실패하면 아래와 같이 404 에러를 띄우면서, “요청을 처리할 수 없습니다“ 메시지를 띄우는 컨트롤러를 작성 하시면 됩니다!



실습, /delete 요청 수정하기!

- 힌트, 응답의 경우 한글이 깨질 수 있으니 응답의 인코딩을 설정하는 아래의 코드를 잘 활용하세요

```
@DeleteMapping(value = "/delete", produces = "text/plain;charset=UTF-8")
```

실습

코드 채우기 파트

```java
@DeleteMapping(value = "/delete", produces = "text/plain;charset=UTF-8")  kdtTetz *
public ResponseEntity<String> postDelete(@RequestParam("id") Long id, HttpServletRequest request) {
    log.info("================> 게시글 삭제 기능 호출, " + request.getRequestURI());

    // 삭제 요청 및 삭제 요청에 대한 결과를 affectedRows 로 받기
    int affectedRows = postRepository.delete(id);

    if (affectedRows > 0) {
        // 여기를 채우세요
    } else {
        // 여기를 채우세요
    }
}
```