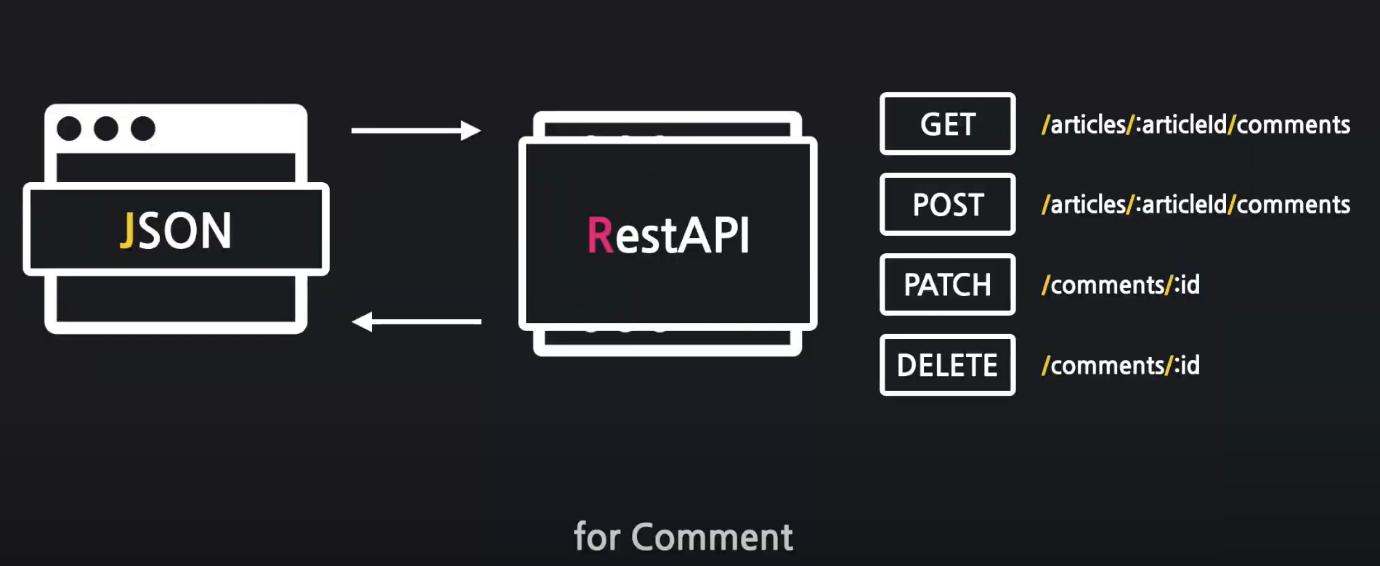
댓글 서비스와 컨트롤러(feat. REST API)



**✨ DTO와 Entity**

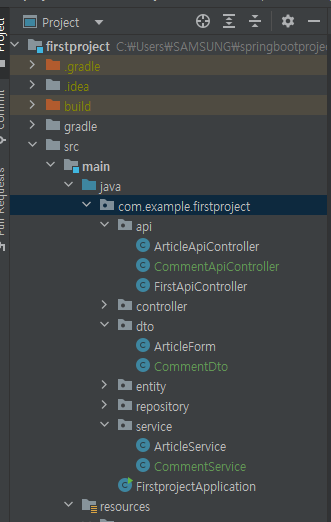
DTO와 ENTITY는 모두 소프트웨어 개발에서 데이터 모델링을 위해 사용되는 용어입니다.

DTO는 Data Transfer Object의 약어로, 데이터 전송을 위한 객체입니다. 주로 서로 다른 시스템 간 데이터를 전송하기 위해 사용됩니다. DTO는 일반적으로 비즈니스 로직을 포함하지 않으며, 단순히 데이터 전송을 위한 데이터 구조를 가지고 있습니다. 예를 들어, 웹 API의 요청 및 응답 데이터를 전송하기 위해 DTO를 사용할 수 있습니다.

반면, ENTITY는 데이터베이스에서 사용되는 객체입니다. 주로 비즈니스 로직을 포함하며, 데이터베이스에서 특정 테이블의 열과 일치하는 데이터를 저장합니다. ENTITY는 데이터베이스 테이블에 매핑되어 있으며, 엔티티 클래스는 데이터베이스 테이블의 열과 일치하는 필드를 가지고 있습니다.

즉, DTO는 데이터 전송을 위한 단순한 데이터 객체이며, ENTITY는 비즈니스 로직을 포함하고 데이터베이스와 직접 연결되어 있는 객체입니다. DTO는 시스템 간 데이터 전송에 사용되며, ENTITY는 데이터베이스에서 데이터를 저장, 검색 및 업데이트하는 데 사용됩니다.

