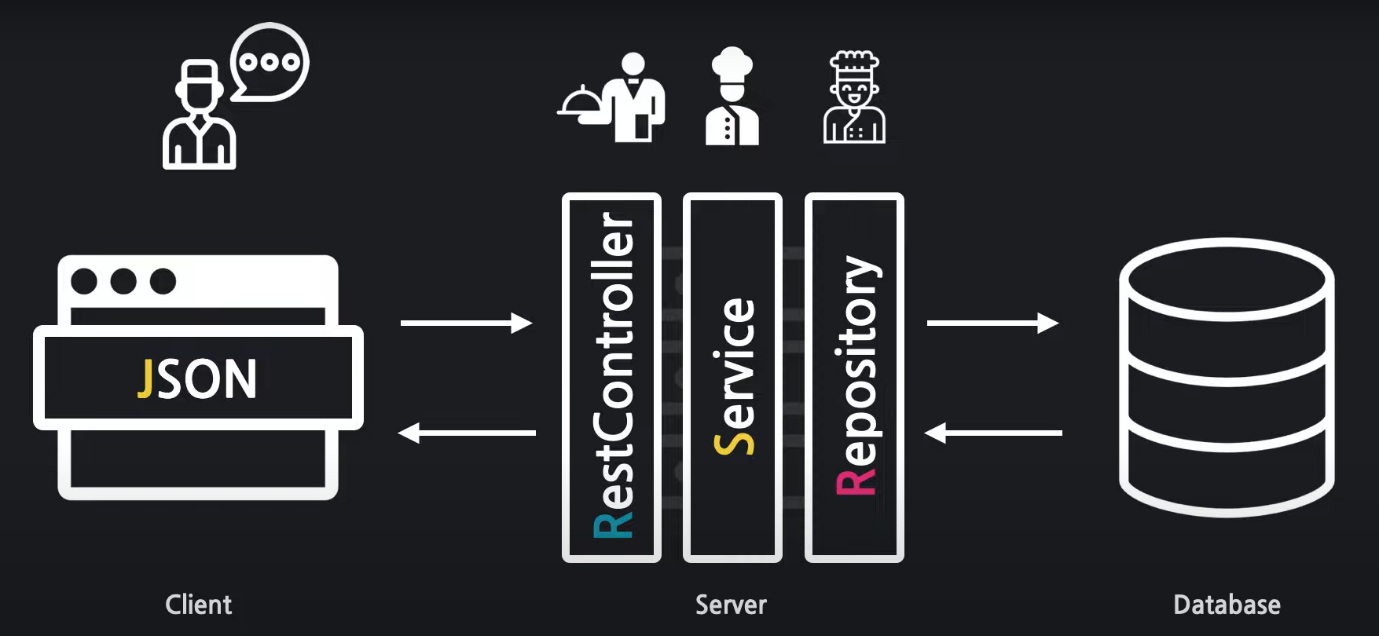
서비스와 트랜잭션, 그리고 롤백

**서비스**

서비스는 Controller(웨이터)와 Repository(보조솊)사이에 셰프역할을 한다.  


