

11장 HTTP와 REST 컨트롤러

출처: 코딩 자율학습 스프링부트3 자바 백엔드 개발 입문, 홍팩, 길벗, 2023

REST API 동작 이해하기

- HTTP 요청 메시지의 구조

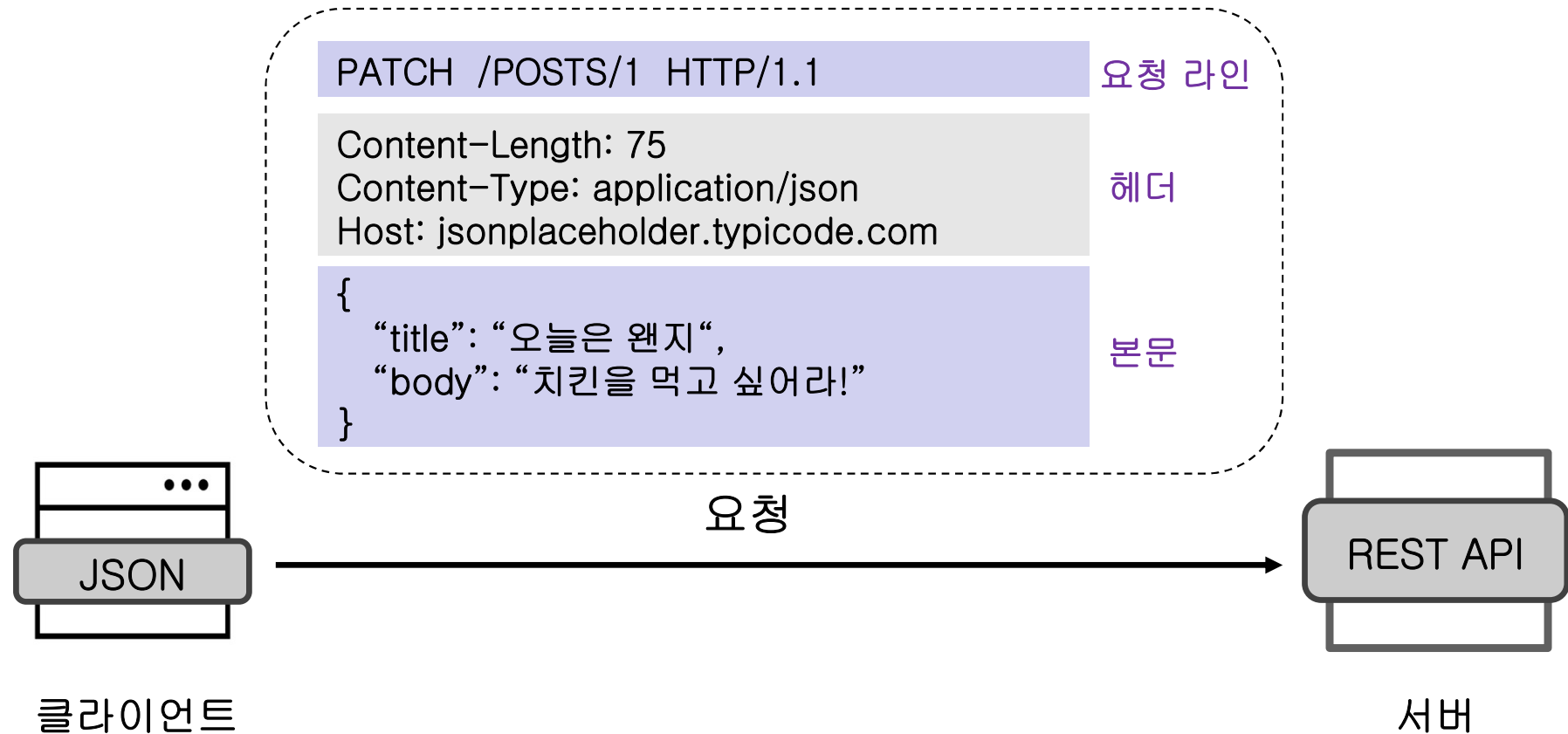


그림 11-1 요청 메시지 구조

REST API 동작 이해하기

- HTTP 응답 메시지의 구조

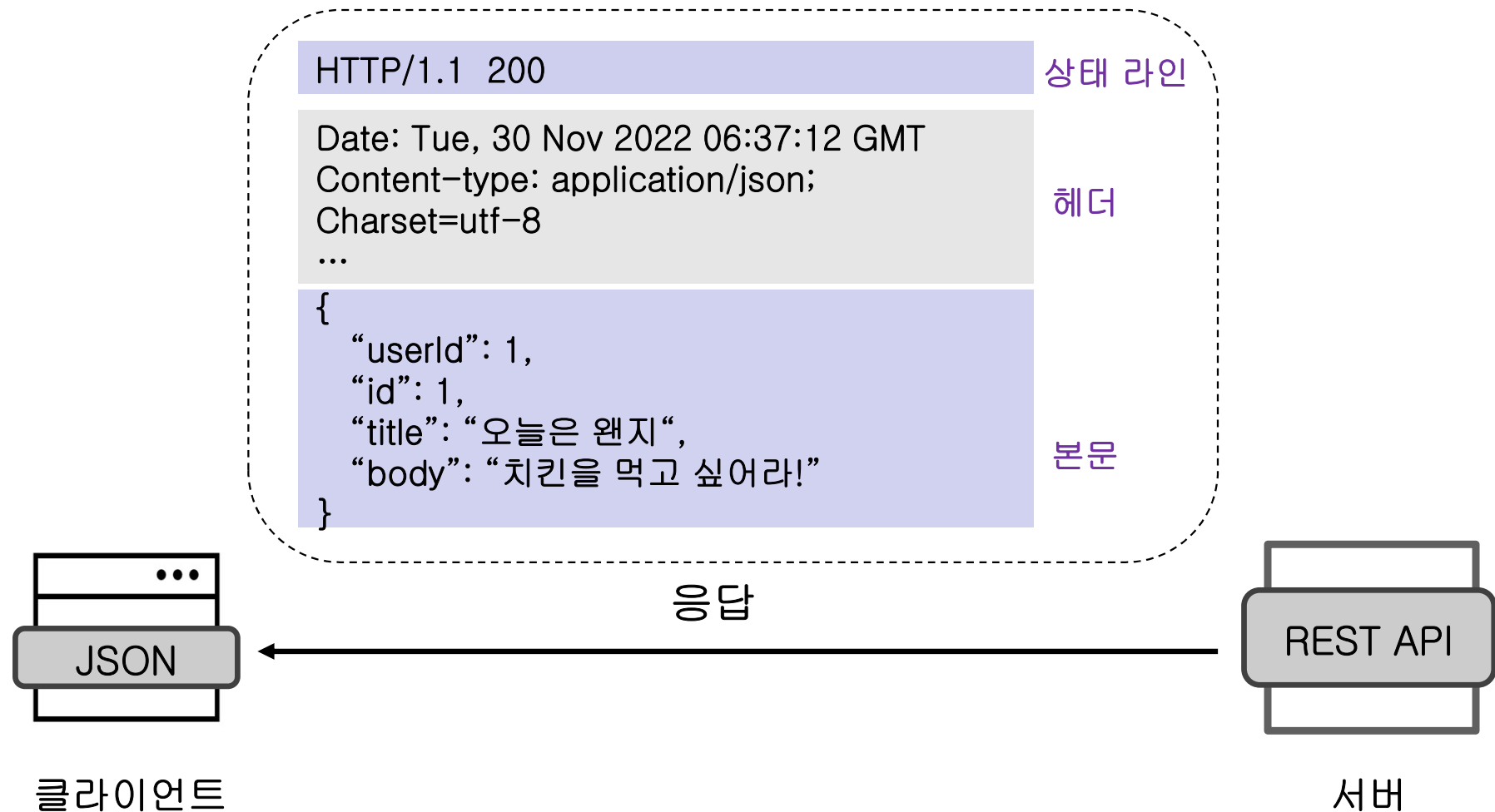


그림 11-2 응답 메시지 구조

- JSON 데이터 예시

```
{
  "id": 1,
  "name": "Park",
  address: {
    "street": "Nambu Street 151",
    "suite": "Central Villat 301",
    ...
  },
  "likes": ["singing", "jogging", "writing"]
}
```

REST

- **REST**

- HTTP URL로 서버의 자원(resource)을 명시하고,
- HTTP 메서드(POST, GET, PATCH/PUT, DELETE)로 해당 자원에 대해 CRUD(생성, 조회, 수정, 삭제) 수행

- **API**

- 클라이언트가 서버의 자원을 요청할 수 있도록 서버에서 제공하는 인터페이스

- **REST API 활용**

- 클라이언트가 기기에 구매받지 않고, 서버의 자원을 이용할 수 있음
- 서버가 클라이언트의 요청에 체계적으로 대응할 수 이어, 서버 프로그램의 재사용성과 확장성이 좋아짐

REST API의 구현 과정

● REST API의 주소 설계

- 조회요청(GET): /api/articles 또는 /api/articles/{id}
- 생성 요청(POST): /api/articles