**REST 컨트롤러,서비스,DTO 생성**



**CommentApiController**

package com.example.firstproject.api;

@RestController

public class CommentApiController {

@Autowired

private CommentService commentService;

// 댓글 목록 조회

// 댓글 생성

// 댓글 수정

// 댓글 삭제

}

**CommentService**

package com.example.firstproject.service;

@Service

public class CommentService {

@Autowired

private CommentRepository commentRepository;

@Autowired

private ArticleRepository articleRepository;

// 댓글이 달린 게시글도 가져와야하기 때문에 ArticleRepository도 함께 가져오기

}

**CommentDto**

package com.example.firstproject.dto;

@AllArgsConstructor

@NoArgsConstructor

@Getter

@ToString

public class CommentDto {

private Long id;

private Long articleId;

private String nickname;

private String body;

}

**댓글 목록 조회**

**CommentApiController**

* 요청 URL :  
  "/api/articles/4/comments" ->  
  "/api/articles/{articleId}/comments"
* service에게 목록조회 위임(comments( ))
* Entity -> DTO로 변환해서 반환

// 댓글 목록 조회

@GetMapping("/api/articles/{articleId}/comments")

public ResponseEntity<List<CommentDto>> comments(@PathVariable Long articleId) {

// List<Comment> -> List<CommentDto> Comment(엔티티)를 DTO로 만들어서 반환,

// 응답도 같이 보내주기 위해 ResponseEntity로 감싸기

// 서비스에게 위임

List<CommentDto> dtos = commentService.comments(articleId);

// 결과 응답(성공한 경우만 있다고 가정)

return ResponseEntity.status(HttpStatus.OK).body(dtos);

}

**CommentService**

* commentRepository통해 댓글 목록 조회(comments에 담기)
* CommentDto를 담고있는 빈 배열 선언(dtos)
* 댓글 목록 하나하나 꺼내서 Dto변환 메소드에 전달  
  createCommentDto(c)
* Dto변환된 값 배열에 add
* 배열 return

public List<CommentDto> comments(Long articleId) {

// 조회: 댓글 목록 조회(게시글 아이디를 통해 해당 게시글의 댓글 목록 조회)

List<Comment> comments = commentRepository.findByArticleId(articleId);

// 변환: 엔티티 -> DTO

// CommentApiController에서 List<Comment> -> List<CommentDto>로 반환하기로 했기때문에 엔티티 -> DTO로 변환

List<CommentDto> dtos = new ArrayList<CommentDto>();

// 비어있는 dtos에다가 댓글들을 변환해서 add하기

for(int i = 0; i < comments.size(); i++) {

// comments 값을 하나하나 꺼내서 넣기

Comment c = comments.get(i);

CommentDto dto = CommentDto.createCommentDto(c); // Dto로 변환

dtos.add(dto);

}

// 반환

return dtos;

}

**CommentDto**

* Comment(엔티티)를 받아와서 new CommentDto에 값넣고 return

public static CommentDto createCommentDto(Comment comment) {

return new CommentDto(

comment.getId(),

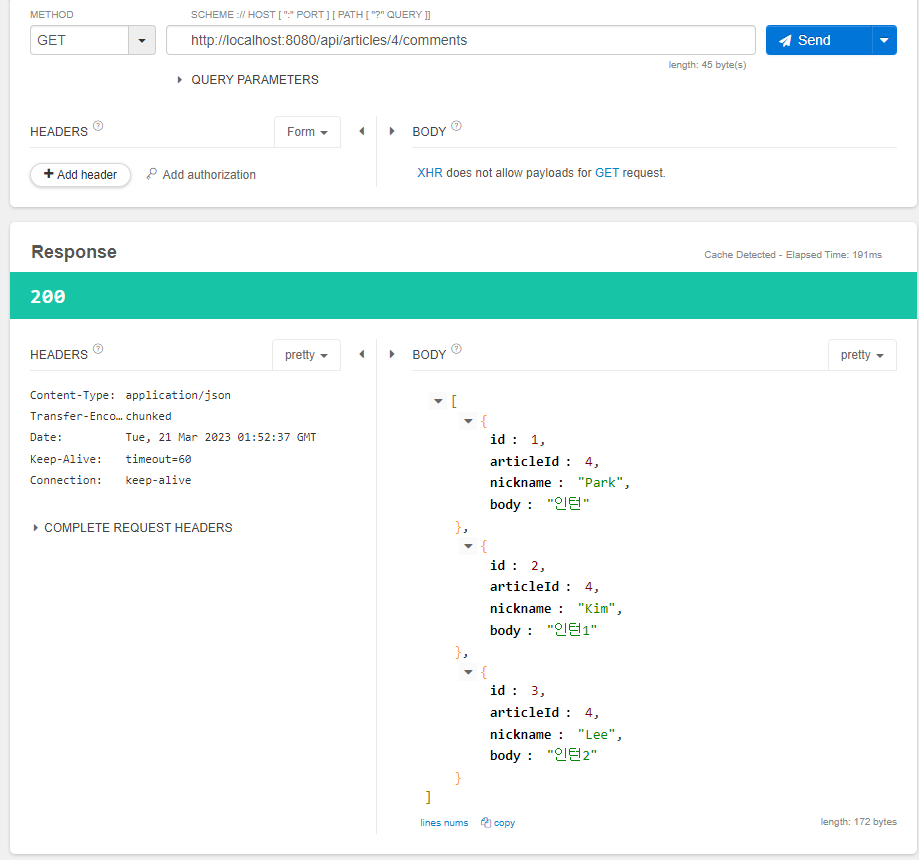
comment.getArticle().getId(), // Article을 가져오고 거기서 Id만 필요하기 때문에 다시 .getId()

comment.getNickname(),

comment.getBody()

);

}

"/api/articles/{articleId}/comments"요청에  
댓글목록 조회 성공 ✨😊  


**for문을 스트림으로**

CommentService에 for문을 스트림문법으로 바꿔보자.

