11장 HTTP와 REST 컨트롤러

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

REST API 동작 이해하기

- **HTTP 요청 메시지의 구조**

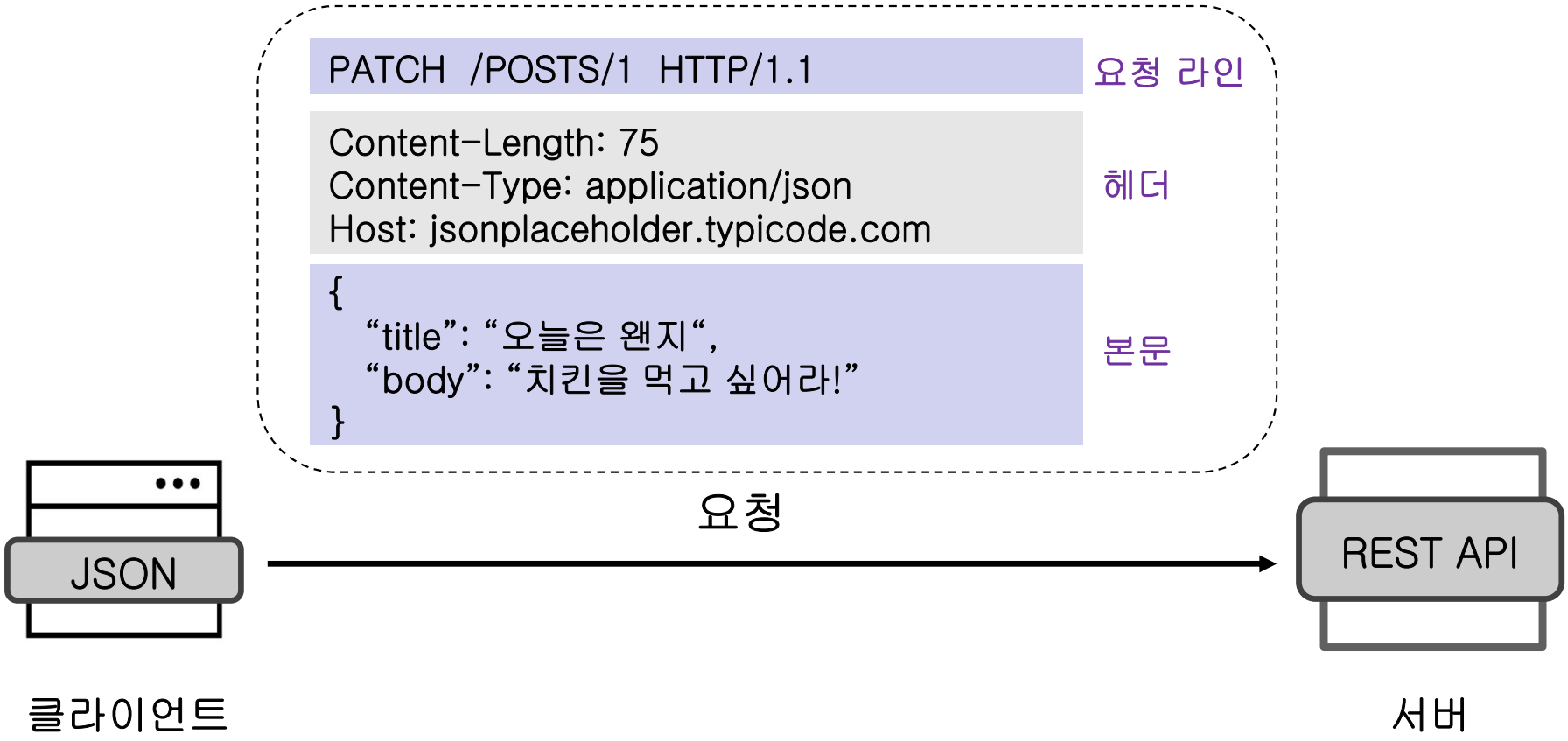


그림 11-1 요청 메시지 구조

REST API 동작 이해하기

- **HTTP 응답 메시지의 구조**

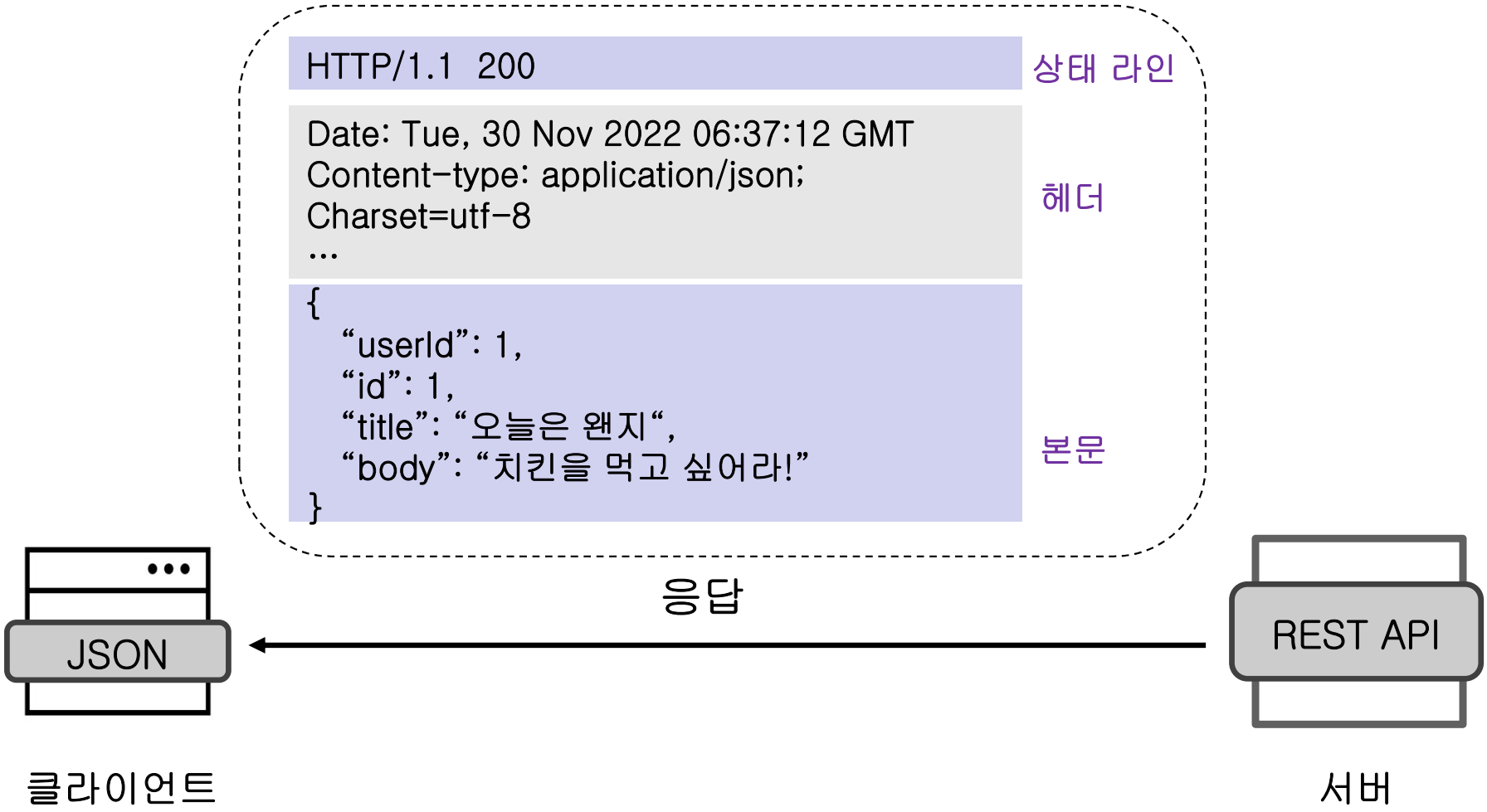


그림 11-2 응답 메시지 구조

- JSON 데이터 예시

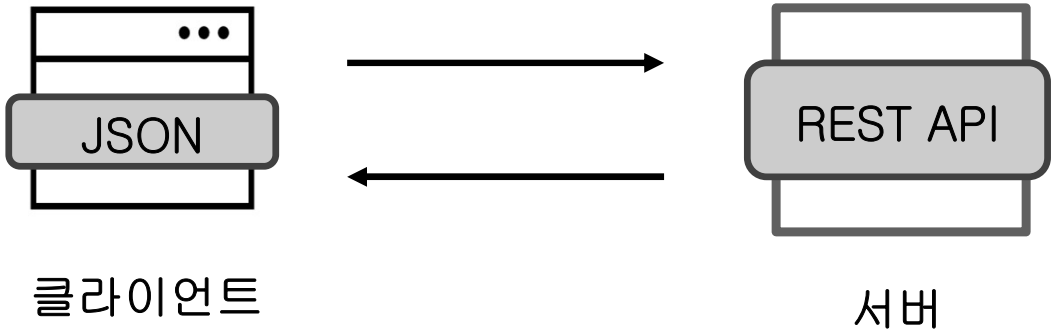
```
{
        “id”: 1,
        “name”: “Park”,
        address: {
                “street”: “Nambu Street 151”,
                “suite”: “Central Villat 301”,
                …
        },
        “likes”: [“singing”, “jogging”, “writing”]
}
```

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 구애받지 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}



