2018 INI Intern Final Project

목차

1. 프로젝트 개요

1.1. 프로젝트 기간

2018.07.24 ~ 2018.08.02 (10일)

1.2. 목적

컨텐츠에 대한 접근 수가 증가하면 서버에 대한 트래픽 양 또한 증가하여 용량이 더 큰 서버로의 컨텐츠 재배치가 요구됨
이를 실시간으로 수행하기 위하여 메모리 기반의 realtime database 인 redis 와 파이썬의 asyncio 라이브러리를 이용한 비동기 파일 재배치 프로그램을 제작

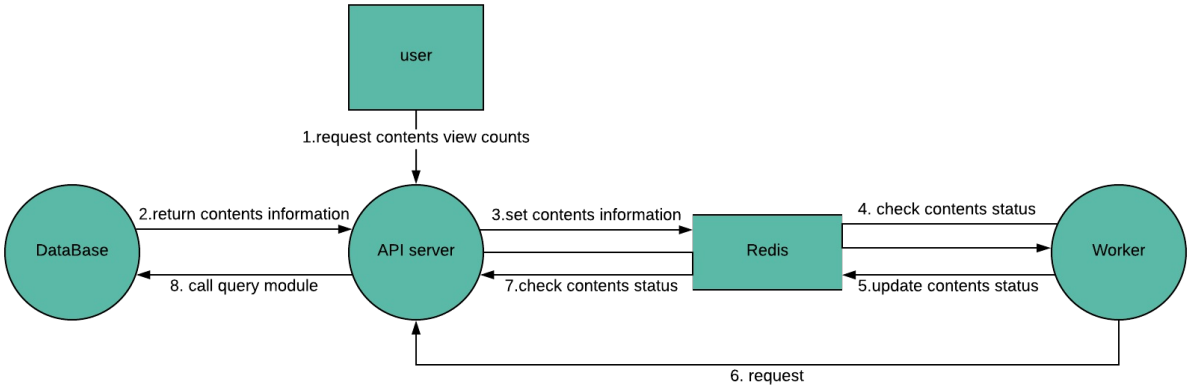
1.3. 참여자 (담당 기능)

- 김지희 (API)

- 박병훈 (Worker)

- 장예훈 (DataBase)

1.4. Process



2. API

담당자: 김지희

2.1. 개발 환경

- OS : ubutu 16.04
- python==3.6.5
- MySQL==5.7
- Flask==1.0.2
- PyMySQL==0.9.2
- redis-server==4.0.10

2.2. 프로세스 상세 및 실행 결과

2.2.1. 컨텐츠 정보 쿼리 및 redis 업데이트(Process step1 ~ step3)

API Specification

- **URL**
  host:port/post_sentence

- **Method:**

  `POST`

- **Data Params**

  `{cid=7&count=664}`

- **Success Response:**
  - **Code:** `200`
  - **Content:**

`{"cid": "7", "count": "1964", "target": "silver", "db_level": "bronze", "filename": "g.mp4", "worker_id": null, "status": "update"}`

- **Faults Response:**
  - **Code:** `404`
  - **Content:**
    - Not found
    - Non existent URI
  - **Code:** `500`
  - **Content:**
    - Internal Server error (MySQL, Redis-server error etc.)
- **Sample Call:**

```
foo@bar:~/$ curl http://192.168.10.108:5000/post_sentence -d "cid=7&count=1964"
{"cid": "7", "count": "1964", "target": "silver", "db_level": "bronze",
"filename": "g.mp4", "worker_id": null, "status": "update"}
```

기능 설명

1. 사용자가 컨텐츠를 조회하면 컨텐츠의 cid 와 count 정보를 포함하여 API를 호출 (e.g.curl http://192.168.10.108:5001/post_sentence -d "cid=3&count=664")
2. db_query 모듈을 이용하여 database의 contents table 과 level table에서 post 된 cid를 가진 content의 현재 위치와 목적 위치를 반환
3. redis-server에 연결하여 post 요청이 들어온 cid에 대한 count, 목적 위치, 현재 위치, filename, worker_id, status 정보를 json type으로 만들어 key:cid, value :json 형식의 정보 로 redis database에 set 한 후 value를 반환 (e.g.)
   `{'3':{"cid": "3", "count": "664", "target": "bronze", "db_level": "silver", "filename": "c.mp4","worker_id": null, "status": "update"}}` 형식으로 저장