- 수정요청(PATCH): /api/articles/{id}
- 삭제요청(DELETE): /api/articles/{id}

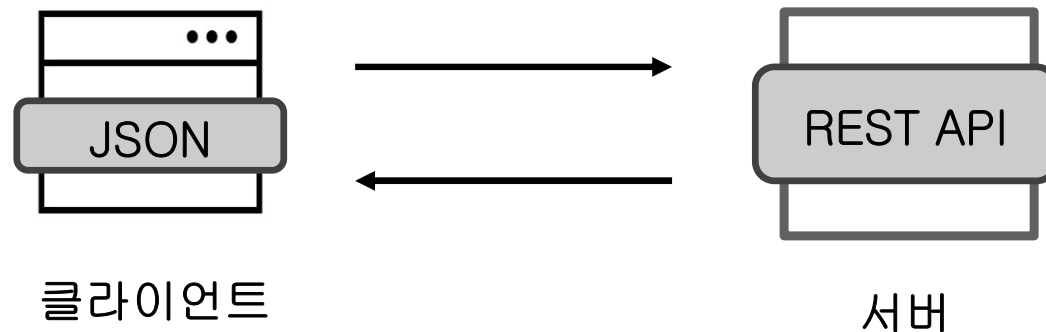


그림 11-4 REST API 주소 설계

REST API URL

GET	/articles
GET	/articles/id
POST	/articles
PATCH	/articles/id
DELETE	/articles/id

REST API의 구현 과정

● REST 컨트롤러

- REST API로 설계된 URL 요청을 받아 그 결과를 JSON으로 반환해 줄 컨트롤러
- REST API로 요청과 응답을 주고받을 때는 REST 컨트롤러를 사용
- 응답에 적절한 상태 코드를 반환하기 위해 ResponseEntity 사용



그림 11-5 RES 컨트롤러와 ResponseEntity의 역할

REST 컨트롤러 맛보기

- 패키지 생성: api
 - [src-main-java - com.example.firstproject.api]
- api 패키지에 FirstApiController 클래스 생성
 - @RestController
 - ✓ REST 컨트롤러

FirstApiController.java

```
package com.example.firstproject.api;

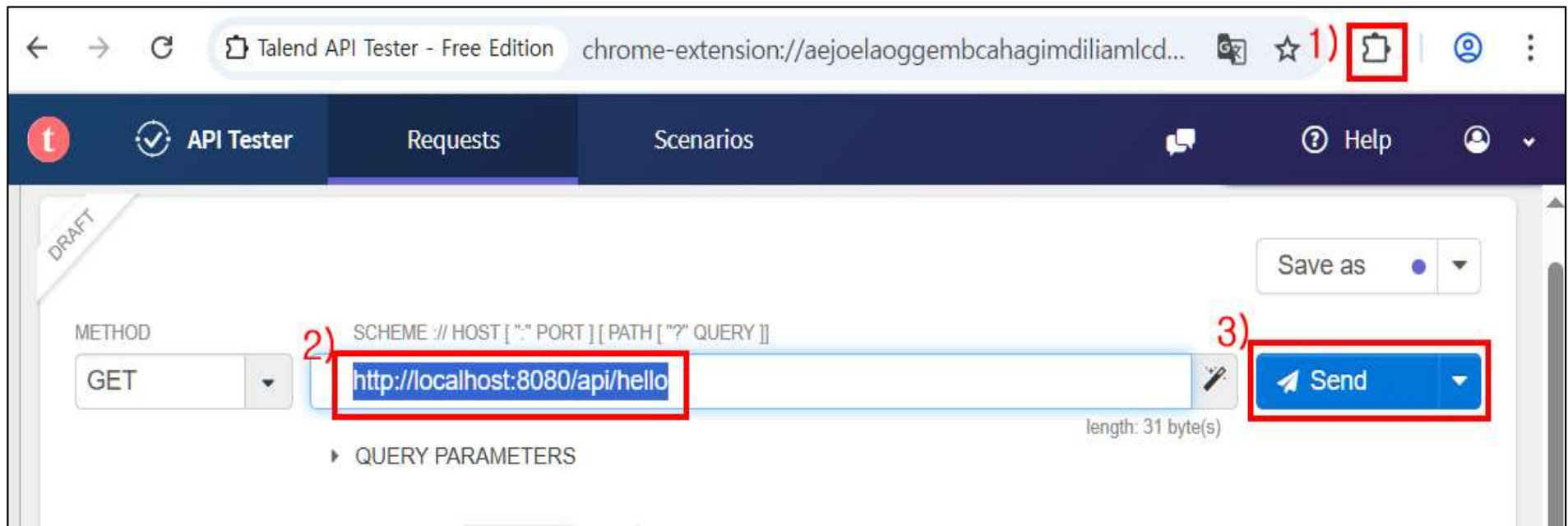
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class FirstApiController {

    @GetMapping("/api/hello")
    public String hello(){
        return "Hello World!";
    }
}
```