**for문**

List<Comment> comments = commentRepository.findByArticleId(articleId);

List<CommentDto> dtos = new ArrayList<CommentDto>();

for(int i = 0; i < comments.size(); i++) {

Comment c = comments.get(i);

CommentDto dto = CommentDto.createCommentDto(c);

dtos.add(dto);

}

**stream**

위의 전체 코드가 아래 코드로 간단해질 수 있다.👍

return commentRepository.findByArticleId(articleId) // commentRepository에 목록조회

.stream() // stream으로 변경

.map(comment -> CommentDto.createCommentDto(comment)) //createCommentDto를 통해 comment를 하나하나전달하여 DTO로 변환

.collect(Collectors.toList()); // map이 반환하는 값이 stream<Object>이기 때문에

**댓글 생성하기**



**CommentApiController**

* 요청URL에서 전달된 articleId @PathVariable  
  POST 요청이기 때문에 @RequestBody사용
* 결과 응답( create 메소드안에서 에러를 발생시켜서 실패한경우에 응답코드를 자동으로 반환할 수 있도록 작성할 예정)

// 댓글 생성

@Transactional @PostMapping("/api/articles/{articleId}/comments")

public ResponseEntity<CommentDto> create(@PathVariable Long articleId,

@RequestBody CommentDto dto) {

// 서비스에게 위임

CommentDto createdDto = commentService.create(articleId,dto);

// 결과 응답( create 메소드안에서 에러를 발생시켜서 실패한경우에 응답코드를 자동으로 반환할 수 있도록 작성할 예정)

return ResponseEntity.status(HttpStatus.OK).body(createdDto);

}

**CommentService**

* @Transactional  
  create메소드는 DB에 영향을 주기 때문에  
  중간에 문제가 생기면 롤백될 수 있도록 @Transactional 추가하기 ✨

@Transactional

public CommentDto create(Long articleId, CommentDto dto) {

// 게시글 조회 및 예외 발생

// .orElseThrow(() -> new IllegalArgumentException()) article이 없다면 예외발생시켜서 다음 코드가 실행되지 않는다.

Article article = articleRepository.findById(articleId)

.orElseThrow(() -> new IllegalArgumentException("댓글 생성 실패!! 대상 게시글이 없습니다."));

// 댓글 엔티티 생성

Comment comment = Comment.createComment(dto, article);

// 댓글 엔티티를 DB로 저장

Comment created = commentRepository.save(comment);

// DTO로 변환하여 반환

return CommentDto.createCommentDto(created);

}

**Comment**

댓글 엔티티 생성하는 메소드.

public static Comment createComment(CommentDto dto, Article article) {

// 예외 발생

if(dto.getId() != null) // 받아온 데이터에 id가 있다면

throw new IllegalArgumentException("댓글 생성 실패! 댓글의 id가 없어야 합니다.");

if(dto.getArticleId() != article.getId()) // 요청url의 id(articleId)와 요청데이터에 article\_id가 다르면

throw new IllegalArgumentException("댓글 생성 실패! 게시글의 id가 잘못되었습니다.");