worker_id는 worker에서 지정되므로 null로 초기화하며 status는 현재 위치와 목적 위치가 같은 경우

'done', 다른 경우 'update'로 초기화

contents table

#	cid	content_level	filename	generate_time	update_time
1	1	gold	a.mp4	2018-07-29 …	2018-08-02 18:30:17
2	2	bronze	b.mp4	2018-07-29 …	2018-08-02 10:47:40
3	3	silver	c.mp4	2018-07-29 …	2018-08-02 18:30:18
4	4	bronze	d.mp4	2018-07-29 …	2018-08-01 16:25:19
5	5	silver	e.mp4	2018-07-29 …	2018-08-02 18:30:18
6	6	gold	f.mp4	2018-07-29 …	2018-08-01 17:55:23

level table

#	content_level	max_counts	path	min_counts
1	bronze	999	/etc/inisoft/redis_project/bronze/	0
2	gold	3001	/home/inisoft/workspace/inisoft/redis_project/gold/	2000
3	silver	1999	/var/www/html/redis_project/silver/	1000

2.2.2. redis 값 체크 및 MySQL database 업데이트(Process step6 ~ step8)

API Specification

- **URL**
  host:port/update_sentence
- **Method:**

  `POST`
- **Data Params**

  `{cid:7}`
- **Success Response:**
  - **Code:** `200`
  - **Content:**
    `7`
- **Faults Response:**

- **Code:** `404`
- **Content:**
  - Not found
  - Non existent URI
- **Code:** `500`
- **Content:**
  - Internal Server error (MySQL, Redis-server error etc.)

- **Sample Call:**

```
foo@bar:~/$ curl http://192.168.10.108:5000/update_sentence -d "cid=7"
7
foo@bar:~/$ curl http://192.168.10.108:5000/update_sentence -d "cid=8"
check your status again
```

기능 설명

1. worker가 파일을 재배치한 후 해당 contents 의 status 를 'done' 으로 바꾸고 API 를 호출 (e.g. curl http://192.168.10.108:5000/update_sentence -d "cid=3")

2. request를 받으면 cid 를 Key 값으로 redis에서 해당 content의 status 가 'done' 인지 검사하고 MySQL의 contents table 에 새로운 level 과 update time 을 업데이트
   만약 status 가 'done' 이 아니면 "check your status again" 메세지를 반환

   redis status check 후 content_level과 update_time update 결과

#	cid	content_level	filename	generate_time	update_time
7	7	silver	g.mp4	2018-07-29 ...	2018-08-06 17:30:16

3. Worker

담당자: 박병훈

3.1. 개발환경

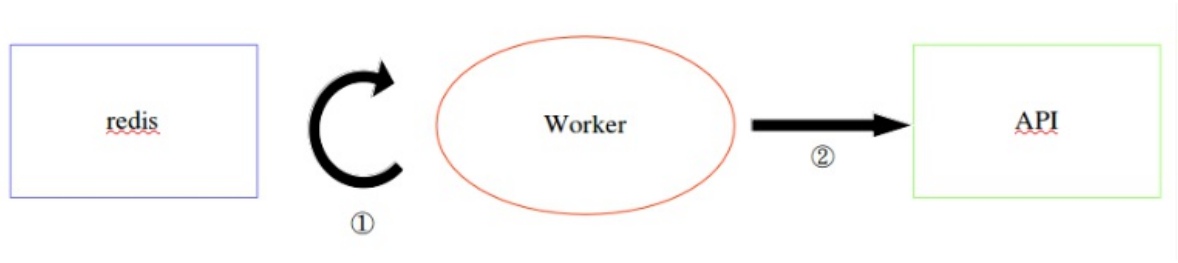
- 서버

- ubuntu (16.0.4 version)

- 백엔드

  - Framework : Python(3.5.2)
  - Database : mysql(14.14 Distrib 5.7.23)
  - Editor : vi

3.2. 실행 흐름



①: Worker는 항시 redis의 정보를 불러 옴. 이 때, redis에 저장된 콘텐츠 중 상태가 'update'인 것이 있으면 Worker는 worker_id를 생성하고 파일이 지정된 서버 경로에 실제로 존재하는지 확인 및 동명의 파일이 존재하는지 검사 후, 파일 이동. 파일 이동이 완료되면 worker_id를 NULL로 변경하고 해당 콘텐츠의 상태를 'done'으로 redis에 업데이트.

