15 장 완성 코드

api/CommentApiController.java

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

import com.example.firstproject.dto.CommentDto;
import com.example.firstproject.service.CommentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class CommentApiController {
    @Autowired
    private CommentService commentService;

    // 1. 댓글 조회
    @GetMapping("/api/articles/{articleId}/comments")
    public   ResponseEntity<List<CommentDto>>   comments(@PathVariable   Long
articleId) {
        // 서비스에게 작업 위임
        List<CommentDto> dtos = commentService.comments(articleId);

        // 결과 응답
        return ResponseEntity.status(HttpStatus.OK).body(dtos);
    }

    // 2. 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
        return ResponseEntity.status(HttpStatus.OK).body(createdDto);
    }

    // 3. 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);
    }

    // 4. 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);
    }
}
```

service/CommentService.java

```
package com.example.firstproject.service;

import com.example.firstproject.dto.CommentDto;
import com.example.firstproject.entity.Article;
import com.example.firstproject.entity.Comment;
import com.example.firstproject.repository.ArticleRepository;
import com.example.firstproject.repository.CommentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class CommentService {
    @Autowired
    private CommentRepository commentRepository;

    @Autowired
    private ArticleRepository articleRepository;

    public List<CommentDto> comments(Long articleId) {
        /*// 1. 댓글 조회
        List<Comment> comments = commentRepository.findByArticleId(articleId);

        // 2. 엔티티 -> DTO 변환
        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);
        }*/

        // 3. 결과 반환
        return commentRepository.findByArticleId(articleId)
                .stream()
                .map(comment -> CommentDto.createCommentDto(comment))
                .collect(Collectors.toList());
    }

    @Transactional
    public CommentDto create(Long articleId, CommentDto dto) {
        // 1. 게시글 조회 및 예외 발생
        Article article = articleRepository.findById(articleId)
                .orElseThrow(() -> new IllegalArgumentException("댓글 생성 실패!
" +
                        "대상 게시글이 없습니다."));

        // 2. 댓글 엔티티 생성
        Comment comment = Comment.createComment(dto, article);

        // 3. 댓글 엔티티를 DB로 저장
        Comment created = commentRepository.save(comment);

        // 4. DTO로 변환하여 반환
        return CommentDto.createCommentDto(created);
    }

    public CommentDto update(Long id, CommentDto dto) {
        // 1. 댓글 조회 및 예외 발생
        Comment target = commentRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("댓글 수정 실패!"
+
                        "대상 댓글이 없습니다."));

        // 2. 댓글 수정
        target.patch(dto);

        // 3. DB로 갱신
        Comment updated = commentRepository.save(target);

        // 4. 댓글 엔티티를 DTO로 변환 및 반환
        return CommentDto.createCommentDto(updated);
    }
    @Transactional
    public CommentDto delete(Long id) {
        // 1. 댓글 조회 및 예외 발생
```

```
        Comment target = commentRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("댓글 삭제 실패! "
+
                        "대상이 없습니다."));

        // 2. 댓글 삭제
        commentRepository.delete(target);

        // 3. 삭제 댓글을 DTO로 변환 및 반환
        return CommentDto.createCommentDto(target);
    }
}
```

dto/CommentDto.java

```
package com.example.firstproject.dto;

import com.example.firstproject.entity.Comment;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.ToString;

@AllArgsConstructor // 모든 필드를 파라미터로 갖는 생성자 자동 생성
@NoArgsConstructor // 파라미터가 아예 없는 기본 생성자 자동 생성
@Getter // 각 필드 값을 조회할 수 있는 getter 메서드 자동 생성
@ToString // 모든 필드를 출력할 수 있는 toString 메서드 자동 생성
public class CommentDto {
    private Long id; // 댓글의 id
    private Long articleId; // 댓글의 부모 id
    private String nickname; // 댓글 작성자
    private String body; // 댓글 본문

    public static CommentDto createCommentDto(Comment comment) {
        return new CommentDto(
                comment.getId(), // 댓글 엔티티의 id
                comment.getArticle().getId(), // 댓글 엔티티가 속한 부모 게시
글의 id
                comment.getNickname(), // 댓글 엔티티의 nickname
                comment.getBody() // 댓글 엔티티의 body
        );
    }
}
```

entity/Comment.java

```
package com.example.firstproject.entity;

import com.example.firstproject.dto.CommentDto;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.Objects;

@Entity // 해당 클래스가 엔티티임을 선언, 클래스 필드를 바탕으로 DB에 테이블 생성
@Getter // 각 필드 값을 조회할 수 있는 getter 메서드 자동 생성
@ToString // 모든 필드를 출력할 수 있는 toString 메서드 자동 생성
@AllArgsConstructor // 모든 필드를 매개변수로 갖는 생성자 자동 생성
@NoArgsConstructor // 매겨변수가 아예 없는 기본 생성자 자동 생성

public class Comment {
    @Id // 대표키 지정
    @GeneratedValue(strategy= GenerationType.IDENTITY) // DB가 자동으로 1씩 증가
    private Long id; // 대표키

    @ManyToOne // 이 엔티티(Comment)와 부모 엔티티(Article)를 다대일 관계로 설정
    @JoinColumn(name="article_id") // 외래키 생성, Article 엔티티의 기본키(id)와
매핑
    private Article article; // 해당 댓글의 부모 게시글

    @Column // 해당 필드를 테이블의 속성으로 매핑
    private String nickname; // 댓글을 단 사람

    @Column // 해당 필드를 테이블의 속성으로 매핑
    private String body; // 댓글 본문

    public static Comment createComment(CommentDto dto, Article article) {
        // 예외 발생
        if (dto.getId() != null)
            throw new IllegalArgumentException("댓글 생성 실패! 댓글의 id가 없어야 합니
다.");
        if (dto.getArticleId() != article.getId())
          throw new IllegalArgumentException("댓글 생성 실패! 게시글의 id가 잘못됐습니다.");

        // 엔티티 생성 및 반환
        return new Comment(
                dto.getId(), // 댓글 아이디
                article, // 부모 게시글
                dto.getNickname(), // 댓글 닉네임
                dto.getBody() // 댓글 본문
```

```
        );
    }

    public void patch(CommentDto dto) {
        // 예외 발생
        if (this.id != dto.getId())
          throw new IllegalArgumentException("댓글 수정 실패! 잘못된 id가 입력됐습니다.");

        // 객체 갱신
        if (dto.getNickname() != null) // 수정할 닉네임 데이터가 있다면
            this.nickname = dto.getNickname(); // 내용을 반영

        if (dto.getBody() != null) // 수정할 본문 데이터가 있다면
            this.body = dto.getBody(); // 내용을 반영
    }
}
```