REST API URL

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

그림 11-4 REST API 주소 설계

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

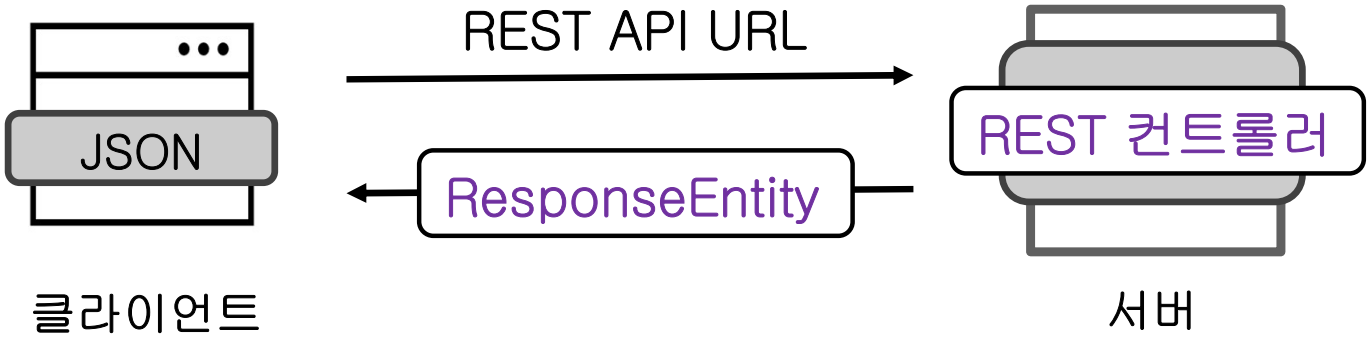


그림 11-5 RES 컨트롤러와 ResponseEntity의 역할

REST 컨트롤러 맛보기

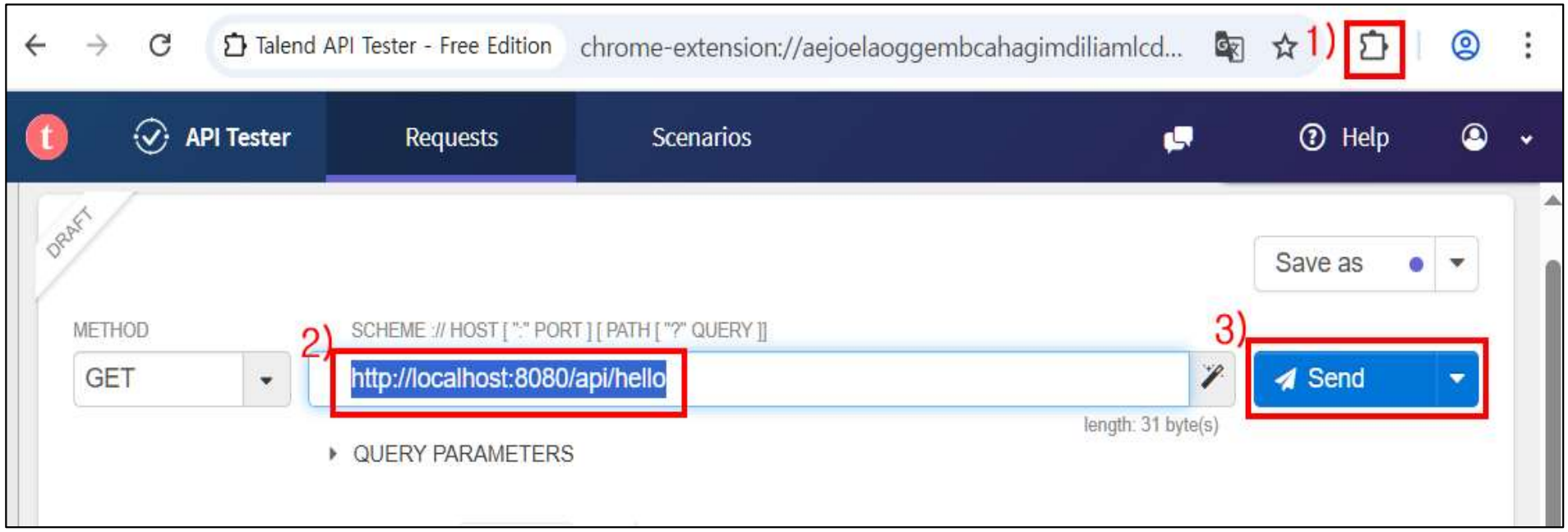
- **패키지 생성: api**
  - [src-main-java - com.example.firstproject.api]
