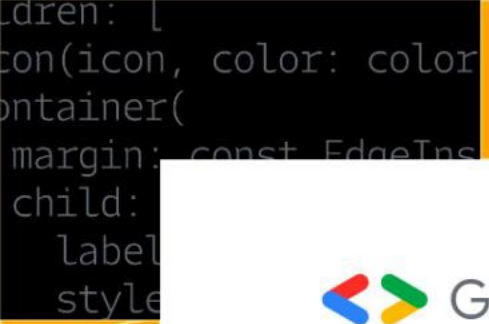


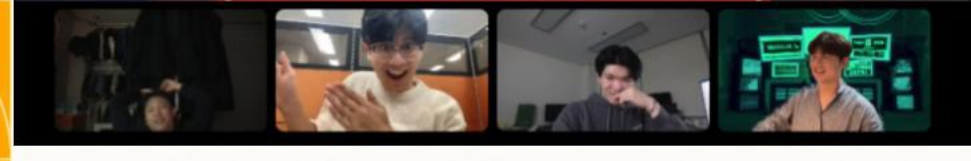
Google Developer Student Clubs
Dongguk University

GDSC- 1st Project: Heart Signal 회고

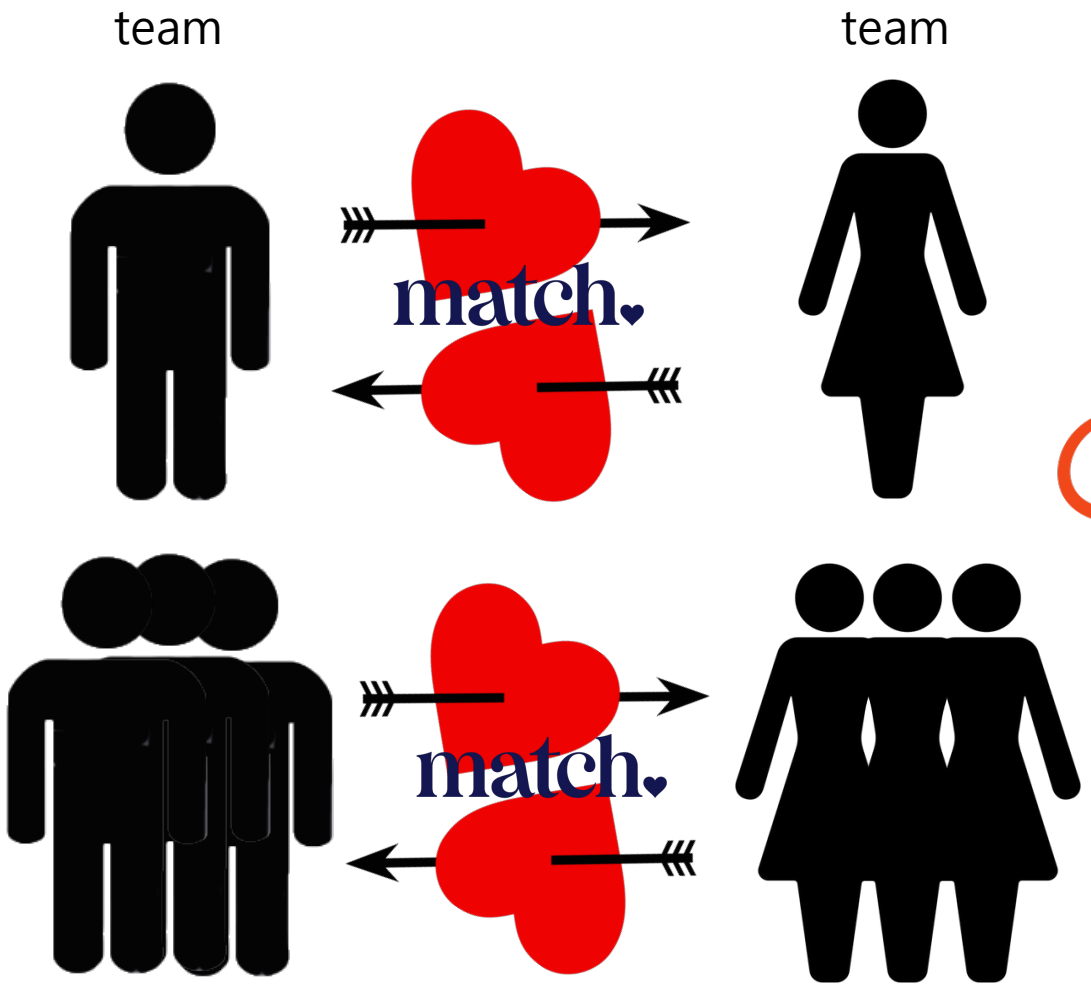
-DB 프로시저 언어와 트리거를 중심으로

Jang DongKyeom / Server/Cloud

Friday October 12



Heart Signal이란?