②: Worker는 ①작업을 완료한 뒤, API를 호출하여 성공적으로 업데이트를 완료했다는 request 메세지를 보냄. 이 때, 해당 콘텐츠의 cid를 같이 보내서 어떤 콘텐츠의 상태가 업데이트 되었는지 API에 알림.

3.3. 프로그램 설명과 결과물

3.3.1. 모듈

- import redis : redis에 저장된 정보를 확인 할 수 있는 모듈

```
redis_db = redis.StrictRedis(host='192.168.10.37', port=6379, db=1)
```

- import pymysql : 개발 서버의 데이터베이스에 저장된 정보를 확인 할 수 있는 모듈

```
conn = pymysql.connect(host='192.168.10.37',user='root',password='ini6223',db='redis_project',charset='utf8')
curs = conn.cursor()

sql = "select path from level"
curs.execute(sql)
```

- import shutiil : 콘텐츠 이동을 위한 모듈

```
shutil.move( goldpath + filename , silverpath + filename)
```

- import requests : API에게 worker의 작업이 끝났음을 알리기 위한 모듈

```
r = requests.post('http://192.168.10.108:5000/update_sentence', data = {'cid': cid})
```

- import asyncio : worker가 비동기로 작동하게 할 모듈

```
async def redis_func():
```

```
loop = asyncio.get_event_loop()
loop.run_until_complete(redis_func())
loop.close()
```

- import json : redis에 저장된 콘텐츠 정보들을 json 형식으로 불러옴

```
json_input_redis = json.loads(input_redis)
```

3.3.2. 구현 방법

- 조회 수에 따라 콘텐츠들이 위치할 파일 경로(level)에 대한 정보를 개발 서버의 데이터베이스에서 불러옴

```
sql = "select path from level"
curs.execute(sql)

rows = curs.fetchall()
path1 = str(rows[0])
path2 = str(rows[1])
path3 = str(rows[2])

bronzepath = path1[2:-3]
goldpath = path2[2:-3]
silverpath = path3[2:-3]
```

- redis의 콘텐츠들은 cid를 key 값으로 순차적으로 저장되어있으므로 Worker가 redis의 마지막 콘텐츠까지 읽었을 때, 다시 처음부터 읽을 수 있도록 함.

```
if type(redis_db.get(i)) == NoneType :
    i = 1
    print('You go fist')
    continue
```

- json 형식으로 저장되어 있는 redis의 콘텐츠 정보를 Worker에서 json 형식으로 받음. 이 때 콘텐츠의 type은 byte이므로 decode('utf-8')을 사용해서 string으로 type을 변경

```
input_redis = redis_db.get(i).decode('utf-8')
json_input_redis = json.loads(input_redis)
```

- 콘텐츠의 상태가 update이고 worker_id가 NULL 이라면 Worker는 worker_id를 생성하고 파일 이동 작업을 실시.

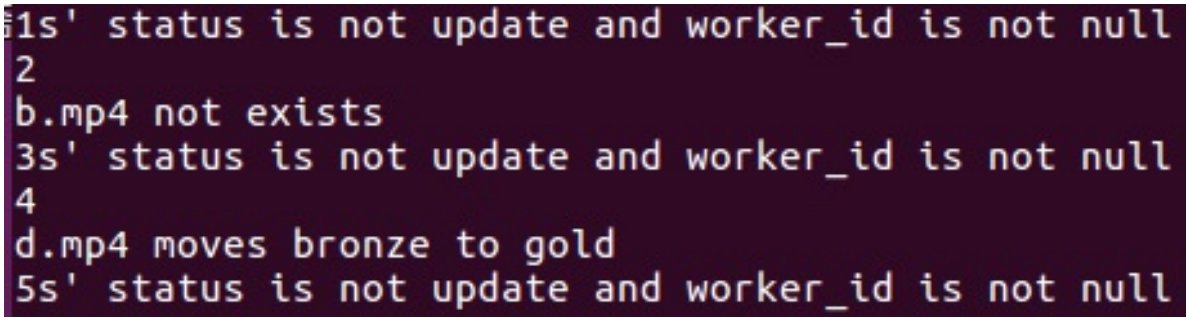
```
json_input_redis['worker_id']='working'
json_input_redis = json.dumps(json_input_redis)
redis_db.set(i,json_input_redis)
```