- **api 패키지에 FirstApiController 클래스 생성**
  - @RestController
    - ✓ REST 컨트롤러

FirstApiController.java

```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

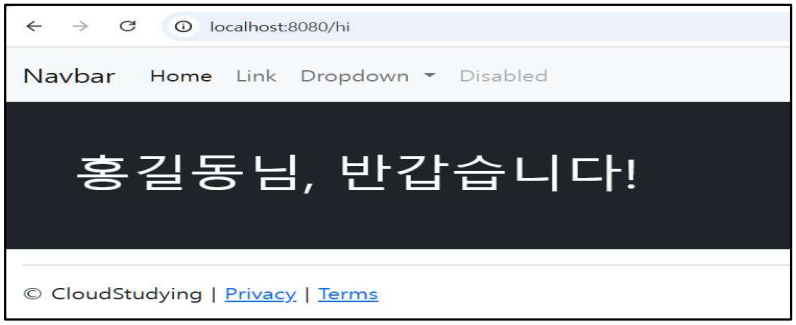
- http://localhost:8080/api/hello

REST 컨트롤러와 일반 컨트롤러의 차이

- 2장에서 구현한 FirstController

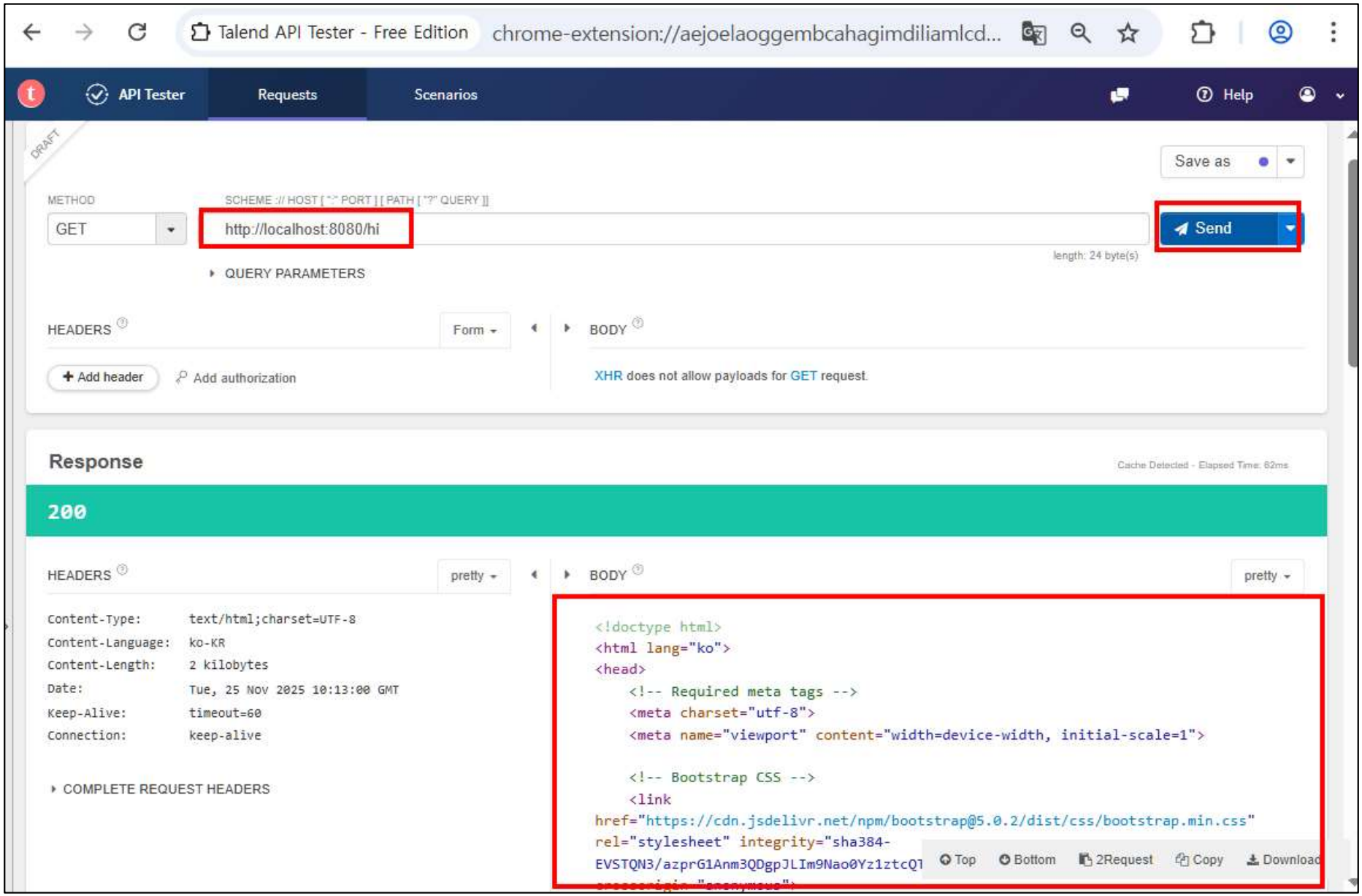
```java
@Controller
public class FirstController {