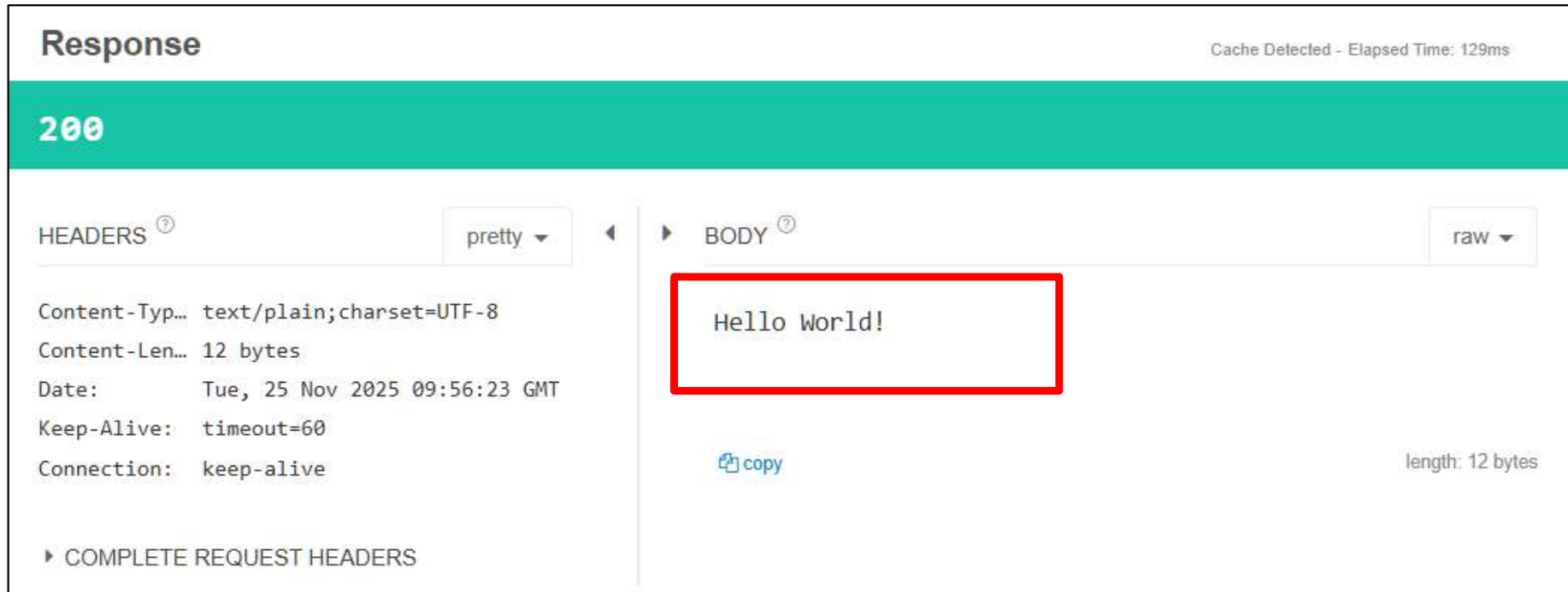

REST 컨트롤러 맛보기

- 서버 실행
- Talend API Tester
 - 메소드: GET
 - URL: **http://localhost:8080/api/hello**



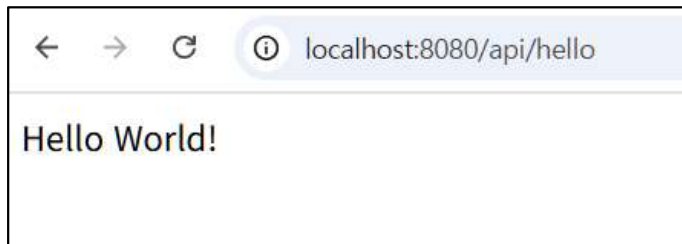
REST 컨트롤러 맛보기

- 응답 확인



The screenshot shows a REST client interface with a 'Response' tab selected. The status is 200, and the response time is 129ms. The 'HEADERS' section is expanded, showing 'Content-Type: text/plain; charset=UTF-8', 'Content-Length: 12 bytes', 'Date: Tue, 25 Nov 2025 09:56:23 GMT', 'Keep-Alive: timeout=60', and 'Connection: keep-alive'. The 'BODY' section is also expanded, showing 'Hello World!' in a red box. A 'copy' button is visible below the body text, and the length is noted as 'length: 12 bytes'.

- <http://localhost:8080/api/hello>



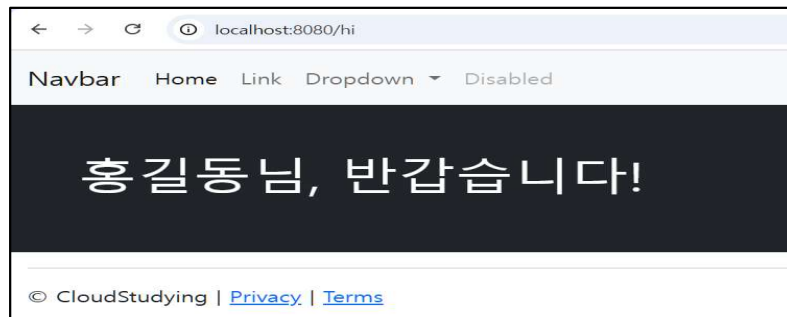
REST 컨트롤러와 일반 컨트롤러의 차이

- 2장에서 구현한 FirstController

```
@Controller
public class FirstController {

    @GetMapping("/hi")
    public String niceMeetYou(Model model){
        model.addAttribute(attributeName: "username", attributeValue: "홍길동");
        return "greetings";
    }
}
```

- localhost:8080/hi



REST 컨트롤러와 일반 컨트롤러의 차이

- Talend API Tester에서 확인

The screenshot displays the Talend API Tester interface. The top navigation bar includes 'API Tester', 'Requests', and 'Scenarios'. The main area is divided into 'REQUEST' and 'RESPONSE' sections.

