Buzzni Backend Engineer Assignment

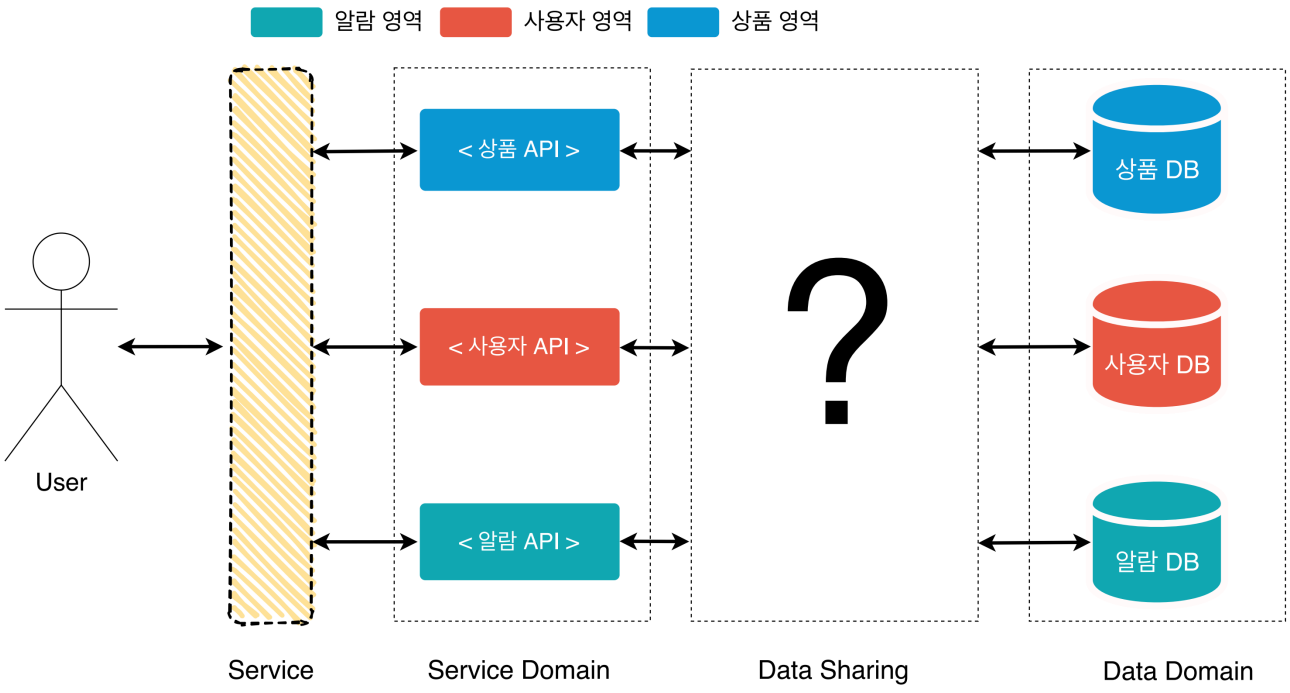
버즈니 # 백엔드 # 마이크로서비스 # 데이터베이스

버즈니의 새내기 백앤드 엔지니어 **준익**은 **"홈쇼핑모아"** 서비스에서 최근 본 상품 API, 상품 리스트 API, 알람 정보 API를 만들어 달라는 요청을 받았다. 설계를 위해 **준익**은 기존에 구성된 데이터베이스 스키마를 확인해 보았고 스키마는 다음과 같았다.

상품 DB <Item> Table	사용자 DB <User> Table	알람 DB <Alarm> Table
id: int name: str broadcaster: str category: str price: int	id: int name: str access_token: str	id: int item_id: int user_id: int created: datetime

상품 DB, 사용자 DB 그리고 알람 DB가 있었으며, 데이터베이스가 도메인에 따라 분리되어 있었다. 이는 일반적인 SQL Join을 이용하여 데이터베이스를 조회할 수 없다는 것을 의미했다. **준익**은 적절한 방법을 찾다가 최근 본 상품 API, 상품 리스트 API, 알람 정보 API를 만들기 위해서는 위 표에 표시된 각각의 데이터베이스에 모두 접근하여 데이터를 합쳐 제공해야 한다고 생각했다.

그러나 "**홈쇼핑모아**"에서 각각의 데이터베이스가 분리된 이유는 도메인 기반의 마이크로서비스를 추구하기 때문이라는 사실을 깨달았다. 마이크로서비스는 서비스간 느슨한 결합 (Loose Coupling)을 추구하기 때문에 한 도메인의 속한 서비스는 같은 도메인에 속한 데이터베이스에만 접근해야하고, 다른 도메인의 데이터베이스에는 **직접 접근** 하지 않아야한다. 이 사실에 **준익**은 좌절 했으며, 이를 해결할 수 있는 방법을 찾아야 했다.



< 홈쇼핑모아 설계도 >

서비스 API는 같은 도메인의 데이터 이외에도 다른 도메인의 정보가 필요하다. 예를 들어 상품 리스트 API 는 상품 정보(상품 DB)와 알람 세팅 정보(알람 DB)가 필요하다.

결과적으로 위 <홈쇼핑모아 설계도>와 같이 데이터 도메인과 서비스 도메인을 구성했을 때 **준익**을 도와 Data Sharing 부분을 설계하고 데이터를 Mixing하여 서비스 할 수 있는 API를 설계하고 개발하여 아래 'Todo List'를 수행하자.

Todo List

1. Data Sharing 설계 및 구현
2. Service Domain에 포함된 API 설계 및 구현 (API Specification 참고)
3. 문서(markdown 혹은 PDF) 작성

Service API Specification

모든 API는 공통적으로 Header에 USER-TOKEN이 들어가야합니다.