3.3.3. Worker의 비동기

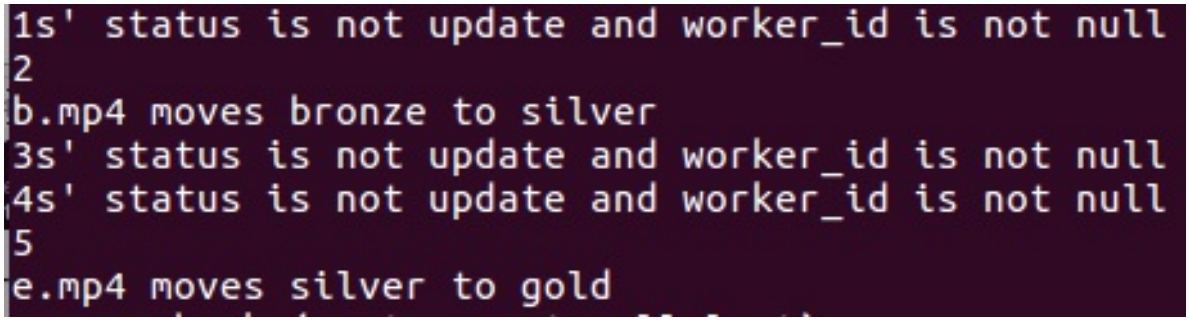
- 동기식 구조에서 프로세서1과 프로세서2가 있을 때, 프로세서1이 모두 진행될 때까지 프로세서2는 대기.
  반대로 비동기로 동작하면 사용자는 프로세서1에게 시작 명령을 내린 뒤, 프로세서2에도 시작 명령을 내리는 것이 가능. 각각 실행을 완료하면 결과를 출력. → Worker는 비동기로 작동하므로 사용자가 Worker를 작동하고 결과를 기다리는 일이 발생하지 않음.

3.3.4. 실행화면

- worker1



- worker2



4. DataBase

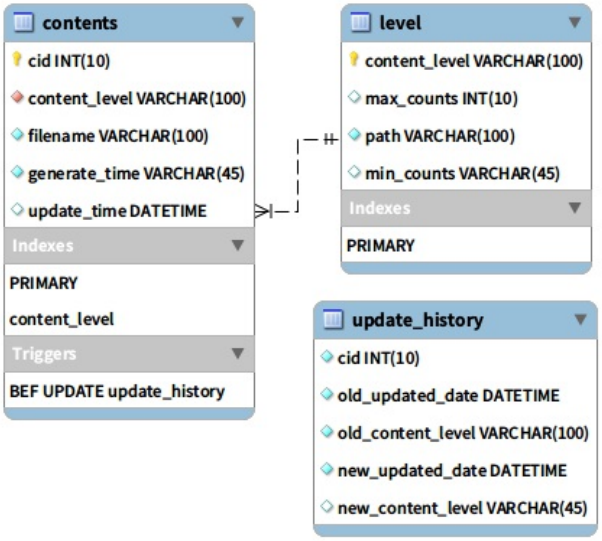
담당자: 장예훈

4.1. 개발 환경

- Ubuntu 16.04.5 LTS
- Python 3.5
- MySQL 5.7.23
- PyMySQL 0.9.2

4.2. DB 구축

4.2.1. ERD



4.2.2. DB 테이블

- level: content의 view count 수에 따른 level 정보를 저장하는 테이블
- contents: content의 정보를 저장하는 테이블
- update_history: contents level 정보가 업데이트 될 때마다 update history를 저장하는 테이블로 contents 테이블의 `content_level`에 업데이트 이벤트가 발생하면 데이터를 insert 하는 트리거와 연결

4.2.3. DB 트리거

- contents 테이블의 업데이트 이벤트가 발생하면 트리거가 작동되고 update_history에 업데이트 시점과 업데이트 된 content level이 기록됨

- 현재 content level과 업데이트 된 content level 이 다를 때만 트리거가 작동

- 트리거

```
CREATE DEFINER = `root`@`localhost` trigger update_history after update on
contents for each row
begin
if (old.content_level != new.content_level) then
  insert into update_history values(old.cid, old.update_time,
    old.content_level, now(), new.content_level);
end if;
end
```

4.3. DB 쿼리 클래스 생성

프로세스 내에서 필요한 db 쿼리 클래스 생성

4.3.1. 클래스 구조

```
class db:
  def __init__(self):
    ...

  def select(self, table, column, where_clause=None, order_by=None):
    ...

  def insert_contents(self, table, cid, file_name):
    ...

  def update_level(self, cid, content_level):
    ...

  def check_max_count(self, count):
    ...
```

4.3.2. 함수 설명

1. __init__

- db 클래스 생성 시 `PyMySQL` 로 MySQL 연결할 때 필요한 파라미터 설정
- `pymysql.connect()` 를 이용하여 MySQL 연결