// 엔티티 생성 및 반환

return new Comment(

dto.getId(),

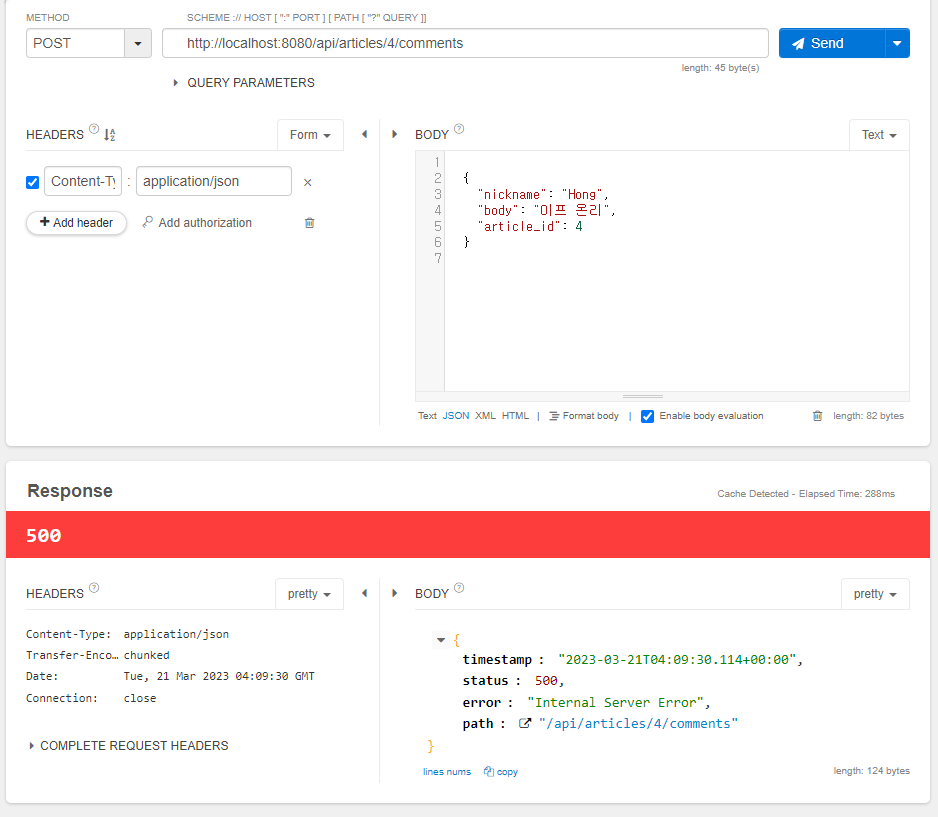
article,

dto.getNickname(),

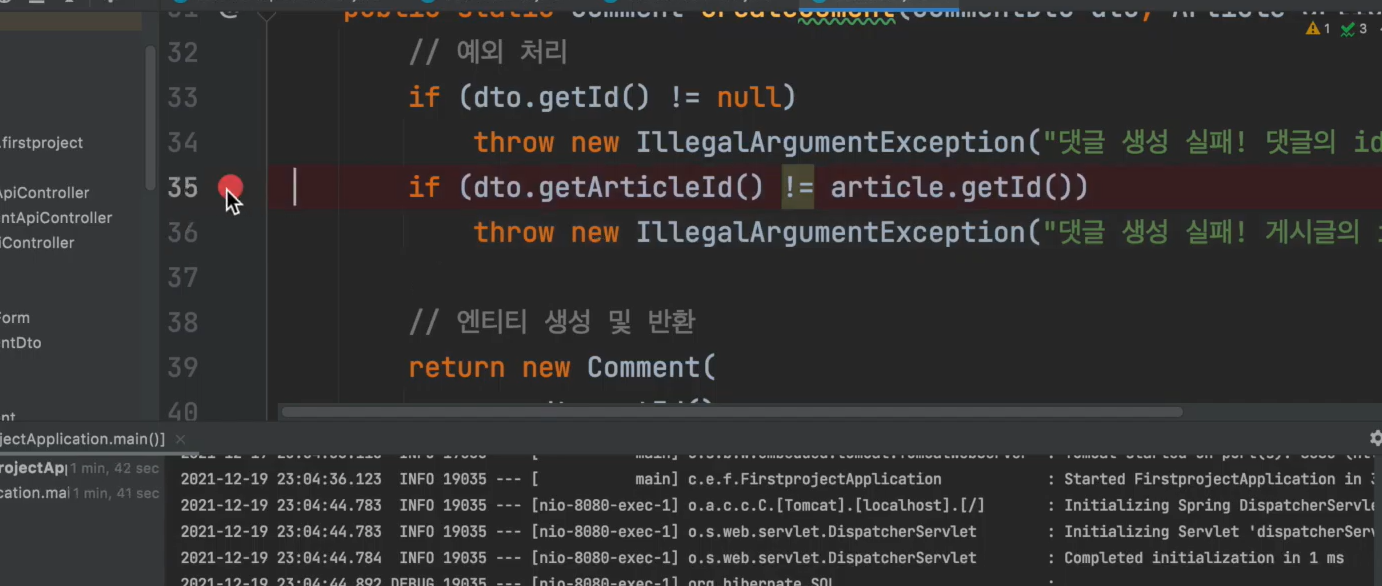
dto.getBody()

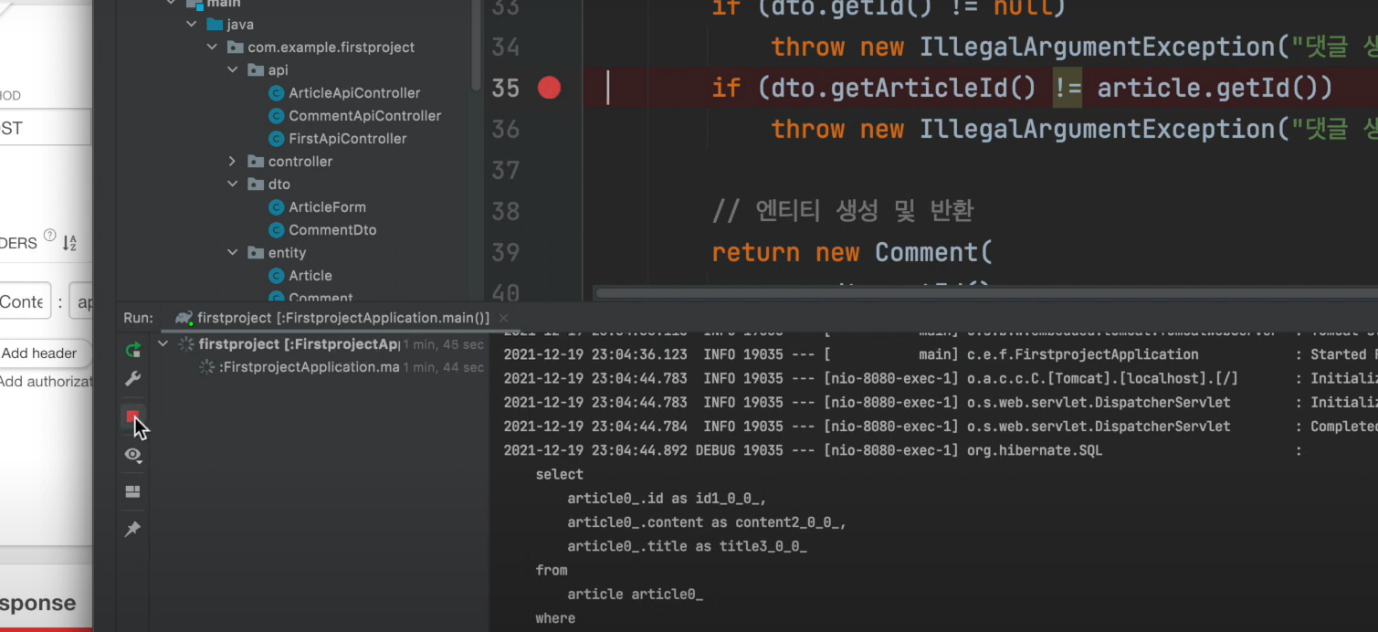
);