    @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



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 게시글 조회

- com.example.firstproject.api에 ArticleApiController 클래스 생성

ArticleApiController.java

```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

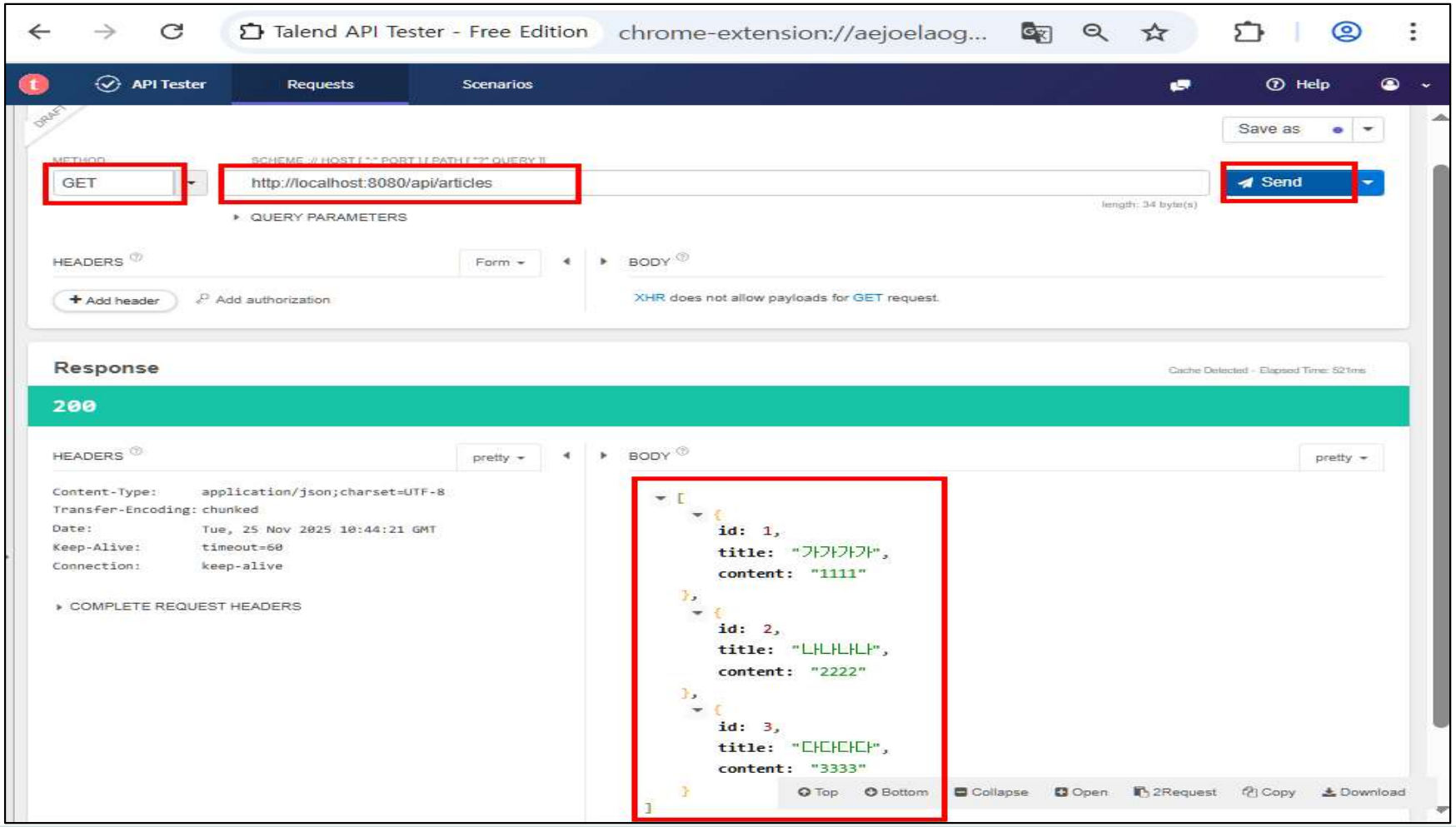
- GET 요청을 처리하는 index() 메서드

api/ArticleController.java