**트랜잭션**

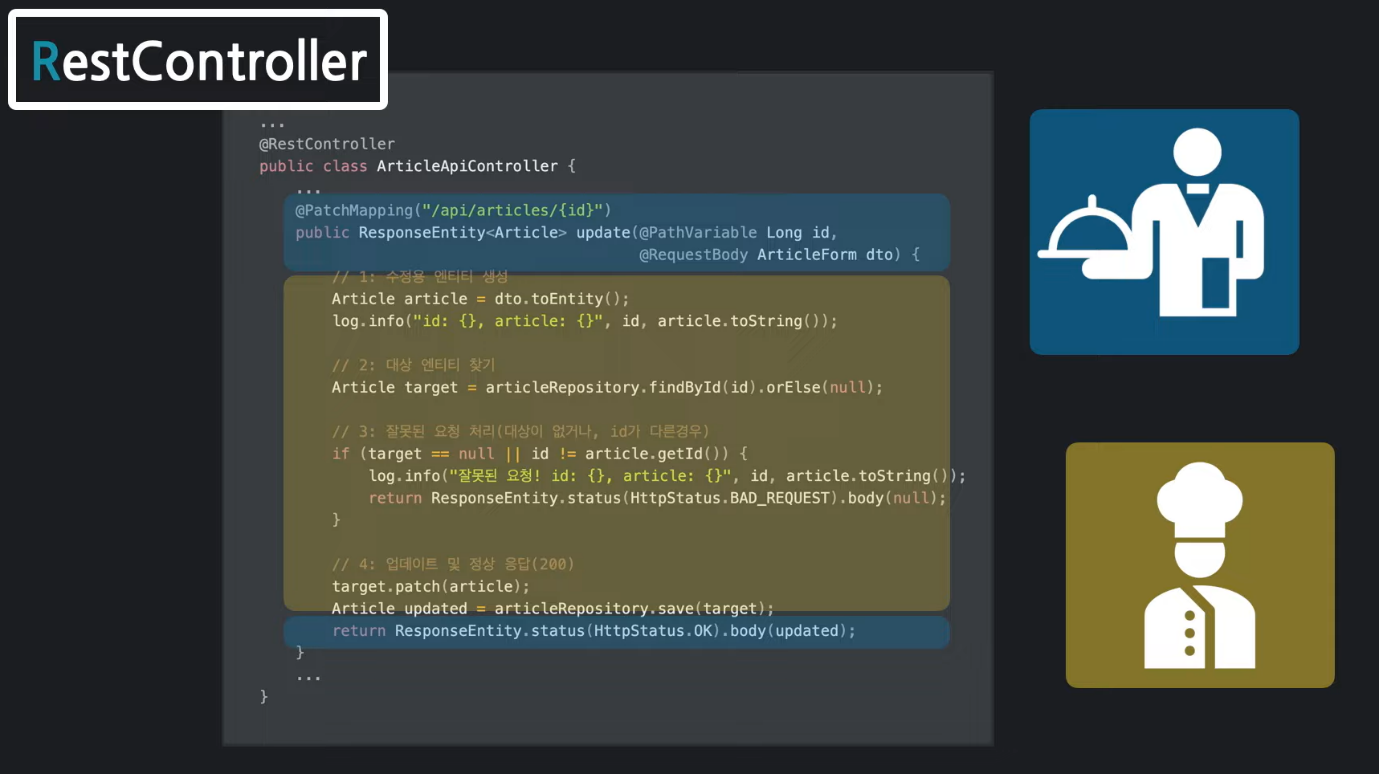
Transaction은 데이터베이스 연산을 수행하는 데 있어서 일련의 작업들을 하나의 원자적인 단위로 처리하기 위한 기능입니다.

예를 들어, 동시에 여러 개의 데이터베이스 연산을 수행하는 경우, 각 연산이 독립적으로 실행될 때 발생할 수 있는 문제들이 발생할 수 있습니다. 이런 문제들을 해결하기 위해서는, 하나의 작업 단위로 묶어서 처리하는 것이 필요합니다. 이때 Transaction을 사용하면 하나의 논리적인 작업 단위로 묶어서 처리할 수 있습니다.

Spring에서는 Transaction을 지원하기 위한 여러 가지 방법들이 제공됩니다. 예를 들어, @Transactional 어노테이션을 이용하여 트랜잭션을 적용할 수 있습니다. 이 어노테이션을 메소드나 클래스에 붙여주면, 해당 메소드나 클래스에서 실행되는 모든 데이터베이스 연산은 하나의 트랜잭션으로 처리됩니다.

**롤백**

롤백(Rollback)은 데이터베이스 트랜잭션에서 실행된 일련의 작업들을 모두 취소하는 것을 말합니다. 즉, 트랜잭션 실행 중에 문제가 발생하여 트랜잭션을 중단하고 이전 상태로 되돌리는 작업을 의미합니다.