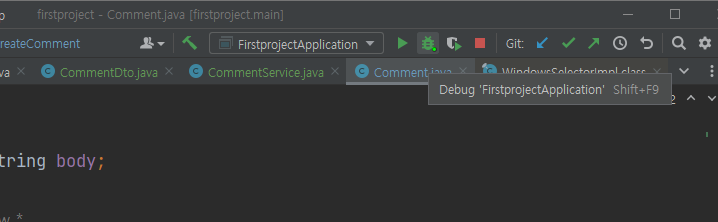
}

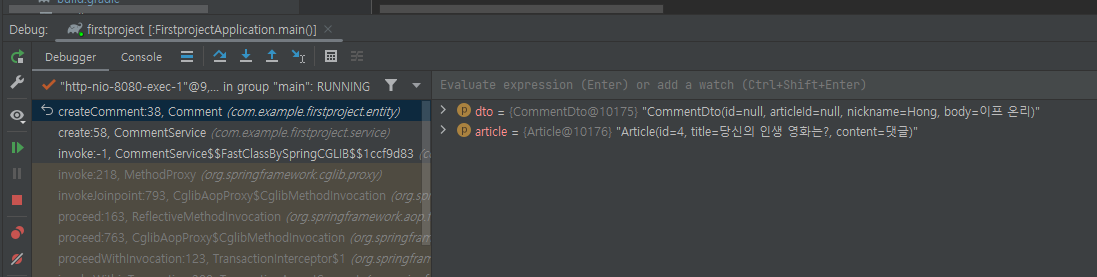
요청보냈는데 에러가난다..... ✨ 디버깅을 해보자!!!  


**디버깅**

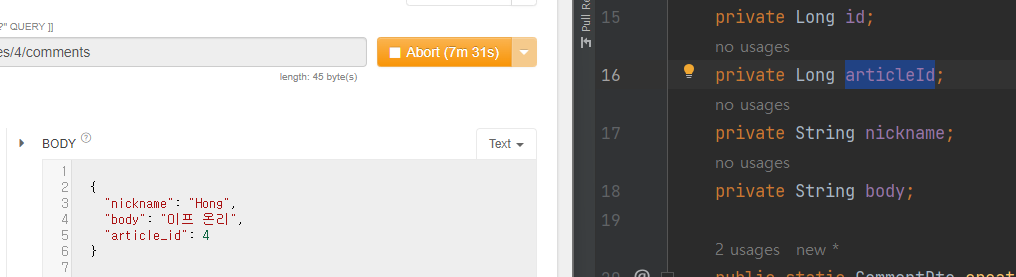
브레이크 포인트 선택  


서버 중지  


디버깅 누르기  


다시 Talend API 요청 보내고 로그 확인해보면  
**articleId가 null 이다**..!!!!  


Talend API에서 articleId를 보낼때 컬럼명으로 article\_id CommentDto에서 articleId

  
이렇게 이름이 다르기때문에 값을 제대로 받아올 수 없다!!

이처럼 JSON데이터를 받아 올 때 변수명이 다르면  
@JsonProperty("컬럼명")  
어노테이션을 추가해서 컬럼명을 명시해준다.

public class CommentDto {

private Long id;

@JsonProperty("article\_id")