Request Section:

- METHOD:** GET
- URL:** `http://localhost:8080/hi` (highlighted with a red box)
- Send Button:** A blue button with a paper plane icon and the text 'Send' (highlighted with a red box).
- HEADERS:** Includes '+ Add header' and 'Add authorization' buttons.
- BODY:** A message states 'XHR does not allow payloads for GET request.'

Response Section:

- Status:** 200 (highlighted with a green bar)
- HEADERS:** Lists response headers: Content-Type: text/html; charset=UTF-8, Content-Language: ko-KR, Content-Length: 2 kilobytes, Date: Tue, 25 Nov 2025 10:13:00 GMT, Keep-Alive: timeout=60, Connection: keep-alive.
- BODY:** Displays the HTML response (highlighted with a red box):

```
<!doctype html>
<html lang="ko">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <!-- Bootstrap CSS -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQ1
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

REST 컨트롤러와 일반 컨트롤러의 차이

- 일반 컨트롤러와 REST 컨트롤러 비교
 - 일반 컨트롤러는 뷰 페이지를 반환
 - REST 컨트롤러는 JSON이나 텍스트 데이터 반환

REST API: GET 구현하기 - 모든 게시물 조회

- com.example.firstproject.api에 ArticleApiController 클래스 생성

ArticleApiController.java

```
package com.example.firstproject.api;

import org.springframework.web.bind.annotation.RestController;

@RestController
public class ArticleApiController {
    // GET
    // POST
    // PATCH
    // DELETE
}
```

REST API: GET 구현하기 - 모든 게시글 조회

- GET 요청을 처리하는 index() 메서드

api/ArticleController.java

```
@RestController
public class ArticleApiController {

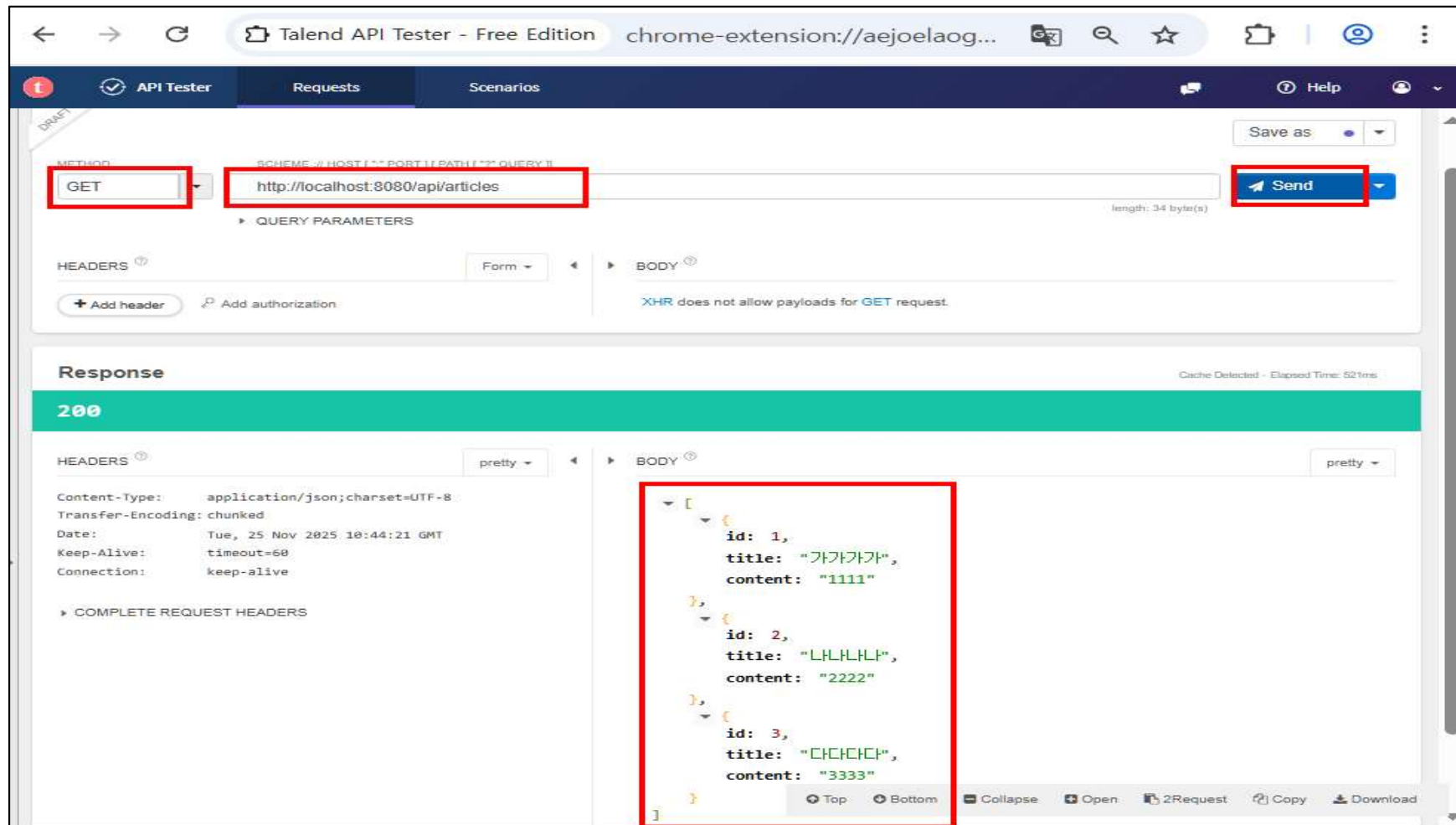
    @Autowired
    private ArticleRepository articleRepository;

    // GET
    @GetMapping("/api/articles")
    public List<Article> index() {
        return articleRepository.findAll();
    }

    // POST
    // PATCH
    // DELETE
}
```

REST API: GET 구현하기 - 모든 게시글 조회

- 서버 재구동
- Talend API Tester
 - 메서드 : GET
 - URL: <http://localhost:8080/api/articles>



REST API: GET 구현하기 - 모든 게시글 조회

- 응답 메시지의 JSON 데이터 확인



The screenshot shows a REST client interface with four tabs: HISTORY, ASSERTIONS, HTTP, and DESCRIPTION. The HTTP tab is selected, displaying the details of a GET request to /api/articles. The response is an HTTP 200 OK with a Content-Type of application/json; charset=UTF-8. The response body is a JSON array of three objects, each representing an article. The JSON data is highlighted with a red rectangular box.