```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

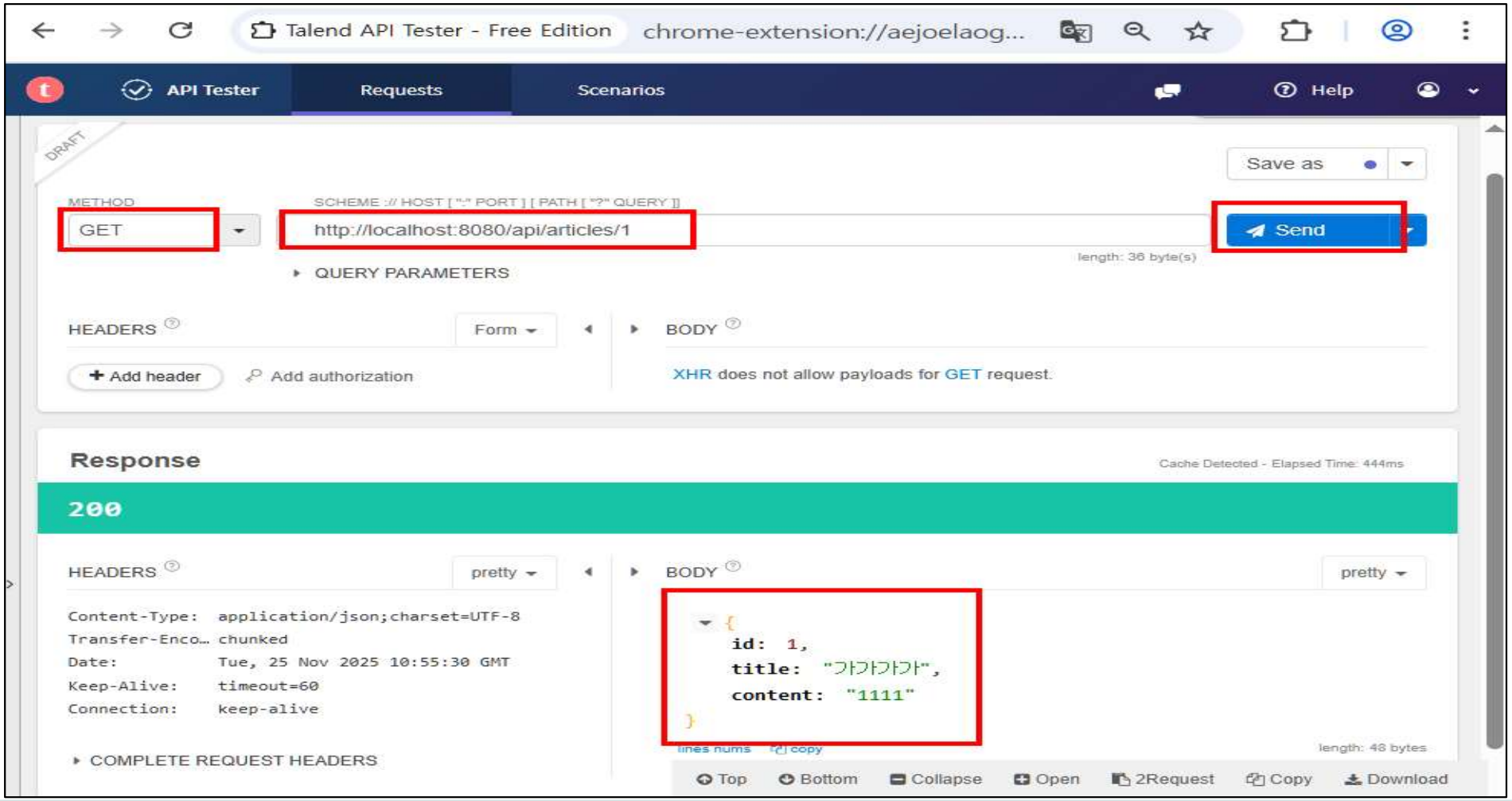
REST API: GET 구현하기 – 단일 게시글 조회

- 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



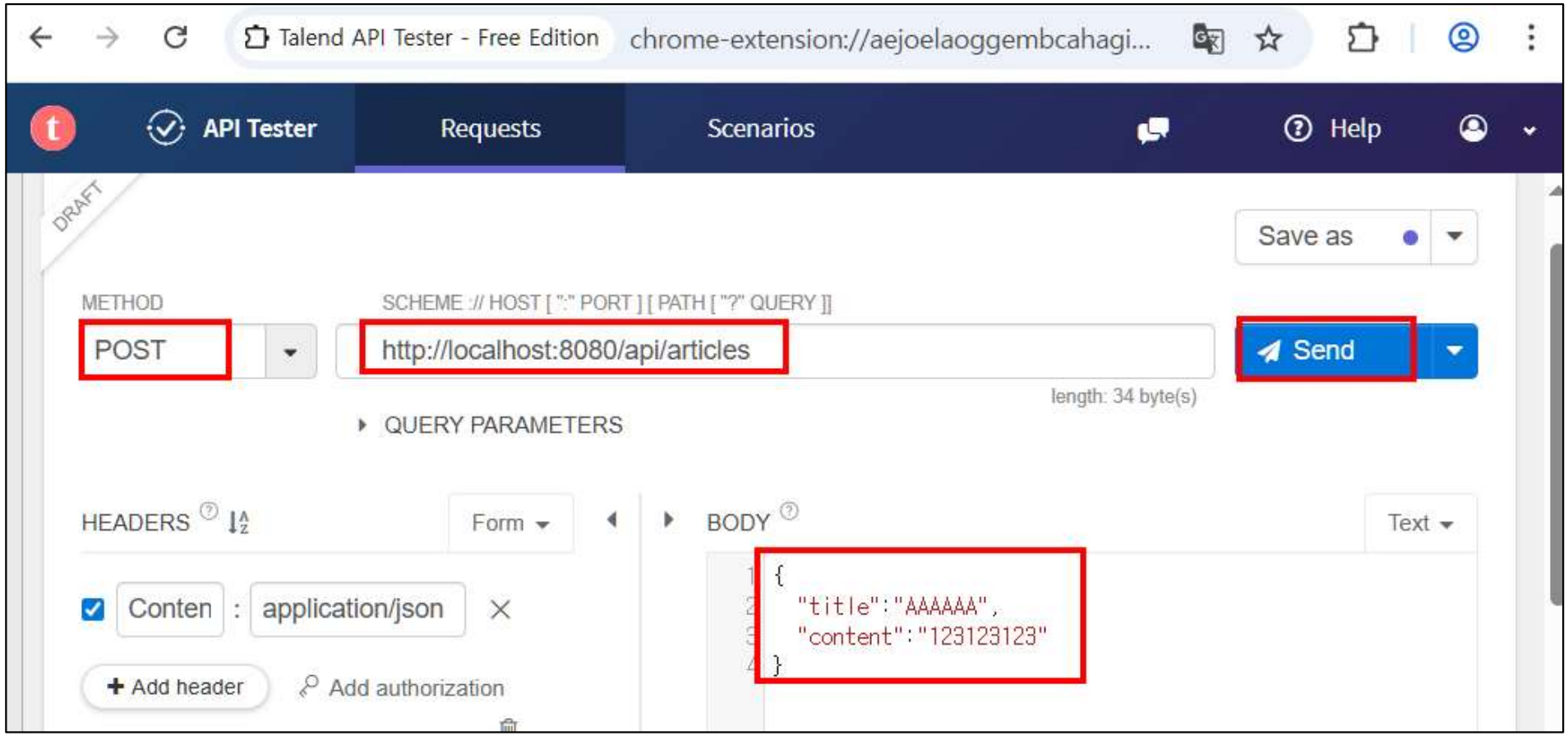
