

# 와플 스튜디오 세미나

스프링 부트 세미나 2

# 과제

## ■ 리뷰

- 가독성이 매우 중요
- 주석을 잘 달자
- PR 리뷰 프로세스

## ■ 과제 검사

- 리뷰는 말했던 것처럼 조에서 한 번
- 완료 여부는 확인 예정

# 과제 진행상황 조사

- 다 했거나 오늘 안에 끝낼 수 있다.
- 아직 못했다. 하루 정도 시간이 더 필요하다.
- 손도 거의 못 댔다. 너무 어렵다.


# 과제

Conversation 20

Commits 12

Checks 0

Files changed 34



sangggggg commented on 16 May • edited ▾

Member 😊 ⋮

## 구현 세부

- middleware
  - api-key-validator middleware 추가
  - user guard & attach middleware 추가
  - global error handler filter 추가
  - morgan request logger middleware 추가
- service
  - auth service 추가
  - user service 추가

## 다음에 할 일

- user, auth 라우팅
- fb, apple auth 추가

# 과제

The screenshot shows a GitHub pull request interface for the repository 'wafflestudio / snutt-ev'. The pull request is titled 'SNUTT-151 snutt-ev 서버 배포하기 #2' and is in the 'Open' state. It shows that 'davin111' wants to merge 4 commits into the 'develop' branch from the 'feature/SNUTT-151' branch. The interface includes navigation tabs for Code, Issues, Pull requests (1), Actions, Projects, Wiki, Security, and Insights. Below the title, there are tabs for Conversation (3), Commits (4), Checks (1), and Files changed (6). The 'Files changed' tab is active, showing a diff for the file '.github/workflows/deploy.yml'. The diff shows a new workflow named 'Deploy' triggered on push to the 'develop' or 'feature/SNUTT-151' branches.

wafflestudio / snutt-ev Public

<> Code • Issues 🔗 Pull requests 1 ▶ Actions 📁 Projects 📖 Wiki ⚠ Security 📈 Insights

## SNUTT-151 snutt-ev 서버 배포하기 #2

🔗 Open davin111 wants to merge 4 commits into develop from feature/SNUTT-151 📄

💬 Conversation 3 🔗 Commits 4 📄 Checks 1 📄 Files changed 6

Changes from all commits ▾ File filter ▾ Conversations ▾ Jump to ▾ ⚙ ▾

69 .github/workflows/deploy.yml 📄

```
... @@ -0,0 +1,69 @@
1 + name: Deploy
2 +
3 + on:
4 +   push:
5 +     branches: [ develop, feature/SNUTT-151 ]
```

# 과제

```
45 + - name: Delete untagged images in ECR
46 +   run: |
47 +     UNTAGGED_IMAGES=$( aws ecr list-images --repository-name $ECR_REPOSITORY --filter "tagStatus=UNTAGGED" --query 'imageIds[*]' --output json )
48 +     aws ecr batch-delete-image --repository-name $ECR_REPOSITORY --image-ids "$UNTAGGED_IMAGES" || true
49 +
50 + - name: Deploy to ElasticBeanstalk
51 +   run: |
52 +     aws elasticbeanstalk create-application-version \
53 +       --application-name snutt-ev-server \
54 +       --version-label $BUILD_NUMBER \
55 +       --description $BUILD_NUMBER \
56 +       --source-bundle S3Bucket=$S3_BUCKET_NAME,S3Key='deploy.zip'
57 +     aws elasticbeanstalk update-environment \
58 +       --environment-name snutt-ev-server-production \
59 +       --version-label $BUILD_NUMBER
```

Commenting on lines +46 to +59

Write

Preview

📎 H B I ☰ <> 🔗 ☰ ☷ ☑ @ ↗ ↶

리뷰는 이렇게 달면 됩니다!

Attach files by dragging & dropping, selecting or pasting them.



Cancel

Add single comment

Start a review

# AOP와 Proxy 패턴

```
@Component
@Aspect
public class PerfAspect {

    @Around("bean(simpleEventService)")
    public Object logPerf(ProceedingJoinPoint pjp) throws Throwable{
        long begin = System.currentTimeMillis();

        Object retVal = pjp.proceed(); // 메서드 호출 자체를 감쌌다
        System.out.println(System.currentTimeMillis() - begin);

        return retVal;
    }
}
```

# Error handling

## ■ AOP의 일종

- *@RestControllerAdvice* 혹은 *@ControllerAdvice* 를 통해 모든 Controller 메소드를 감싼다.
- *@ExceptionHandler* 를 단 함수를 통해 특정 Exception 에 대한 동작을 설정할 수 있다.