```
GET /api/articles HTTP/1.1
Host: localhost:8080

HTTP/1.1 200
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 25 Nov 2025 10:44:21 GMT
Keep-Alive: timeout=60
Connection: keep-alive

[{"id":1,"title":"가가가가","content":"1111"}, {"id":2,"title":"나나나나","content":"2222"}, {"id":3,"title":"다다다다","content":"3333"}]
```

REST API: GET 구현하기 - 단일 게시물 조회

- GET 요청을 처리하는 show() 메서드

```
@RestController
public class ArticleApiController {
    @Autowired
    private ArticleRepository articleRepository;

    (중략)

    @GetMapping("/api/articles/{id}")
    public Article show(@PathVariable Long id){
        return articleRepository.findById(id).orElse( other: null);
    }

    // POST
    // PATCH
    // DELETE
}
```

REST API: GET 구현하기 - 단일 게시글 조회

- 서버 재구동
- Talend API Tester
 - 메서드 : GET
 - URL: <http://localhost:8080/api/articles/1>

The screenshot shows the Talend API Tester interface. The 'Requests' tab is active. The 'METHOD' dropdown is set to 'GET', and the 'URL' field contains 'http://localhost:8080/api/articles/1'. The 'Send' button is highlighted. Below the request fields, the 'Response' section shows a '200' status code. The 'BODY' tab is selected, displaying a JSON response:

```
{  "id": 1,  "title": "가가가가",  "content": "1111"}
```

. The response is formatted as 'pretty' JSON. The 'Headers' section shows various response headers like 'Content-Type: application/json; charset=UTF-8' and 'Date: Tue, 25 Nov 2025 10:55:30 GMT'.

REST API: POST 구현하기

- POST 요청을 처리하는 create() 만들기

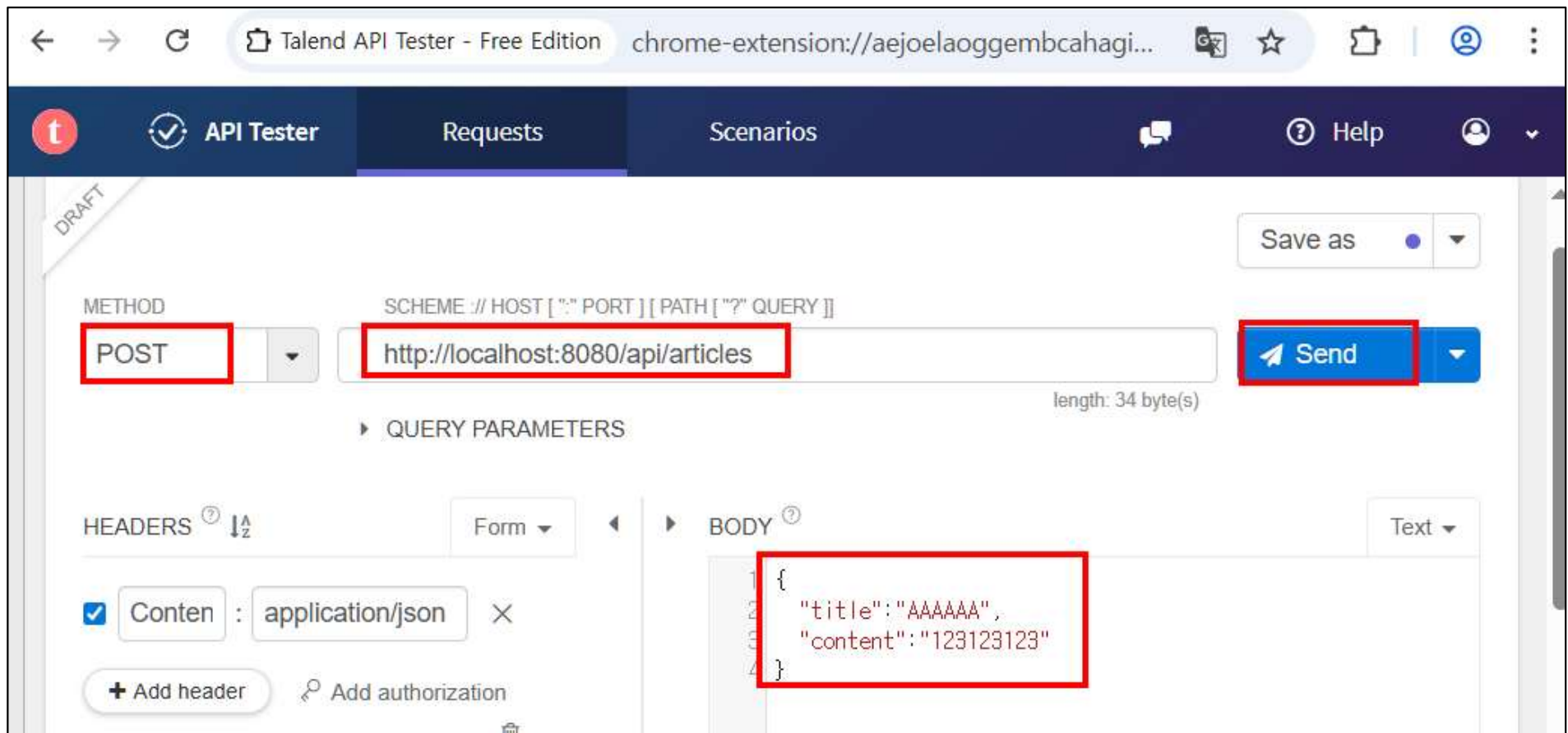
```
@RestController
public class ArticleApiController {
    @Autowired
    private ArticleRepository articleRepository;
    (중략)

    // POST
    @PostMapping("/api/articles")
    public Article create(ArticleForm dto){
        Article article = dto.toEntity();
        return articleRepository.save(article);
    }

    // PATCH
    // DELETE
}
```

REST API: POST 구현하기

- 서버 재실행
- Talend API Tester에서 게시물 생성 요청
 - 메서드: POST
 - URL: <http://localhost:8080/api/articles>
 - BODY에 생성할 데이터를 JSON 형식으로 입력



REST API: POST 구현하기

- 응답 결과 확인

Response

Cache Detected - Elapsed Time: 455ms

200

HEADERS ?

pretty ▼