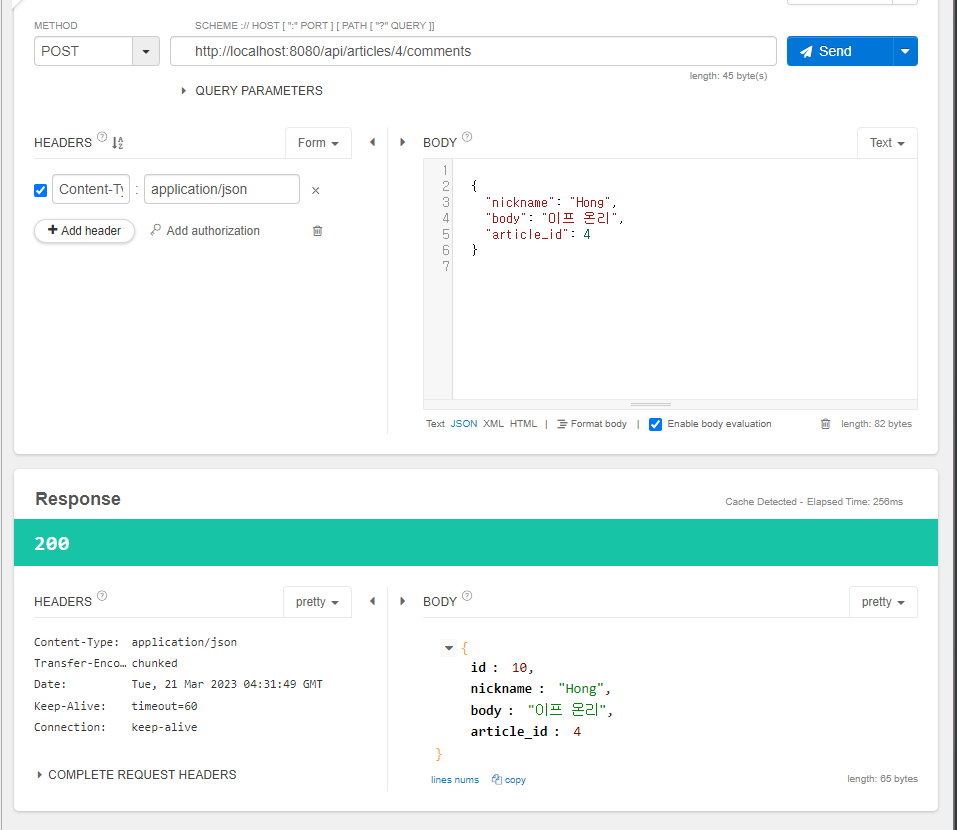
private Long articleId;

private String nickname;

private String body;

...

}

코드 수정후 다시 요청보내면 댓글 생성 성공✨✨  


**댓글 수정하기**



**CommentApiController**

// 댓글 수정

@PatchMapping("/api/comments/{id}")

public ResponseEntity<CommentDto> update(@PathVariable Long id,

@RequestBody CommentDto dto) {

// 서비스에게 위임

CommentDto updatedDto = commentService.update(id,dto);

// 결과 응답( create 메소드안에서 에러를 발생시켜서 실패한경우에 응답코드를 자동으로 반환할 수 있도록 작성할 예정)

return ResponseEntity.status(HttpStatus.OK).body(updatedDto);

}

**CommentService**

* DB에 영향을 끼치기 때문에 @Transactional 추가
* id로 댓글 찾기 (target)

@Transactional

public CommentDto update(Long id, CommentDto dto) {

// 댓글 조회 및 예외 발생

Comment target = commentRepository.findById(id)

.orElseThrow(() -> new IllegalArgumentException("댓글 수정 실패! 대상 댓글이 없습니다."));

// 댓글 수정

target.patch(dto);

// 댓글 DB로 갱신

Comment updated = commentRepository.save(target);

// 댓글 엔티티를 DTO로 변환 및 반환

return CommentDto.createCommentDto(updated);

}

**Comment**

patch 메소드 만들기.

public void patch(CommentDto dto) {

// 예외 발생

if(this.id != dto.getId()) { // 요청URL id와 댓글의 id가 다를때

throw new IllegalArgumentException("댓글 수정 실패! 잘못된 id가 입력되었습니다.");

}

// 객체를 갱신

if(dto.getNickname() != null) {

this.nickname = dto.getNickname();

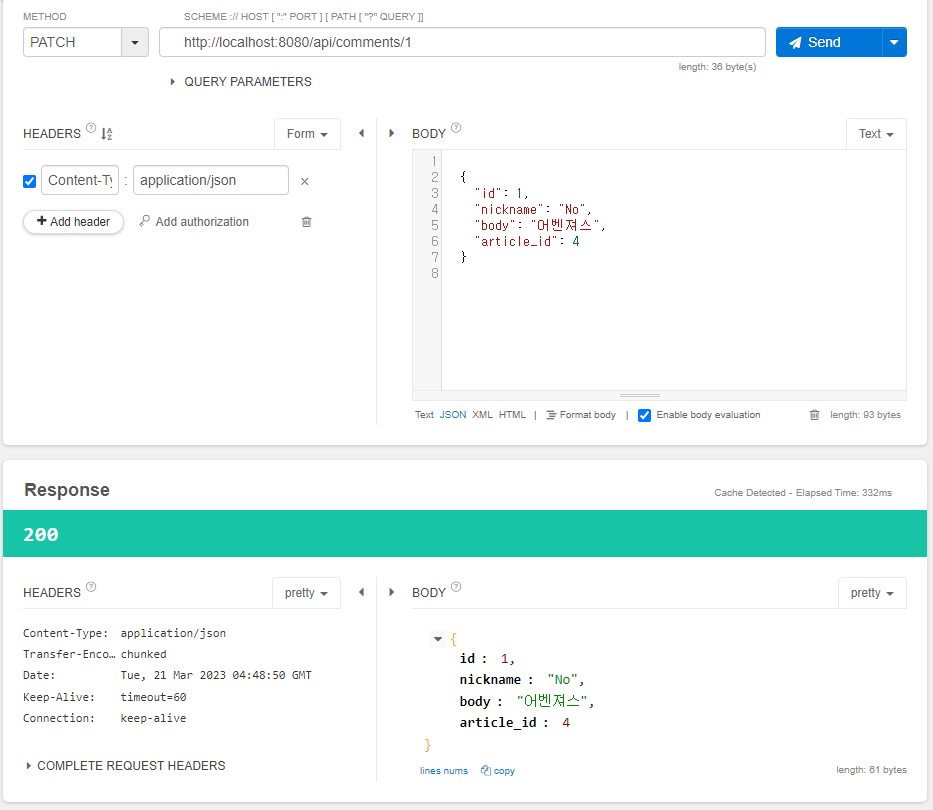
}

if(dto.getBody() != null) {

this.body = dto.getBody();