## ■ 해당 함수의 리턴값이 http response로 전달된다.

```
@RestControllerAdvice
class SurveyControllerAdvice() {
    private val logger = LoggerFactory.getLogger(this.javaClass.name)
    @ExceptionHandler(value = [DataNotFoundException::class])
    fun notfound(e: WaffleException) =
        ResponseEntity(ErrorResponse(e.errorType.code, e.errorType.name, e.detail), HttpStatus.NOT_FOUND)

    @ExceptionHandler(value = [InvalidRequestException::class])
    fun badRequest(e: WaffleException) =
        ResponseEntity(ErrorResponse(e.errorType.code, e.errorType.name, e.detail), HttpStatus.BAD_REQUEST)

    @ExceptionHandler(value = [NotAllowedException::class])
    fun notAllowed(e: WaffleException) =
        ResponseEntity(ErrorResponse(e.errorType.code, e.errorType.name, e.detail), HttpStatus.FORBIDDEN)

    @ExceptionHandler(value = [RequestConflictException::class])
    fun conflict(e: WaffleException) =
        ResponseEntity(ErrorResponse(e.errorType.code, e.errorType.name, e.detail), HttpStatus.CONFLICT)

    @ExceptionHandler(value = [Exception::class])
    fun internalError(e: Exception) {
        logger.debug("{} ", e.toString())
        ResponseEntity(ErrorResponse(-1, errorMessage: "서버 오류", detail: ""), HttpStatus.INTERNAL_SERVER_ERROR)
    }
}
```