```
Content-Type: application/json;charset=UTF-8
Transfer-Enc... chunked
Date: Tue, 25 Nov 2025 12:46:49 GMT
Keep-Alive: timeout=60
Connection: keep-alive
```

▶ COMPLETE REQUEST HEADERS

BODY ?

pretty ▼

```
{
  id: 4,
  title: null,
  content: null
}
```

[lines](#) [nums](#) [copy](#)

length: 36 bytes

REST API: POST 구현하기

- **@RequestBody**
 - REST API에서 요청 메시지의 본문(Body)에 실려오는 데이터를 받아오기 위해서 사용하는 어노테이션

api/ArticleApiController.java

```
// POST
@PostMapping("/api/articles")
public Article create(@RequestBody ArticleForm dto){
    Article article = dto.toEntity();
    return articleRepository.save(article);
}
```


REST API: POST 구현하기

- 서버 재실행
- Talend API Tester에서 게시물 생성 요청
 - 메서드: POST
 - URL: <http://localhost:8080/api/articles>
 - BODY에 생성할 데이터를 JSON 형식으로 입력

The screenshot shows the Talend API Tester interface with the following configuration:

- METHOD:** POST (highlighted with a red box)
- URL:** `http://localhost:8080/api/articles` (highlighted with a red box)
- Content-Type:** application/json (checked)
- BODY:**

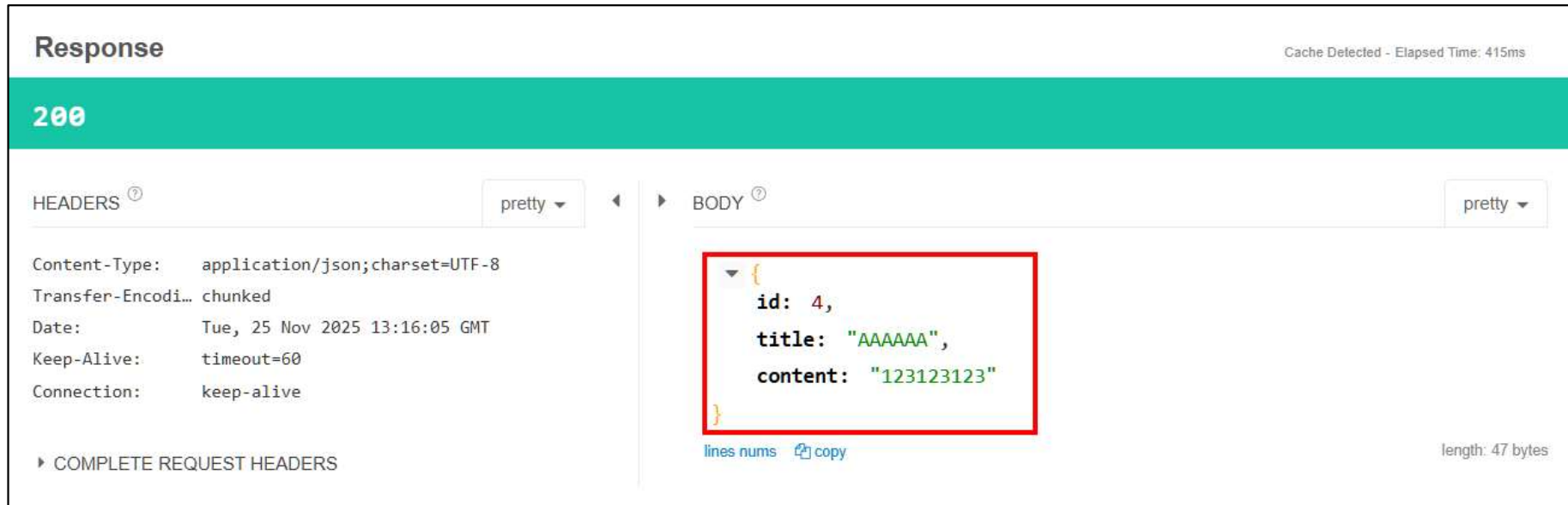
```
{
  "title": "AAAAAA",
  "content": "123123123"
}
```

 (highlighted with a red box)

The interface also shows tabs for 'API Tester', 'Requests', and 'Scenarios', and a 'Send' button.

REST API: POST 구현하기

- 응답 결과 확인



The screenshot shows a REST client interface with a response status of 200. The response body is a JSON object: `{id: 4, title: "AAAAAA", content: "123123123"}`. The JSON is highlighted with a red box. The headers section shows `Content-Type: application/json; charset=UTF-8`. The interface also includes a 'pretty' dropdown and a 'length: 47 bytes' indicator.

```
Response
```

Cache Detected - Elapsed Time: 415ms

200

HEADERS pretty BODY pretty

Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 25 Nov 2025 13:16:05 GMT
Keep-Alive: timeout=60
Connection: keep-alive

COMPLETE REQUEST HEADERS

```
{  
  id: 4,  
  title: "AAAAAA",  
  content: "123123123"  
}
```

length: 47 bytes

- <https://localhost:8080/articles> 접속



The screenshot shows a web browser at `localhost:8080/articles`. The page has a navbar with a hamburger menu icon. Below the navbar is a table with columns 'Id', 'Title', and 'Content'. The table contains four rows of data. At the bottom, there is a 'New Article' link and a footer with '© CloudStudying | Privacy | Terms'.

Id	Title	Content
1	가가가가	1111
2	나나나나	2222
3	다다다다	3333
4	AAAAAA	123123123

[New Article](#)

© CloudStudying | [Privacy](#) | [Terms](#)

REST API: PATCH 구현하기

- 게시물 수정 요청을 받아 처리하는 update() 만들기

api/ArticleApiController.java

```
// PATCH
@PatchMapping("/api/articles/{id}")
public Article update(@PathVariable Long id, @RequestBody ArticleForm dto){
    // 1. DTO를 엔티티로 변환하기
    // 2. 타킷 조회하기
    // 3. 잘 못된 요청 처리하기
    // 4. 업데이트 및 정상 응답(200) 하기
}
```

```

@Sf4j
@RestController
public class ArticleApiController {

    (중략)