}

}

수정 완료 ✨✨  


**댓글 삭제**

**CommentApiController**

// 댓글 삭제

@DeleteMapping("/api/comments/{id}")

public ResponseEntity<CommentDto> delete(@PathVariable Long id) {

// 서비스에게 위임

CommentDto deletedDto = commentService.delete(id);

// 결과 응답

return ResponseEntity.status(HttpStatus.OK).body(deletedDto);

}

**CommentService**

@Transactional

public CommentDto delete(Long id) {

// 댓글 조회 및 예외 발생

Comment target = commentRepository.findById(id)

.orElseThrow(() -> new IllegalArgumentException("댓글 삭제 실패! 대상 댓글이 없습니다."));

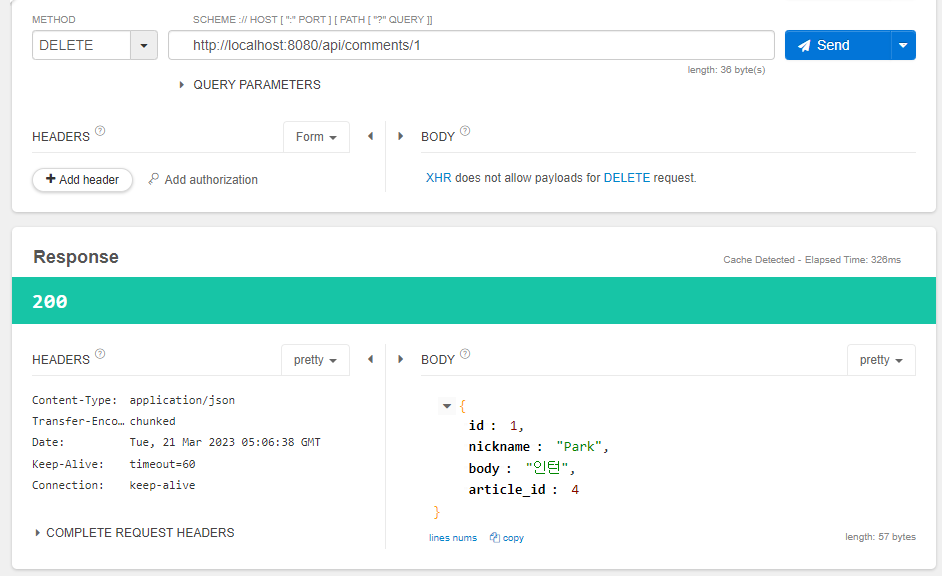
// 댓글 DB에서 삭제

commentRepository.delete(target);

// 삭제 댓글을 DTO로 반환

return CommentDto.createCommentDto(target);

}

삭제 성공 ✨  


**전체코드**

**CommentApiController**

package com.example.firstproject.api;

@RestController

public class CommentApiController {

@Autowired

private CommentService commentService;

// 댓글 목록 조회

@GetMapping("/api/articles/{articleId}/comments")

public ResponseEntity<List<CommentDto>> comments(@PathVariable Long articleId) {

// List<Comment> -> List<CommentDto> Comment(엔티티)를 DTO로 만들어서 반환,

// 응답도 같이 보내주기 위해 ResponseEntity로 감싸기

// 서비스에게 위임

List<CommentDto> dtos = commentService.comments(articleId);

// 결과 응답 (성공한 경우만 있다고 가정)

return ResponseEntity.status(HttpStatus.OK).body(dtos);

}

// 댓글 생성

@PostMapping("/api/articles/{articleId}/comments")

public ResponseEntity<CommentDto> create(@PathVariable Long articleId,

@RequestBody CommentDto dto) {

// 서비스에게 위임

CommentDto createdDto = commentService.create(articleId,dto);

// 결과 응답( create 메소드안에서 에러를 발생시켜서 실패한경우에 응답코드를 자동으로 반환할 수 있도록 작성할 예정)

return ResponseEntity.status(HttpStatus.OK).body(createdDto);

}

// 댓글 수정

@PatchMapping("/api/comments/{id}")

public ResponseEntity<CommentDto> update(@PathVariable Long id,

@RequestBody CommentDto dto) {

// 서비스에게 위임

CommentDto updatedDto = commentService.update(id,dto);

// 결과 응답

return ResponseEntity.status(HttpStatus.OK).body(updatedDto);

}

// 댓글 삭제

@DeleteMapping("/api/comments/{id}")

public ResponseEntity<CommentDto> delete(@PathVariable Long id) {

// 서비스에게 위임

CommentDto deletedDto = commentService.delete(id);