2. select

- api 서버와 db 간 데이터 상호 교환 시 반복 사용되는 MySQL select 쿼리문을 모듈화 한 것
- `table` , `column` , `where_clause` , `order_by` 가 함수 인자이며 이 중 필수 인자는 `table` 과 `column`

3. insert_contents

- content가 새로 업로드 될 때 콘텐츠에 대한 정보를 MySQL contents 테이블에 insert 해주는 기능을 모듈화 한 것
- min_count가 0인 level이 default `level`
- `generate time` 은 insert 쿼리문 실행 시점이 적용되므로 본 함수는 콘텐츠가 업로드 됨과 동시에 실행되어야 함

4. update_level

- api 서버가 본 함수를 호출하면 MySQL contents 테이블에서 content_level을 update 하는 쿼리 문을 모듈화 한 것
- update 하고자 하는 콘텐츠의 `cid`와 relocate 된 `contet_level`이 필수 인자
- 본 함수가 실행되면 contents table에서 content_level과 update_time column이 업데이트 이전에 update_history 테이블과 연결된 트리거가 먼저 작동

5. check_max_count

- MySQL level 테이블에서 최상위 level의 max_counts와 함수의 파라미터로 들어온 count의 크기를 비교하여 더 큰 count로 최상위 level의 max_counts를 update 해주는 기능을 모듈화 한 것
- api 서버에서 target level을 추출하기 위해 `select 함수`로 level 테이블을 쿼리하기 전에 실행됨

4.4. 프로세스 상세 및 실행 결과

4.4.1. redis에 content 정보 업데이트

1. api 서버 - db 쿼리 함수 호출

- api 서버는 content의 현재 위치 level과 count가 해당되는 범위의 위치 level 비교를 위해 쿼리 모듈로 db에서 데이터를 쿼리 함
  - content 테이블에서 user로부터 받은 cid로 해당 content의 현재 위치 level을 쿼리 하는 `select 함수` 호출
  - `select 함수`로 level 테이블의 모든 coulmn을 쿼리 한 후 user로부터 받은 count와 비교 연산하여 target level을 반환함

2. db - 쿼리 결과 api 서버에 반환

- db는 `select 함수`로 쿼리 된 결과를 api 서버에 반환함
  - MySQL content 테이블 내의 cid 1 정보

#	cid	content_level	filename	generate_time	update_time
1	1	gold	a.mp4	2018-07-29 15:31:24	2018-08-02 18:30:17

  - api 서버에서 `select 함수`를 호출하여 cid가 1인 content의 level 추출하는 함수 결과

```
#python shell
\>>> import redis_encode as re
\>>> re.get_level_from_db(1)
{'content_level': 'gold', 'file_name': 'a.mp4'}
```

  - MySQL level 테이블

#	content_level	max_counts	path	min_counts
1	bronze	999	/etc/inisoft/redis_project/bronze/	0
2	gold	3001	/home/inisoft/workspace/inisoft/redis_project/gold/	2000
3	silver	1999	/var/www/html/redis_project/silver/	1000

- api 서버에서 `select 함수`로 level 테이블을 쿼리 한 후 각 level의 count와 countent의 count를 비교 연산하여 target level을 반환하는 함수 결과

```
#python shell
\>>> import redis_encode as re
\>>> re.get_target(2342)
'gold'
\>>> re.get_target(1548)
'silver'
\>>> re.get_target(356)
'bronze'
```

3. api 서버 - redis에 content에 대한 정보 업데이트

- db에서 쿼리 한 데이터를 알맞게 처리한 후 redis에 content 정보를 업데이트

4.4.2. relocate가 완료된 content 정보 업데이트

1. api 서버 - redis 업데이트

- content의 relocate를 마친 worker의 호출을 받은 api 서버는 해당 cid에 대해 redis의 content status가 'done' 인지 확인

2. api 서버 - db 업데이트

- redis의 content status가 'done' 인 것을 확인 한 api 서버가 쿼리 모듈로 db에서 content status를 업데이트 하는 `update_level 함수`를 호출
  - `update_level 함수` 호출 전 MySQL content 테이블 내의 cid 21 정보

#	cid	content_level	filename	generate_time	update_time
1	21	bronze	test.mp4	2018-08-02 15:39:36	NULL

  - api 서버에서 `update_level 함수` 호출 결과

```
\>>> import redis_encode as re
\>>> re.update_db_level(21, 'silver')
1
```