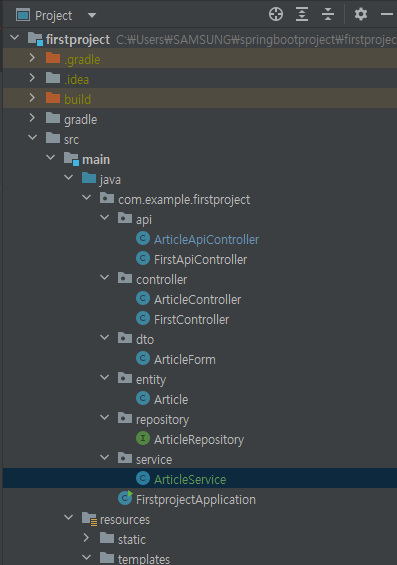
  
현재는 클라이언트의 요청과 응답을 처리함과 동시에  
repository에게 데이터를 가져오도록 명령하는  
웨이터와 세프의 기능을 동시에 진행하고 있었다.

이제 서비스를 두어서 역할을 분담해보자.

**실습**

**서비스 생성**

**Repository -> Service 변경**  
현재 ARticleApiController는 Repository와 협업해서 진행하고 있는데 Service를 추가해서 분업화 해보자.



ArticleService 생성 후 Repository 연결

@Service // 서비스 선언(서비스 객체를 스프링부트에 생성하는 코드)

public class ArticleService {

@Autowired

private ArticleRepository articleRepository;

}

ArticleApiController에 ArticleService를 연결

@Slf4j

@RestController

public class ArticleApiController {

@Autowired // DI, 생성 객체를 가져와 연결

private ArticleService articleService;

...

}

**리팩토링**

**Article 목록 조회**

ArticleApiController  
Repository가 아닌 service를 이용.

// GET

// 전체게시글 불러오기

@GetMapping("/api/articles")

public List<Article> index() {

return articleService.index();

}

ArticleService에서  
Repository이용해서 데이터 불러온 후 return.

public List<Article> index() {

return articleRepository.findAll();

}

**Article 단건(상세)조회**

// 게시글 하나 불러오기

@GetMapping("/api/articles/{id}")

public Article show(@PathVariable Long id) {

return articleService.show(id);

}

public Article show(Long id) {

return articleRepository.findById(id).orElse(null);

}

**Article 생성**

// POST

@PostMapping("/api/articles")

public ResponseEntity<Article> create(@RequestBody ArticleForm dto) {

Article created = articleService.create(dto);

// 잘 생성된경우 GOOD : 안됐을때 BAD

return (created != null) ? GOOD : BAD ;

}

// POST

@PostMapping("/api/articles")

public ResponseEntity<Article> create(@RequestBody ArticleForm dto) {

Article created = articleService.create(dto);

// 잘 생성된경우 GOOD : 안됐을때 BAD

return (created != null) ?

ResponseEntity.status(HttpStatus.OK).body(created) :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build() ;

}

public Article create(ArticleForm dto) {

Article article = dto.toEntity();

return articleRepository.save(article);

}

create를 할때 id를 넣어서 보내면  
id값의 내용이 수정되기 때문에 수정되지 않도록 코드를 추가하자.

public Article create(ArticleForm dto) {

Article article = dto.toEntity();

// 생성할때 id를 넣었으면 null을 리턴

if(article.getId() != null) {

return null;

}

return articleRepository.save(article);

}

**Article 수정**

// PATCH

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

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

@RequestBody ArticleForm dto) {

Article updated = articleService.update(id, dto);

return (updated != null) ?