// 결과 응답

return ResponseEntity.status(HttpStatus.OK).body(deletedDto);

}

}

**CommentService**

package com.example.firstproject.service;

@Service

public class CommentService {

@Autowired

private CommentRepository commentRepository;

@Autowired

private ArticleRepository articleRepository;

// 댓글이 달린 게시글도 가져와야하기 때문에 ArticleRepository도 함께 가져오기

public List<CommentDto> comments(Long articleId) {

// 조회: 댓글 목록 조회(게시글 아이디를 통해 해당 게시글의 댓글 목록 조회)

// List<Comment> comments = commentRepository.findByArticleId(articleId);

// 변환: 엔티티 -> DTO

// CommentApiController에서 List<Comment> -> List<CommentDto>로 반환하기로 했기때문에 엔티티 -> DTO로 변환

// List<CommentDto> dtos = new ArrayList<CommentDto>();

// 비어있는 dtos에다가 댓글들을 변환해서 add하기

// for(int i = 0; i < comments.size(); i++) {

// // comments 값을 하나하나 꺼내서 넣기

// Comment c = comments.get(i);

// CommentDto dto = CommentDto.createCommentDto(c); // Dto로 변환

// dtos.add(dto);

// }

// 반환

return commentRepository.findByArticleId(articleId) // commentRepository에 목록조회

.stream() // stream으로 변경

.map(comment -> CommentDto.createCommentDto(comment)) //createCommentDto를 통해 comment를 하나하나전달하여 DTO로 변환

.collect(Collectors.toList()); // map이 반환하는 값이 stream<Object>이기 때문에

}

@Transactional

public CommentDto create(Long articleId, CommentDto dto) {

// 게시글 조회 및 예외 발생

// .orElseThrow(() -> new IllegalArgumentException()) article이 없다면 예외발생시켜서 다음 코드가 실행되지 않는다.

Article article = articleRepository.findById(articleId)

.orElseThrow(() -> new IllegalArgumentException("댓글 생성 실패!! 대상 게시글이 없습니다."));

// 댓글 엔티티 생성

Comment comment = Comment.createComment(dto, article);

// 댓글 엔티티를 DB로 저장

Comment created = commentRepository.save(comment);

// DTO로 변환하여 반환

return CommentDto.createCommentDto(created);

}

@Transactional

public CommentDto update(Long id, CommentDto dto) {

// 댓글 조회 및 예외 발생

Comment target = commentRepository.findById(id)

.orElseThrow(() -> new IllegalArgumentException("댓글 수정 실패! 대상 댓글이 없습니다."));

// 댓글 수정

target.patch(dto);

// 댓글 DB로 갱신

Comment updated = commentRepository.save(target);

// 댓글 엔티티를 DTO로 변환 및 반환

return CommentDto.createCommentDto(updated);

}

@Transactional

public CommentDto delete(Long id) {

// 댓글 조회 및 예외 발생

Comment target = commentRepository.findById(id)

.orElseThrow(() -> new IllegalArgumentException("댓글 삭제 실패! 대상 댓글이 없습니다."));

// 댓글 DB에서 삭제

commentRepository.delete(target);

// 삭제 댓글을 DTO로 반환

return CommentDto.createCommentDto(target);

}

}

**CommentDto**

package com.example.firstproject.dto;

@AllArgsConstructor

@NoArgsConstructor

@Getter

@ToString

public class CommentDto {

private Long id;

@JsonProperty("article\_id")

private Long articleId;

private String nickname;

private String body;

public static CommentDto createCommentDto(Comment comment) {

return new CommentDto(

comment.getId(),

comment.getArticle().getId(), // Article을 가져오고 거기서 Id만 필요하기 때문에 다시 .getId()

comment.getNickname(),

comment.getBody()

);

}

}

**Comment**

package com.example.firstproject.entity;

@Entity

@Getter

@ToString

@AllArgsConstructor

@NoArgsConstructor

public class Comment {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne // 해당 댓글 엔티티 여러개가 하나의 Article에 연관된다!!

@JoinColumn(name = "article\_id") // article 테이블의 id를 가져올떄 name

private Article article; // 댓글의 부모 게시글

@Column

private String nickname;

@Column

private String body;

public static Comment createComment(CommentDto dto, Article article) {

// 예외 발생

if(dto.getId() != null) // 받아온 데이터에 id가 있다면

throw new IllegalArgumentException("댓글 생성 실패! 댓글의 id가 없어야 합니다.");

if(dto.getArticleId() != article.getId()) // 요청url의 id(articleId)와 요청데이터에 article\_id가 다르면

throw new IllegalArgumentException("댓글 생성 실패! 게시글의 id가 잘못되었습니다.");

// 엔티티 생성 및 반환

return new Comment(

dto.getId(),

article,

dto.getNickname(),

dto.getBody()

);

}

public void patch(CommentDto dto) {

// 예외 발생

if(this.id != dto.getId()) { // 요청URL id와 댓글의 id가 다를때

throw new IllegalArgumentException("댓글 수정 실패! 잘못된 id가 입력되었습니다.");

}

// 객체를 갱신

if(dto.getNickname() != null) {

this.nickname = dto.getNickname();

}

if(dto.getBody() != null) {

this.body = dto.getBody();

}

}

}