    // PATCH
    @PatchMapping("/api/articles/{id}")
    public ResponseEntity<Article> update(@PathVariable Long id, @RequestBody ArticleForm dto){
        // 1. DTO를 엔티티로 변환하기
        Article article = dto.toEntity();
        log.info("id: {}, article: {}", id, article.toString());
        // 2. 타킷 조회하기
        Article target = articleRepository.findById(id).orElse( other: null);
        // 3. 잘 못한 요청 처리하기
        if(target == null || id != article.getId()) {
            // 400, 잘 못한 요청 응답
            log.info("잘 못한 요청! id: {}, article: {}", id, article.toString());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
        }
        // 4. 업데이트 및 정상 응답(200) 하기
        Article updated = articleRepository.save(article);
        return ResponseEntity.status(HttpStatus.OK).body(updated);
    }

    // DELETE
}

```

log.info()에서 {} 플레이스 홀더 사용하기

- 사용법 예

- log.info("메시지 {}", 값);
- log.info("메시지 {}, {}, {}", 값1, 값2, 값3);

- 예

- *log*.info("id: {}, article: {}", id, article.toString());
- log.info("Order created. User: {}, Product: {}, Quantity: {}, Price: {}",
userId, productName, quantity, price);

ResponseEntity와 HttpStatus

- **ResponseEntity**

- org.springframework.http.ResponseEntity<T>
- REST 컨트롤러의 반환형
- REST API의 응답을 위해 사용하는 클래스
- REST API의 요청을 받아 응답할 때,
ResponseEntity 객체에 HTTP 상태 코드, 헤더, body를 실어 보냄

- **HttpStatus**

- org.springframework.http.HttpStatus
- HTTP의 상태 코드(200, 404, 500 등)를 관리하는 Enum 클래스
 - ✓ 요청 성공: 200, HttpStatus.OK
 - ✓ 잘못된 요청: 400, HttpStatus.BAD_REQUEST
 - ✓ 접근 권한 없음: 403, HttpStatus.FORBIDDEN
 - ✓ 리소스 없음: 404, HttpStatus.NOT_FOUND
 - ✓ 서버 내부 오류: 500, HttpStatus.INTERNAL_SERVER_ERROR
- REST API의 응답상태를 표현함

REST API: PATCH 구현하기

- 서버 재실행
- Talend API Tester에서 게시물 생성 요청
 - 메서드: POST
 - URL: <http://localhost:8080/api/articles>
 - BODY에 생성할 데이터를 JSON 형식으로 입력

The screenshot shows the Talend API Tester interface. The top navigation bar includes 'API Tester', 'Requests', and 'Scenarios'. The main area is titled 'DRAFT' and contains the following fields:

- METHOD:** A dropdown menu with 'POST' selected.
- URL:** A text input field containing 'http://localhost:8080/api/articles'.
- Send:** A blue button with a paper plane icon.
- QUERY PARAMETERS:** A section with a plus icon and the text 'QUERY PARAMETERS'.
- HEADERS:** A section with a plus icon and the text 'HEADERS'.
- Body:** A section with a plus icon and the text 'BODY'. It contains a JSON object:

```
{  "title": "AAAAAA",  "content": "123123123"}
```

At the bottom, there are buttons for '+ Add header' and 'Add authorization'.

REST API: PATCH 구현하기

- 응답 결과 확인

Response

200

HEADERS ?

pretty ▼

Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 25 Nov 2025 15:18:45 GMT
Keep-Alive: timeout=60
Connection: keep-alive

▶ COMPLETE REQUEST HEADERS

BODY ?

{
 id: 1,
 title: "ABCABCABC",
 content: "aaaaaaaa"
}

lines nums copy

REST API: PATCH 구현하기

- 잘못된 요청 보내기 1

The screenshot shows a REST client interface with the following details:

- METHOD:** PATCH
- URL:** http://localhost:8080/api/articles/1
- QUERY PARAMETERS:** (Collapsed)
- HEADERS:** Content-Type: application/json
- BODY:** {
 "id": 3,
 "title": "ABCABCABC",
 "content": "????????"
}

- 로그 확인

Controller : 잘 못된 요청! id: 1, article: Article(id=3, title=ABCABCABC, content=????????)

- 응답 결과

The screenshot shows the response details in a REST client:

- Status:** 400
- HEADERS:** Content-Length: 0 byte, Date: Tue, 25 Nov 2025 15:23:15 GMT, Connection: close
- BODY:** No Content

REST API: PATCH 구현하기

- 잘못된 요청 보내기 2

The screenshot shows a REST client interface with the following details:

- METHOD:** PATCH
- URL:** http://localhost:8080/api/articles/100 (The '100' is underlined in red)
- Length:** 38 byte(s)
- HEADERS:** Content-Type: application/json
- BODY:**

```
{
  "id": 100,
  "title": "ABCABCABC",
  "content": "aaaaaaaa"
}
```

- 응답 결과

The screenshot shows the response details in a REST client interface:

- Status:** 400 (highlighted in a red bar)
- Cache Detected - Elapsed Time:** 384ms
- HEADERS:** Content-Type: application/json; charset=UTF-8, Transfer-Encoding: chunked, Date: Tue, 25 Nov 2025 15:35:58 GMT, Connection: close
- BODY:**

```
{
  "timestamp": "2025-11-25T15:35:58.199+00:00",
  "status": 400,
  "error": "Bad Request",
  "path": "/api/articles/100"
}
```

REST API: PATCH 구현하기- 일부 데이터 수정

- 데이터의 일부만 수정 요청

DRAFT

Save as [dropdown]

METHOD: **PATCH** SCHEME://HOST[:PORT][PATH["?" QUERY]]
http://localhost:8080/api/articles/1 length: 36 byte(s)

Send [button]

QUERY PARAMETERS

HEADERS [icon] [icon] [icon]
Form [dropdown]
☒ Content-Type: application/json [X]
+ Add header [icon] Add authorization [icon]

BODY [icon]
Text [dropdown]

```
{
  "id": 1,
  "content": "aaaaaaaa"
}
```

- 응답 결과 확인

Response Cache Detected - Elapsed Time: 212ms

200

HEADERS [icon] pretty [dropdown]
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 25 Nov 2025 15:42:27 GMT
Keep-Alive: timeout=60
Connection: keep-alive
COMPLETE REQUEST HEADERS

BODY [icon] pretty [dropdown]

```
{
  id: 1,
  title: null,
  content: "aaaaaaaa"
}
```


lines nums [icon] copy length: 42 bytes

REST API: PATCH 구현하기- 일부 데이터 수정

- 일부 데이터만 수정하기 위해 기존 데이터(target)에 수정할 새 데이터(article)를 붙여줌
 - Article 클래스에 patch() 만들기