ResponseEntity.status(HttpStatus.OK).body(updated) :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

public Article update(Long id, ArticleForm dto) {

// dto를 Entity로 변경

Article article = dto.toEntity();

// target 찾기

Article target = articleRepository.findById(id).orElse(null);

// 잘못된 요청이면 null

if(target == null || id != article.getId()) {

return null;

}

// 업데이트 후 updated 리턴

target.patch(article);

Article updated = articleRepository.save(target);

return updated;

}

**Article 삭제**

// DELETE

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

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

Article deleted = articleService.delete(id);

return (deleted != null) ?

ResponseEntity.status(HttpStatus.NO\_CONTENT).build() :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

public Article delete(Long id) {

Article target = articleRepository.findById(id).orElse(null);

if(target == null) {

return null;

}

articleRepository.delete(target);

return target;

}

**트랜잭션 맛보기**

트랜잭션(반드시 성공해야할 일련의 과정) -> 성공못하면 롤백!!

**묶음 Article 생성**

@PostMapping("/api/transaction-test")

public ResponseEntity<List<Article>> transactionTest(@RequestBody List<ArticleForm> dtos) {

List<Article> createsList = articleService.createArticles(dtos);

return (createsList != null) ?

ResponseEntity.status(HttpStatus.OK).body(createsList) :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

id를 -1해서 강제로 예외를 발생시켜보자.

public List<Article> createArticles(List<ArticleForm> dtos) {

// dto 묶음을 entity 묶음으로 변환

List<Article> articleList = dtos.stream().map(dto -> dto.toEntity()).collect(Collectors.toList());

// 위코드 for문으로 작성

// List<Article> articleList = new ArrayList<>();

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

// ArticleForm dto = dtos.get(i);

// Article entity = dto.toEntity();

// articleList.add(entity);

// }

// entity 묶음을 DB로 저장

articleList.stream().forEach(article -> articleRepository.save(article));

// 위코드 for문으로 작성

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

// Article article = articleList.get(i);

// articleRepository.save(article);