- `update_level 함수` 호출 후 MySQL content 테이블 내의 cid 12 정보

#	cid	content_level	filename	generate_time	update_time
1	21	silver	test.mp4	2018-08-02 15:39:36	2018-08-10 17:35:54

3. db 업데이트

- api 서버의 쿼리 모듈 호출로 content 테이블이 업데이트 되면 db에 `insert 트리거` 가 작동하여 content 테이블 업데이트 시점에 update_history 테이블에 row가 추가됨

  - MySQL content 테이블의 `insert 트리거` 작동 전 cid 21 의 update_history 테이블

#	cid	old_updated_date	old_content_level	new_updated_date	new_content_level
1	21	NULL	bronze	2018-08-10 17:34:43	silver

  - MySQL content 테이블의 `insert 트리거` 작동 후 cid 21 의 update_history 테이블

#	cid	old_updated_date	old_content_level	new_updated_date	new_content_level
1	21	NULL	bronze	2018-08-10 17:34:43	silver
2	21	2018-08-10 17:35:54	silver	2018-08-10 17:39:48	gold

5. 프로젝트 후기

김지희
내가 구현한 API의 기능이 까다롭거나 개발 난이도가 높지는 않았지만 과제를 받을 당시에는 API 개념이나 개발 목적에 대한 지식이 부족하여 어떤 기능이 있어야 하는지조차도 모호했다. 이 때문에 개발 자체보다 개발을 위해 공부하고 소통하는 과정에서 더 많이 고민했던 것 같다. 공부하고 팀원들과 회의를 하는 과정에서 요구 사항이 명확해져서 결국 예상보다 빠르게 프로젝트를 끝낼 수 있었다.
개발 중 겪은 어려움으로는 remote server에서 이용되는 library나 module의 가용성을 계속 체크해야 해서 시간이 소요되었고 권한 문제나 보안 상의 이유로 파일의 설정 또한 로컬에서 단독으로 개발 할 때보다 복잡했다. 이 프로젝트를 통해 API 와 같이 이전에 접해보지 못한 개념에 대해 제대로 공부할 수 있어서 좋았고 에러 해결을 하면서 구글링 실력이 향상된 것을 느낄 수 있었다.

박병훈
worker를 작업하며 얻은 가장 큰 수확은 파이썬의 문법 및 모듈에 대한 많은 실습을 해보았다는 것이다. c/c++만을 사용해왔기 때문에 파이썬은 낯설고 어려운 언어였지만 이번 프로젝트를 통하여 많이 친숙해졌다. 전체적인 프로젝트의 흐름을 이해하며 데이터베이스, API의 쓰임새를 이해한 것 또한 나에겐 좋은 경험이었다. worker를 작동하기 위해선 개발 서버의 데이터베이스와의 연결, API호출 등의 이해가 필요한데 이러한 부분을 코딩하며 충분하진 않지만 서버에 대하여 최소한의 지식을 쌓은 것 같다.
팀 프로젝트를 진행하며 많은 부담감이 있었지만 기한 내에 결과물을 만들 수 있어서 뿌듯했다. 항상 어렵게 느껴졌었던 프로젝트를 잘 마무리하며 자신감을 얻을 수 있었고 둘도 없는 좋은 경험이었다.

장예훈
이번 프로젝트를 진행하면서 가장 많이 배운 점은 코드의 완성도를 높이는 방법에 대한 고민과 각 구현

기능들이 요구사항에 얼마나 적합한 지 검토하는 방법인 것 같다. 다른 파트들에 비해 DB 모듈화는 개인적으로 러닝커브나 코딩 난이도, 진입장벽이 낮았기 때문에 스스로도 해당 기능 완성도에 대한 기대치가 높았다.
기한 내에 요구사항에 완벽히 들어 맞는 기능을 구현하기 위해서는 설계부터 사용자들의 피드백까지 체계적인 단계가 필수적이라고 생각하였다. 그래서 팀원들과 회의도 수차례 진행하고, 요구사항 명세화 작업도 하며 완성도에 집중하였고 결국엔 요구사항에 부합하는 프로세스를 구현하였다.
처음부터 내가 직접 설계하고 구현하며 피드백을 받아 보완하는 스텝을 차례대로 겪고 나니 성취감도 높았고, 프로젝트에 대한 애착이 생겼으며 다른 프로젝트를 진행할 때도 막막한 느낌 없이 잘 해낼 수 있을 것 같다.