상품 API

GET	http://{item_api} - 상품 목록 API
상품의 전체 목록을 주는 API입니다. 상품 목록 내의 상품 정보에는 API를 요청한 유저의 알람 설정 여부가 표시되어야 합니다. 리스트는 상품 id의 내림차순으로 정렬되어야 합니다.	

Request

```
curl -H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X GET http://localhost:5001/
```

Response

200 성공

```
{
  "count": 50,
  "payload": [
    {
      "id": 50,
      "name": "17WINTER 레그미인",
      "broadcaster": "cjmall",
      "category": "패션·잡화",
      "price": 78900,
      "is_alarm_set": true
    },
    .. (생략)
  ]
}
```

GET	http://{item_api}/{item_id} - 단일 상품 상세보기 API

단일 상품의 정보를 응답하고, 조회한 유저의 알람 여부가 표시 되어야 합니다.

이 API를 조회하면 최근 본 상품 이력에 남아야 합니다.

Request

```
curl -H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X GET http://localhost:5001/{item_id}
```

Response

200 성공


```
{
    "payload": {
        "id": 50,
        "name": "[역대최다-13종]17WINTER 레그미인",
        "broadcaster": "cjmall",
        "category": "패션·잡화",
        "price": 78900,
        "is_alarm_set": true
    }
}
```


404

```
{}
```

알람 API

POST	http://{alarm_api} - 알람 설정 API

알람상태를 true로 변경합니다.

Request

```
curl -d '{"item_id":"<item_id>"}' \
-H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X POST http://localhost:6001/
```

Response

201 성공	{ }
404 결과없음	{ }

GET **http://{alarm_api} - 알람 목록 API**

유저가 알람 설정한 상품 리스트가 나와야 합니다.

id 내림차순으로 구현해야 합니다.

Request

```
curl -H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X GET http://localhost:6001/
```

Response

200 성공


```
{
  "count": 5,
  "payload": [
    {
        "id": 50,
        "name": "[역대최다-13종]17WINTER 레그미인",
        "broadcaster": "cjmall",
        "category": "패션·잡화",
        "price": 78900,
    },
    .. (생략)
  ]
}
```

DELETE	http://{alarm_api} - 알람 해제 API
알람상태를 false로 변경합니다.	

Request

```
curl -d '{"item_id":"<item_id>"}' \
-H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X DELETE http://localhost:6001/
```

Response

200 성공	{ }

사용자 API

GET	http://{user_api} - 사용자 목록 API
id 내림차순으로 구현해야합니다.	

Request

```
curl -H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X GET http://localhost:7001/
```

Response

200 성공


```
{
  "count": 10,
  "payload": [
    {
        "id": 10,
        "name": "박규경",
        "access_token": "4a49145e-45a8-4845-af45-1af2a59bb044"
    },
    .. (생략)
  ]
}
```

GET	http://{user_api}/visit - 최근 본 상품 목록 API

유저가 방문한 상품을 최근 방문한 순서대로 보여줘야합니다.

각 상품에 유저의 알람 여부가 표시 되어야 합니다.

같은 상품을 연달아 방문하면 한번만 보여져야 합니다.

최대 3개까지 보여줘야 합니다.

- 방문 순서: A, B, C, D, E, E
- 최근 본 상품 : E, D, C

Request

```
curl -H "Content-Type: application/json" \
-H "USER-TOKEN: 4a49145e-45a8-4845-af45-1af2a59bb044" \
-X GET http://localhost:7001/visit
```

Response

200 성공

```
{
  "count": 3,
  "payload": [
   {
      "id": 50,
      "name": "[역대최다-13종]17WINTER 레그미인",
      "broadcaster": "cjmall",
      "category": "패션·잡화",
      "price": 78900,
      "is_alarm_set": true
   },
   .. (생략)
  ]
}
```

Description

제약사항

1. docker-compose up —build 커맨드 수행시 과제가 **반드시 실행**되어야 합니다.
   - 커맨드가 정확히 동작하지 않으면 평가에 불이익이 있습니다.
   - 도커의 볼륨 마운트 사용을 금지합니다.
2. API 결과물은 최소한 '서비스 API 명세'를 따라야 합니다.(개발의 편의를 위해 추가는 가능합니다.)
3. 소스코드는 python으로 작성되어야 합니다.
4. 요구사항을 충족하기 위해 docker-compose.yaml를 수정할 수 있습니다.
   - 설계를 완성하기 위한 추가 리소스(redis, kafka, rabbitMQ등)혹은 Dockerfile을 추가해도 좋습니다.
5. 각 도메인의 init_database.sql은 수정할 수 없습니다.
6. 다른 서비스(container)의 상태에 적게 영향 받도록 고려하여 설계하세요. (예: 다른 서비스가 동작하지 않을 때를 고려해야합니다. docker-compose.yaml의 일부분을 주석처리해 테스트 할 수 있습니다.)

제출물

- 소스 코드
  - 작업한 코드
  - dockerfiles
  - docker-compose
- 문서 (PDF또는 markdown)에 포함되어야 하는 내용
  - 설계에 대한 이미지 또는 설명
  - 같은 서비스에 속한 데이터에 접근 하는 방법과 설계에 대한 이유
  - 다른 서비스의 데이터로 접근 하는 방법과 설계에 대한 이유
  - 다른 서비스가 죽었을 때의 동작과 설계에 대한 이유

제공하는 것

- Dockerfile (skeleton)
- docker-compose.yaml (skeleton)
- 상품, 유저 데이터