// }

// 강제 예외 발생 시키기

articleRepository.findById(-1L).orElseThrow(

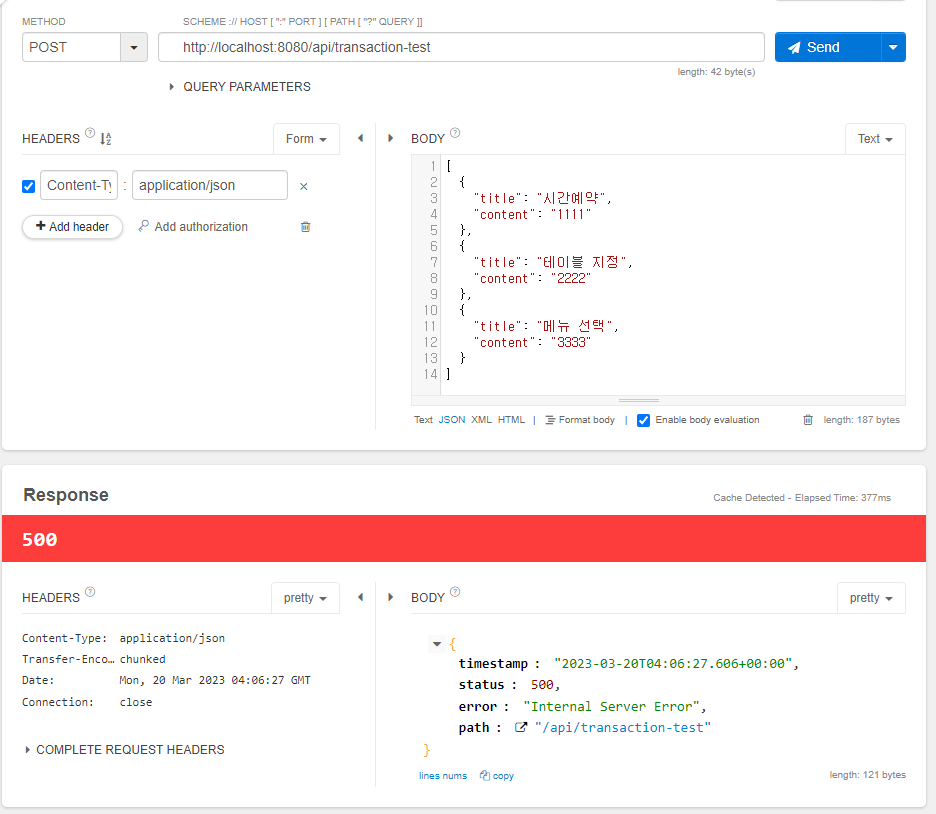
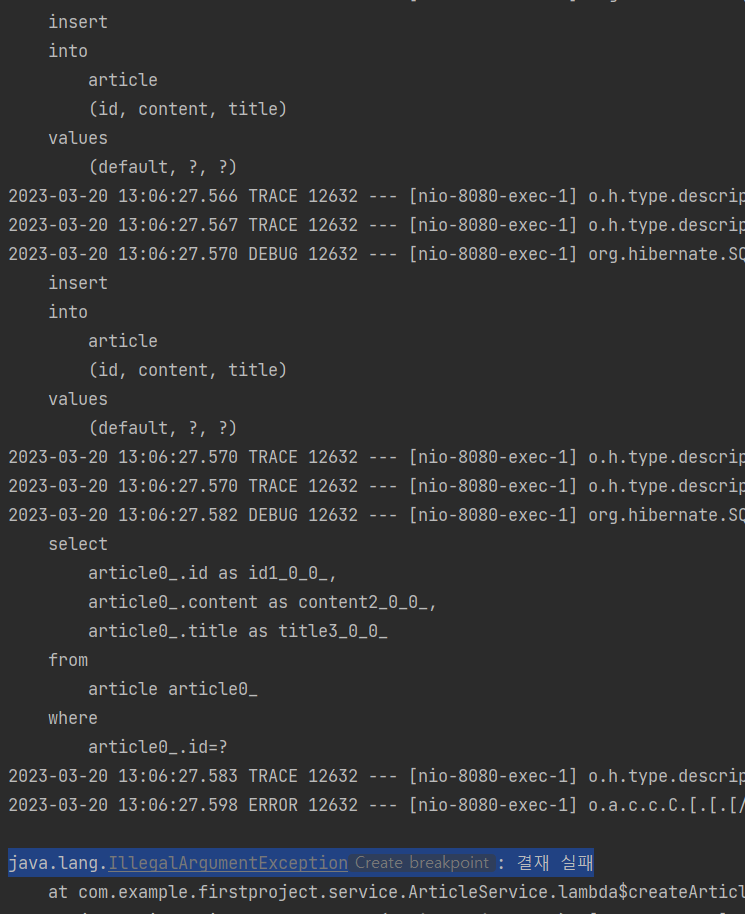
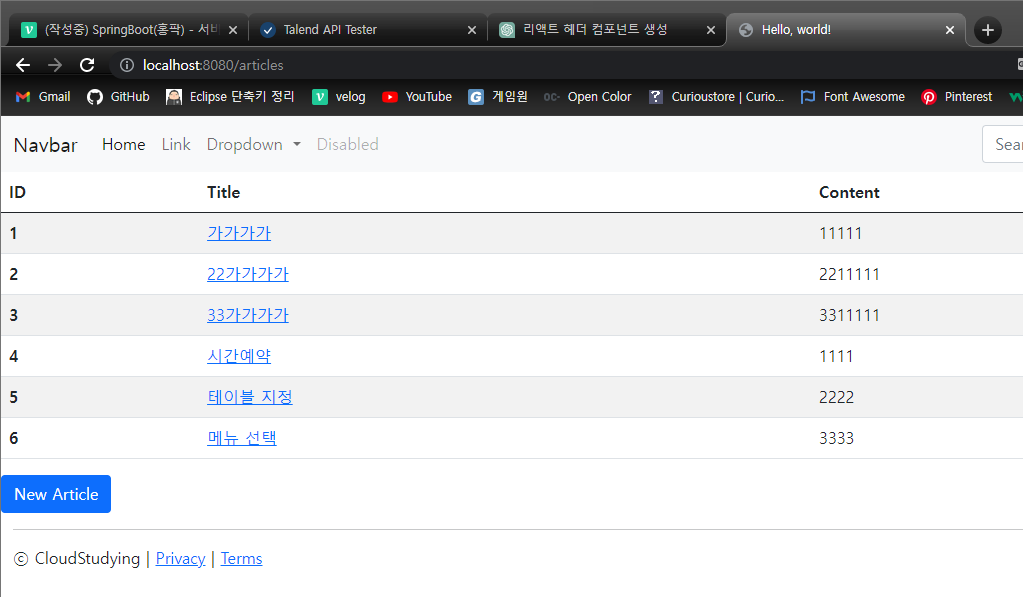
() -> new IllegalArgumentException("결재 실패")