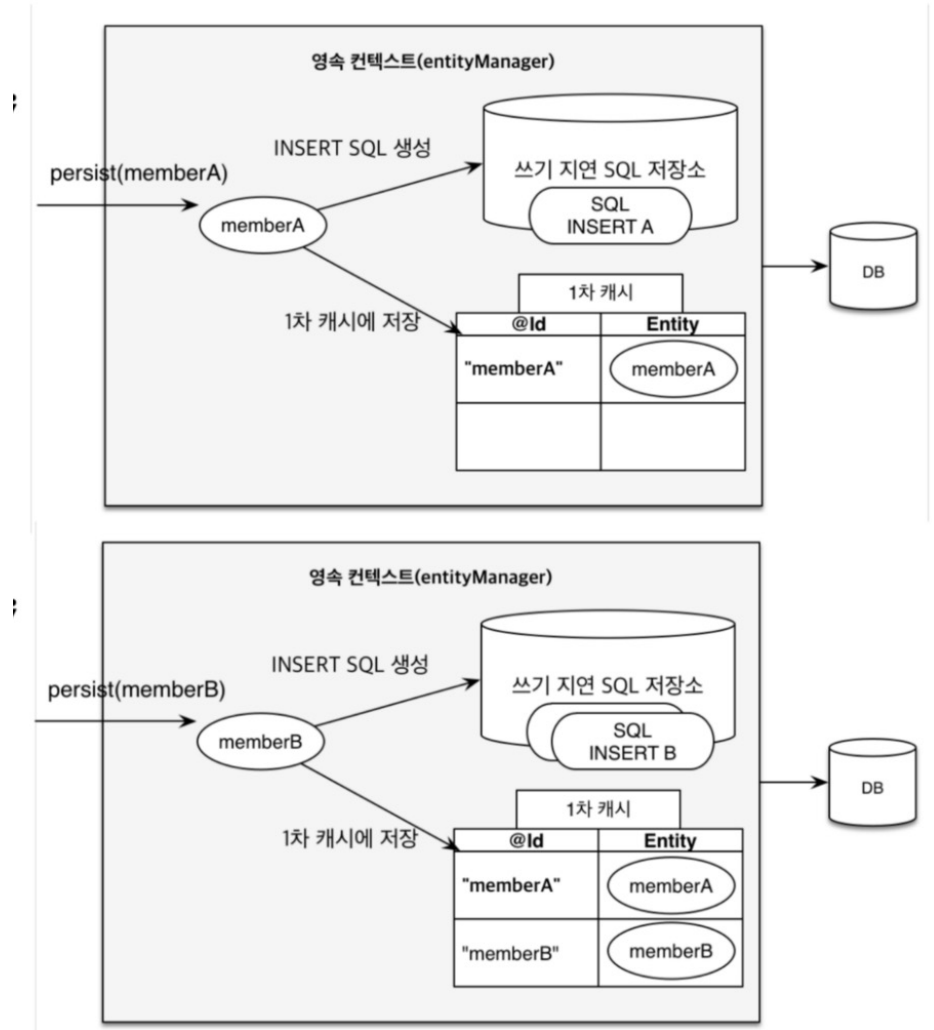


# User login

- 요청 시마다 Header에 유저 정보를 전달해야한다.
- 이 때 탈취를 막기 위해 암호화
  - 세미나에서는 JWT(json web token) 사용
- 요청을 받을지 말지 각각 controller에서 처리할 수도 있지만 모든 함수에 해당 로직을 추가하기 번거롭다.
  - AOP를 적용해 요청이 올 때의 동작을 aspect로 분리해서 처리
  - 세미나에서는 spring security 사용

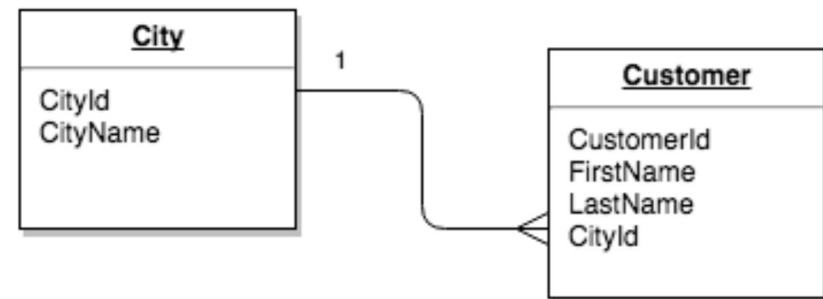
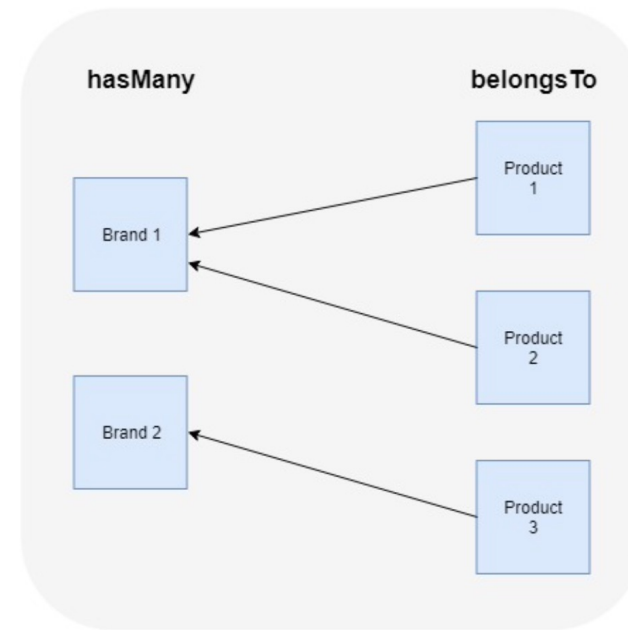
# 영속성

- 엔티티를 영구 저장 하는 환경
- 장점
  - 1차캐시
  - 동일성 보장 (Identity)
  - 트랜잭션을 지원하는 쓰기 지연
  - 변경감지 (Dirty checking)
  - 지연로딩 (Lazy Loading)
- Cascade



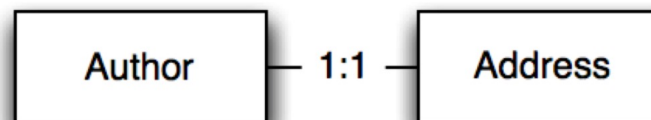
# RDB 복습

- 무엇이 어디에 속하는가?
- 어느 쪽에 ForeignKey를 달 것인가?
  - *Reporters & Articles*
  - *Books & Categories*
  - *Survey results & OSs Action*
- Spring에서 는?
  - `@ManyToOne`
  - `@OneToMany`



# OneToOne

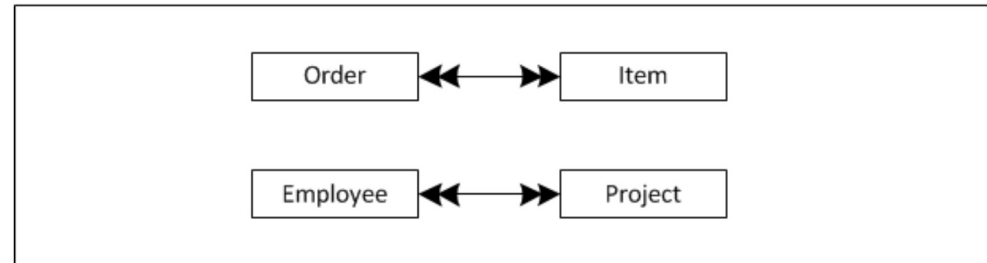
- 1:1의 관계를 나타낸다.
- 해당 관계는 한 테이블로도 충분히 표현 가능하다.
- 신중히 고려해 테이블을 분리시킬지 말지 결정해야 한다.