```
public class Article {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    @Column  
    private String title;  
    @Column  
    private String content;  
  
    public void patch(Article article) {  
        if(article.title != null)  
            this.title = article.title;  
        if(article.content != null)  
            this.content = article.content;  
    }  
}
```

REST API: PATCH 구현하기- 일부 데이터 수정

- api/ArticleApiController.java 수정

```
// PATCH
@PatchMapping("/api/articles/{id}")
public ResponseEntity<Article> update(@PathVariable Long id, @RequestBody ArticleForm dto){
    // 1. DTO를 엔티티로 변환하기
    Article article = dto.toEntity();
    log.info("id: {}, article: {}", id, article.toString());
    // 2. 타킷 조회하기
    Article target = articleRepository.findById(id).orElse( other: null);
    // 3. 잘 못한 요청 처리하기
    if(target == null || id != article.getId()) {
        // 400, 잘 못한 요청 응답
        log.info("잘 못한 요청! id: {}, article: {}", id, article.toString());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
    }
    // 4. 업데이트 및 정상 응답(200) 하기
    target.patch(article);
    Article updated = articleRepository.save(target);
    return ResponseEntity.status(HttpStatus.OK).body(updated);
}
```

REST API: PATCH 구현하기 - 일부 데이터 수정

- 서버 재실행
- 데이터의 일부만 수정 요청

The screenshot shows a REST client interface with the following details:

- METHOD:** PATCH (highlighted with a red box)
- URL:** http://localhost:8080/api/articles/1
- Send Button:** A blue button with a paper plane icon and the text "Send" (highlighted with a red box).
- HEADERS:** Content-Type: application/json (checked with a blue box).
- BODY:** A JSON object:

```
{  "id": 1,  "content": "aaaaaaaaa"}
```

 (highlighted with a red box).

- 응답 결과 확인

The screenshot shows the response of the PATCH request in a REST client interface:

- Response Status:** 200 (highlighted with a green bar).
- HEADERS:** Content-Type: application/json; charset=UTF-8, Transfer-Encoding: chunked, Date: Tue, 25 Nov 2025 15:58:15 GMT, Keep-Alive: timeout=60, Connection: keep-alive.
- BODY:** A JSON object:

```
{  "id": 1,  "title": "가가가가",  "content": "aaaaaaaaa"}
```

 (highlighted with a green box).

REST API: DELETE 구현하기

- DELETE 요청을 처리할 delete() 만들기

- @DeleteMapping
- 반환형: ResponseEntity<Article>

api/ArticleApiController.java

```
// DELETE
@DeleteMapping("/api/articles/{id}")
public ResponseEntity<Article> delete(@PathVariable Long id){
    // 1. 대상 찾기
    // 2. 잘못된 요청 처리하기
    // 3. 대상 삭제하기
}
```


REST API: DELETE 구현하기

- api/ArticleApiController.java

```
// DELETE
@DeleteMapping("/api/articles/{id}")
public ResponseEntity<Article> delete(@PathVariable Long id){
    // 1. 대상 찾기
    Article target = articleRepository.findById(id).orElse( other: null);
    // 2. 잘못된 요청 처리하기
    if(target == null){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
    }
    // 3. 대상 삭제하기
    articleRepository.delete(target);
    return ResponseEntity.status(HttpStatus.OK).build();
}
```

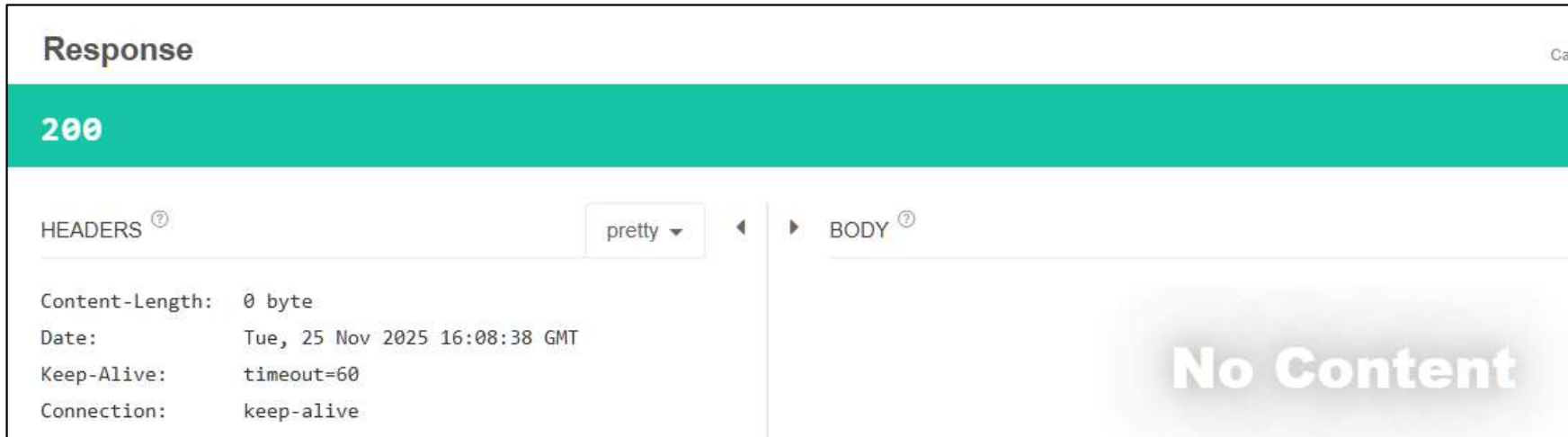
REST API: DELETE 구현하기

- 서버 재실행
- DELETE 요청 보내기



A screenshot of a REST client interface. The top left corner has a 'DRAFT' label. The main area shows a request configuration. The 'METHOD' dropdown is set to 'DELETE'. The 'URL' field contains 'http://localhost:8080/api/articles/1'. To the right of the URL is a 'Send' button. Below the URL, there is a 'QUERY PARAMETERS' section. The top right corner has a 'Save as' button. The bottom right corner shows 'length: 36 byte(s)'.

- 응답 확인



A screenshot of a REST client interface showing the response of a DELETE request. The response status is '200'. The response headers are listed on the left, and the response body is shown on the right. The response body is empty, indicated by the text 'No Content'.

Response	
200	
HEADERS	
Content-Length:	0 byte
Date:	Tue, 25 Nov 2025 16:08:38 GMT
Keep-Alive:	timeout=60
Connection:	keep-alive
BODY	
No Content	