);

// 결과값 반환

return articleList;

}

서비스 코드작성 후 Talend로 요청 보내면 500 에러가난다.  
  
콘솔을 보면 IllegalArgumentException이 발생하고  
그 위에 insert가 3번 된걸 확인할 수 있다.  
  
데이터가 DB에 저장됐다.  
**예외가 발생했을때 DB에 저장되면 안되기 때문에  
트랜잭션 처리가 필요하다.**  


✨ 트랜잭션은 Service가 관리한다. ✨  
트랜잭션 처리할 메소드에 @Transactional 추가.  
이렇게 되면 해당 메소드가 실패할 경우 롤백을 진행한다.

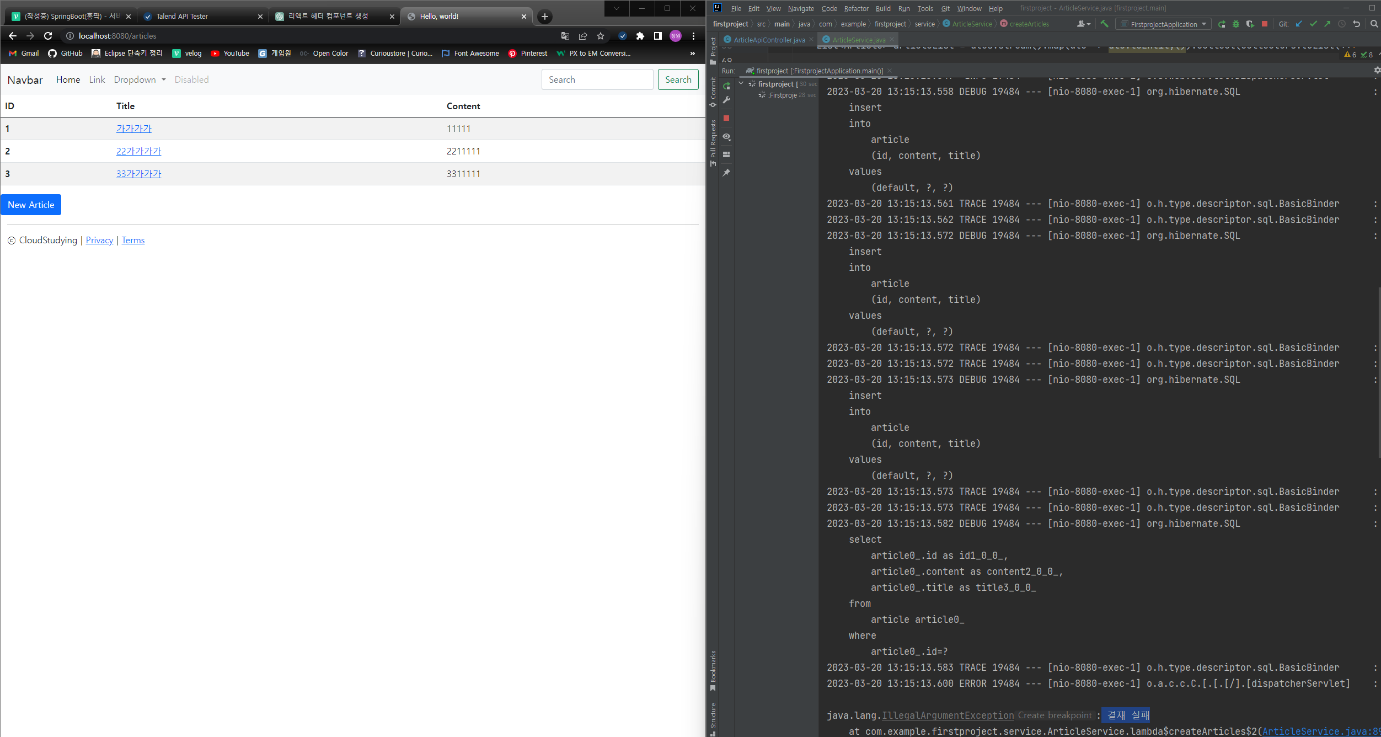
@Transactional // 해당 메소드를 트랜잭션으로 묶는다!