REST API: POST 구현하기

- 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

REST API: POST 구현하기

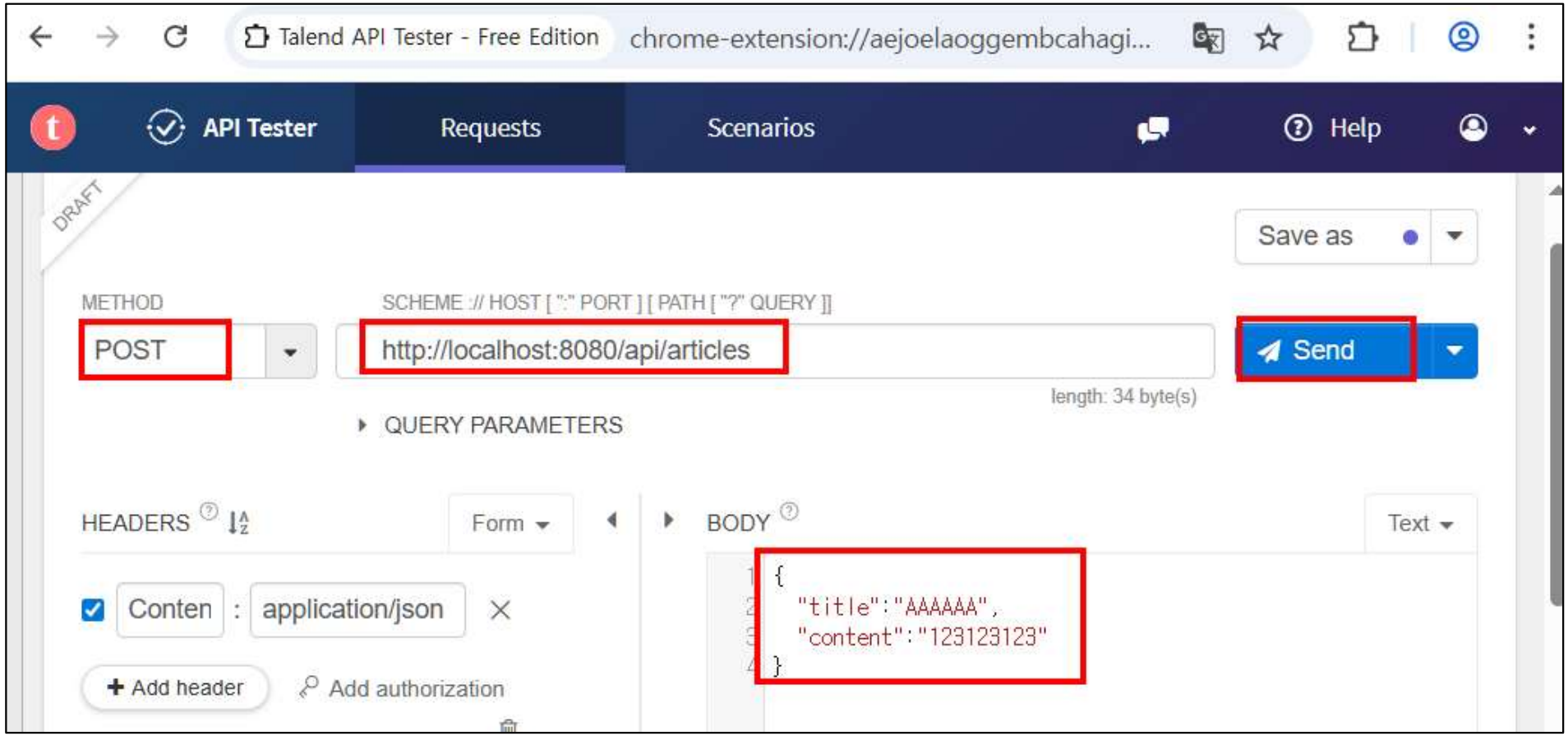
- **@RequestBody**
  - REST API에서 요청 메시지의 본문(Body)에 실려오는 데이터를 받아오기 위해서 사용하는 어노테이션