People		One-to-One	Drivers Licenses	
ID	Name		UserID	Number
1	Alice		1	F25532
2	Bob		2	S43212
3	Cathy		3	B98364

# ManyToMany

- ManyToMany라는 어노테이션을 이용해 정의 가능하지만 내부 구현을 생각해보자
- 독립된 테이블로 직접 다루는 것이 효과적



# Join

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- join은 성능상 무거운 연산
  - 잘 사용해야 한다.
- join의 기본 유형들
  - Inner join
  - natural join
  - left (outer) join
  - right (outer) join

## ■ course natural join prereq

course_id	title	dept_name	credits	prere_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

Note: prereq information missing for CS-315 and course information missing for CS-437.

가좌 테이블			정리된 테이블		
①	②		①	②	
A	10	+	A	Apple	=
B	20		C	Code	
C	30		D	Death	
D	40		D	Dos	
D	50				
A	10		A	10	Apple
B	20		B	20	null
C	30		C	30	Code
D	40		D	40	Death
D	50		D	50	Death
			D	40	Dos
			D	50	Dos

↑ select \* ①, ②, ③  
from 가좌 테이블  
left outer join 정리된 테이블  
on ①. ① = ②. ①

# Join

## ■ 기본 sql 문법

- *SELECT \* FROM response AS r NATURAL JOIN os*
- *SELECT \* FROM response AS r JOIN os USING(os\_id)*
- *SELECT \* FROM response AS r JOIN os ON r.os\_id = os.id*
- *SELECT \* FROM response AS r JOIN os 외 경우?*

# Lazy evaluation

- DB에서는 연관된 관계 매핑을 단순히 id로 저장한다.
- ORM이 적용되면?
  - 연관된 모든 멤버가 객체로 존재
  - 조회 시 비효율이 발생

```
@Entity
class OperatingSystem(
    @field:NotBlank
    val name: String,

    @field:NotBlank
    val description: String,

    @field:NotNull
    val price: Long,

    @OneToMany(cascade = [CascadeType.ALL], mappedBy = "os")
    val surveyResponses : List<SurveyResponse>
) : BaseEntity()
```



# N+1 problem

- 10개의 부모 entity 각각 10개의 자식 entity
- 부모 엔티티 리스트 조회 후 반환 시 11번의 query 호출
- 해결
  - *Fetch join*
    - JPQL의 기능

```
@Query("select o from Owner o join fetch o.cats")  
List<Owner> findAllJoinFetch();
```

- *EntityGraph*

```
@EntityGraph(attributePaths = "cats")  
@Query("select o from Owner o")  
List<Owner> findAllEntityGraph();
```

# Transaction

- 계좌 송금 예제
  - A가 B에게 100만원 송금
  - A계좌 잔액 -= 100만원
  - B계좌 잔액 += 100만원
- 중간에 서버가 터진다면?

# ACID

- Atomicity (원자성)
  - 다 되거나 다 안 되거나, 둘 중에 하나여야 함
- Consistency (일관성)
  - 미리 정해둔 규칙에 맞게 데이터를 저장, 변경해야 함
- Isolation (고립성)
  - 복수 개의 transaction 이 실행될 때 서로 연산이 끼어들면 안 되며, 중간 과정이 이용되어도 안 됨
- Durability (영구성)
  - 일단 commit 했으면 어떠한 문제가 발생해도 영구적으로 반영된 상태여야 함

# 세미나 조사

- 아직 기초가 부족한 것 같다. 천천히 스프링의 여러 기능들을 음미하고 싶다. (난이도 하)
- JPA, DB에 대해 더 배우고 DB 접근에서 성능의 최적화를 이루고 싶다! (난이도 중)
- Webflux + reactor - non-blocking reactive programming을 이용해 서버 단에서 비동기 작업을 최대한 활용해 방대한 트래픽을 다루고 싶다! (난이도 상)