public List<Article> createArticles(List<ArticleForm> dtos) {

List<Article> articleList = dtos.stream().map(dto -> dto.toEntity()).collect(Collectors.toList());

...

}

다시 요청을 보내보면  
위와 같이 500에러, 콘솔에 IllegalArgumentException 발생, 3번의 insert가 진행되지만 DB에는 저장되지 않는다.  
실패하면 롤백.  


**기존 ArticleApiController 코드**

@Slf4j

@RestController // RestAPI용 컨트롤러, 데이터(JSON)을 반환

public class ArticleApiController {

@Autowired //ArticleRepository를 스프링 부트에서 땡겨와야하기 때문에 @Autowired 사용 (DI)

private ArticleRepository articleRepository;

// GET

// 전체게시글 불러오기

@GetMapping("/api/articles")

public List<Article> index() {

return articleRepository.findAll();

}

// 게시글 하나 불러오기

@GetMapping("/api/articles/{id}")

public Article index(@PathVariable Long id) {

return articleRepository.findById(id).orElse(null);

}

// POST

@PostMapping("/api/articles")

public Article index(@RequestBody ArticleForm dto) {

Article article= dto.toEntity();

return articleRepository.save(article);

}

// PATCH

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

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

@RequestBody ArticleForm dto) {

// 1: 수정용 엔티티 생성

Article article = dto.toEntity();

log.info("id: {}, article: {}", id, article.toString());

// 순서대로 {} 안에 들어간다.

// 2: 대상 엔티티를 조회

Article target = articleRepository.findById(id).orElse(null);

// 3: 잘못된 요청 처리(대상이 없거나, id가 다른 경우)

if(target == null || id != article.getId()) {

// 400, 잘못된 요청 응답

log.info("잘못된 요청! id: {}, article: {}", id, article.toString());

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(null);

// HttpStatus.BAD\_REQUEST = 400

}

// 4: 업데이트 및 정상 응답(200)

target.patch(article); // 기존 데이터에 새로운 부분만 추가한다.

Article updated = articleRepository.save(target);

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

// status(HttpStatus.OK = 200

}

// DELETE

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

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

// 대상 찾기

Article target = articleRepository.findById(id).orElse(null);

// 잘못된 요청 처리

if(target == null) {

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(null);

}

// 대상 삭제

articleRepository.delete(target);

// 데이터 반환

// return ResponseEntity.status(HttpStatus.OK).body(null); 아래와 같음

return ResponseEntity.status(HttpStatus.OK).build();

}

}

**service로 나눈 ArticleApiController코드**

@Slf4j

@RestController // RestAPI용 컨트롤러, 데이터(JSON)을 반환

public class ArticleApiController {

@Autowired // DI, 생성 객체를 가져와 연결

private ArticleService articleService;