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

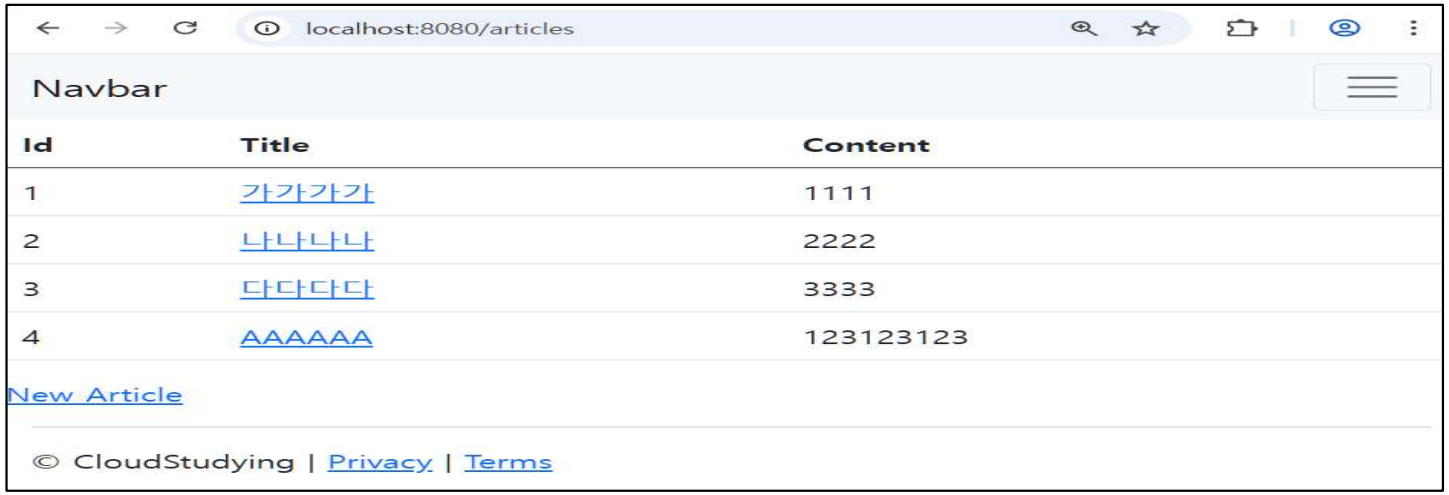
- **서버 재실행**
- **Talend API Tester에서 게시물 생성 요청**
  - 메서드: POST
  - URL: http://localhost:8080/api/articles
  - BODY에 생성할 데이터를 JSON 형식으로 입력



REST API: POST 구현하기

- 응답 결과 확인

- https://localhost:8080/articles 접속



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. DTO을 엔티티로 변환하기
    // 2. 타킷 조회하기
    // 3. 잘 못된 요청 처리하기
    // 4. 업데이트 및 정상 응답(200) 하기
}
```

```java
@Slf4j
@RestController
public class ArticleApiController {
    (중략)
    // PATCH
    @PatchMapping("/api/articles/{id}")
    public ResponseEntity<Article> update(@PathVariable Long id, @RequestBody ArticleForm dto){
        // 1. DTO을 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 못된 요청 처리하기
        if(target == null || id != article.getId()) {
            // 400, 잘 못된 요청 응답
            log.info("잘 못된 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);

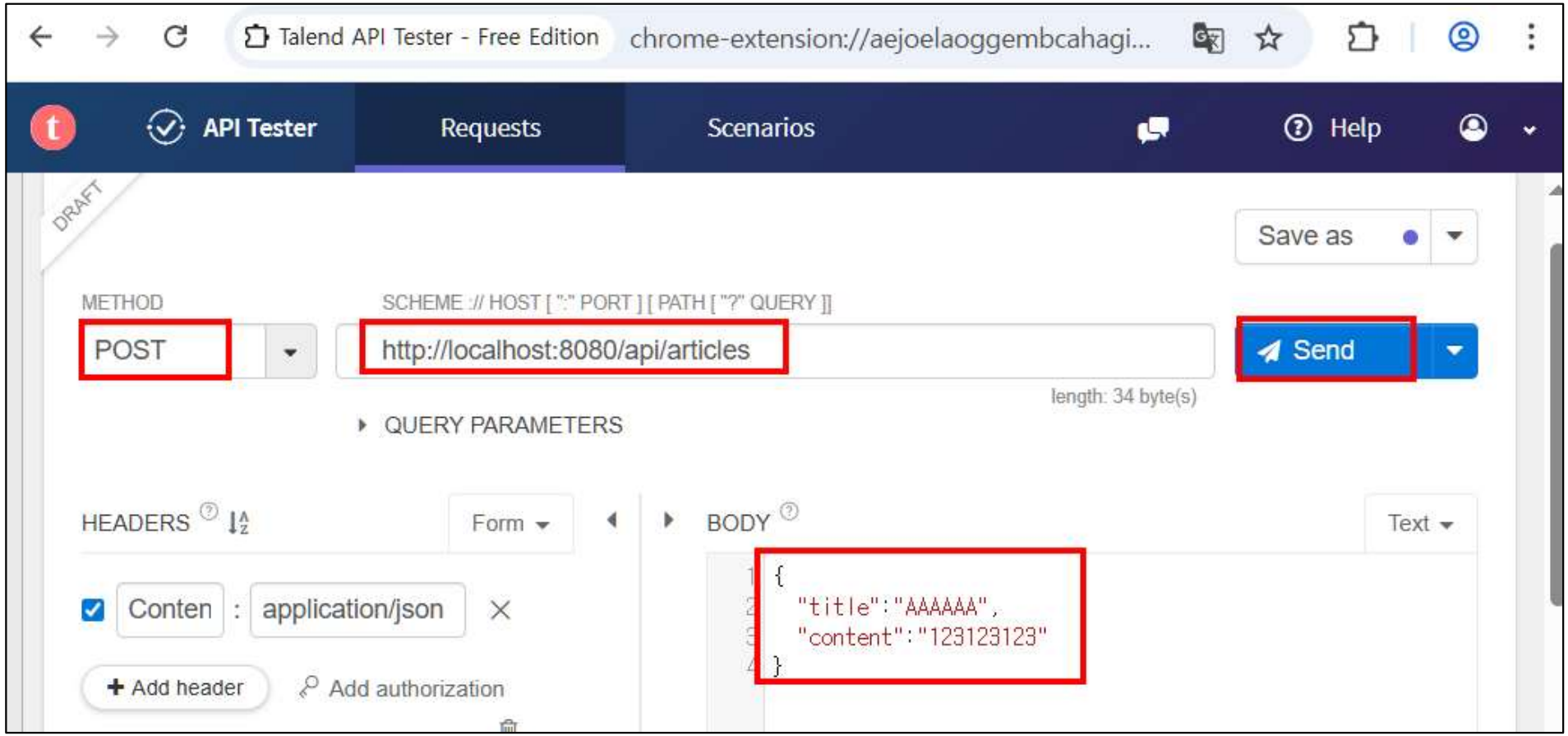
- **예**
  - *log*.info("id: {}, article: {}", id, article.toString());
  - log.info("Order created. User: {}, Product: {}, Quantity: {}, Price: {}",
    userId, productName, quantity, price);

ResponseEntity와 HttpStatus

- **ResponseEntity**
  - org.springframework.http.ResponseEntity〈T〉
  - REST 컨트롤러의 반환형
  - REST API의 응답을 위해 사용하는 클래스
  - REST API의 요청을 받아 응답할 때,
    ResposeEntity 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.BAC_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



REST API: PATCH 구현하기

- 응답 결과 확인

REST API: PATCH 구현하기

- 잘못된 요청 보내기 1

- 로그 확인

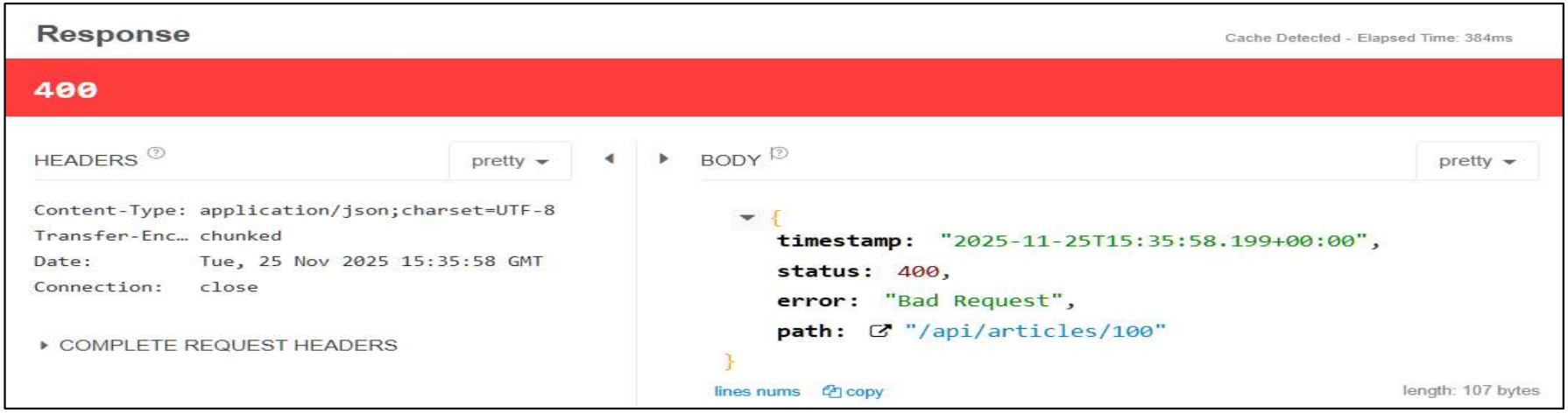
```
Controller              : 잘 못된 요청! id: 1, article: Article(id=3, title=ABCABCABC, content=@@@@@@@@)
```

- 응답 결과

REST API: PATCH 구현하기

- 잘못된 요청 보내기 2

- 응답 결과



REST API: PATCH 구현하기- 일부 데이터 수정

- 데이터의 일부만 수정 요청

- 응답 결과 확인

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

```java
// PATCH
@PatchMapping("/api/articles/{id}")
public ResponseEntity<Article> update(@PathVariable Long id, @RequestBody ArticleForm dto){
    // 1. DTO을 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 못된 요청 처리하기
    if(target == null || id != article.getId()) {
        // 400, 잘 못된 요청 응답
        log.info("잘 못된 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

- 응답 결과 확인

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

```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

- 응답 확인