// GET

// 전체게시글 불러오기

@GetMapping("/api/articles")

public List<Article> index() {

return articleService.index();

}

// 게시글 하나 불러오기

@GetMapping("/api/articles/{id}")

public Article show(@PathVariable Long id) {

return articleService.show(id);

}

// POST

@PostMapping("/api/articles")

public ResponseEntity<Article> create(@RequestBody ArticleForm dto) {

Article created = articleService.create(dto);

// 잘 생성된경우 GOOD : 안됐을때 BAD

return (created != null) ? ResponseEntity.status(HttpStatus.OK).body(created) : ResponseEntity.status(HttpStatus.BAD\_REQUEST).build() ;

}

// PATCH

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

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

@RequestBody ArticleForm dto) {

Article updated = articleService.update(id, dto);

return (updated != null) ?

ResponseEntity.status(HttpStatus.OK).body(updated) :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

// DELETE

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

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

Article deleted = articleService.delete(id);

return (deleted != null) ?

ResponseEntity.status(HttpStatus.NO\_CONTENT).build() :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

}

// 트랜잭션(반드시 성공해야할 일련의 과정) -> 성공못하면 롤백!!

@PostMapping("/api/transaction-test")

public ResponseEntity<List<Article>> transactionTest(@RequestBody List<ArticleForm> dtos) {

List<Article> createsList = articleService.createArticles(dtos);

return (createsList != null) ?

ResponseEntity.status(HttpStatus.OK).body(createsList) :

ResponseEntity.status(HttpStatus.BAD\_REQUEST).build();

}

**ArticleService 전체코드**

package com.example.firstproject.service;

import com.example.firstproject.dto.ArticleForm;

import com.example.firstproject.entity.Article;

import com.example.firstproject.repository.ArticleRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

@Service // 서비스 선언(서비스 객체를 스프링부트에 생성하는 코드)

public class ArticleService {

@Autowired

private ArticleRepository articleRepository;

public List<Article> index() {

return articleRepository.findAll();

}

public Article show(Long id) {

return articleRepository.findById(id).orElse(null);

}

public Article create(ArticleForm dto) {

Article article = dto.toEntity();

// 생성할때 id를 넣었으면 null을 리턴

if(article.getId() != null) {

return null;

}

return articleRepository.save(article);

}

public Article update(Long id, ArticleForm dto) {

// dto를 Entity로 변경

Article article = dto.toEntity();

// target 찾기

Article target = articleRepository.findById(id).orElse(null);

// 잘못된 요청이면 null

if(target == null || id != article.getId()) {

return null;

}

// 업데이트 후 updated 리턴

target.patch(article);

Article updated = articleRepository.save(target);

return updated;

}

public Article delete(Long id) {

Article target = articleRepository.findById(id).orElse(null);

if(target == null) {

return null;

}

articleRepository.delete(target);

return target;

}

@Transactional // 해당 메소드를 트랜잭션으로 묶는다!

public List<Article> createArticles(List<ArticleForm> dtos) {

// dto 묶음을 entity 묶음으로 변환

List<Article> articleList = dtos.stream().map(dto -> dto.toEntity()).collect(Collectors.toList());

// 위코드 for문으로 작성

// List<Article> articleList = new ArrayList<>();

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

// ArticleForm dto = dtos.get(i);

// Article entity = dto.toEntity();

// articleList.add(entity);

// }

// entity 묶음을 DB로 저장

articleList.stream().forEach(article -> articleRepository.save(article));

// 위코드 for문으로 작성

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

// Article article = articleList.get(i);

// articleRepository.save(article);

// }

// 강제 예외 발생 시키기

articleRepository.findById(-1L).orElseThrow(

() -> new IllegalArgumentException("결재 실패")

);

// 결과값 반환

return articleList;

}

}