첫 커밋 10/04, Product 배포: 10/11

주어진 기간 단 일주일..

Commits on Oct 4, 2023

초기설정-연관관계 오류

jjuuuunnii committed last week

be1a6b5

But, 성공적인 배포

총 477명의 실사용자

469	643	남성	내봬누	ESTJ
470	644	남성	띠로링	ISTP
471	645	남성	은새	ISTP
472	646	남성	맵	ENTP
473	647	여성	킥차	ENFP
474	648	남성	사중	ESTJ
475	651	여성	유드림스	ESTJ
476	652	남성	Qwerty	ENTP
477	653	여성	하잉	ESFP

약 100번의 매칭

	meeting_chat_room_id	team1_id	team2_id
1	33	82	69
2	48	118	126
3	61	127	137
4	64	52	56
5	66	100	106
6	79	49	184
7	84	220	227
8	87	241	130
9	89	246	255
10	93	266	240
11	95	234	229
12	97	229	234
13	100	230	245

활발한(?) 주점채팅

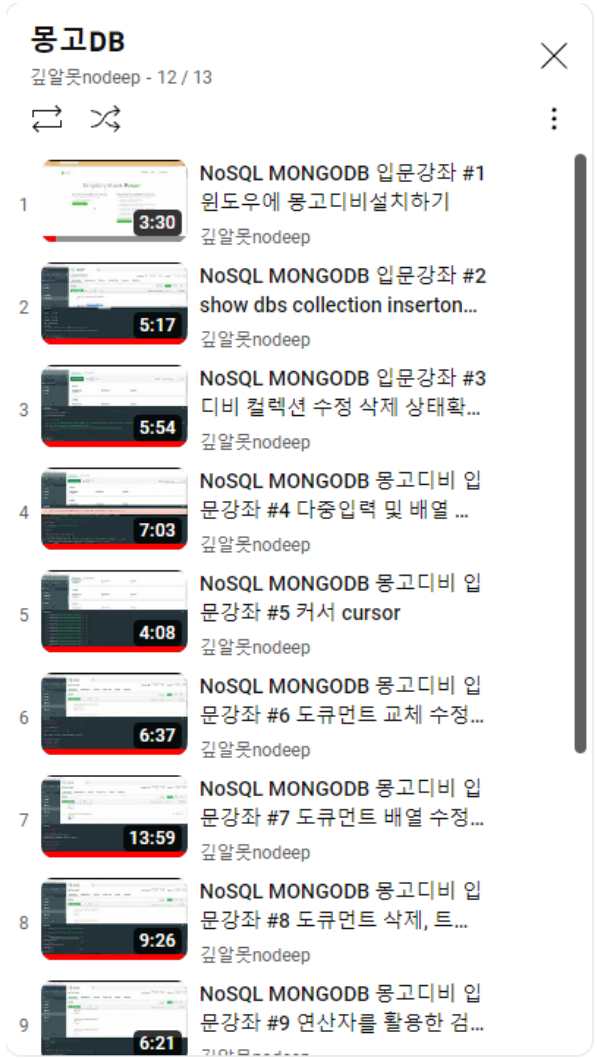
내가 한 일

- DB 설계 + 프로시저, 트리거
- API 최적화
- 미들웨어 구축(RDS, Mongo DB)
- 시스템 테스팅

기획 단계(추석 연휴)

- 서비스의 주 기능은 " 채팅"

-> NoSQL 사용



기획 단계(추석 연휴)

· 축제사이트 · 서버

기획 상에 존재하는 기능들을 구현하는 데에 있어서,
각종 Join과, 트랜잭션 데이터(유저,팀,시그널 관련 정보)를 처리해야 함

++ 모두가 NoSQL이 어색하여 API 설계에 어려움 존재

"동겸아 RDB 설계좀 해줘라 "

유저 정보, 팀, 시그널, 채팅방 정보

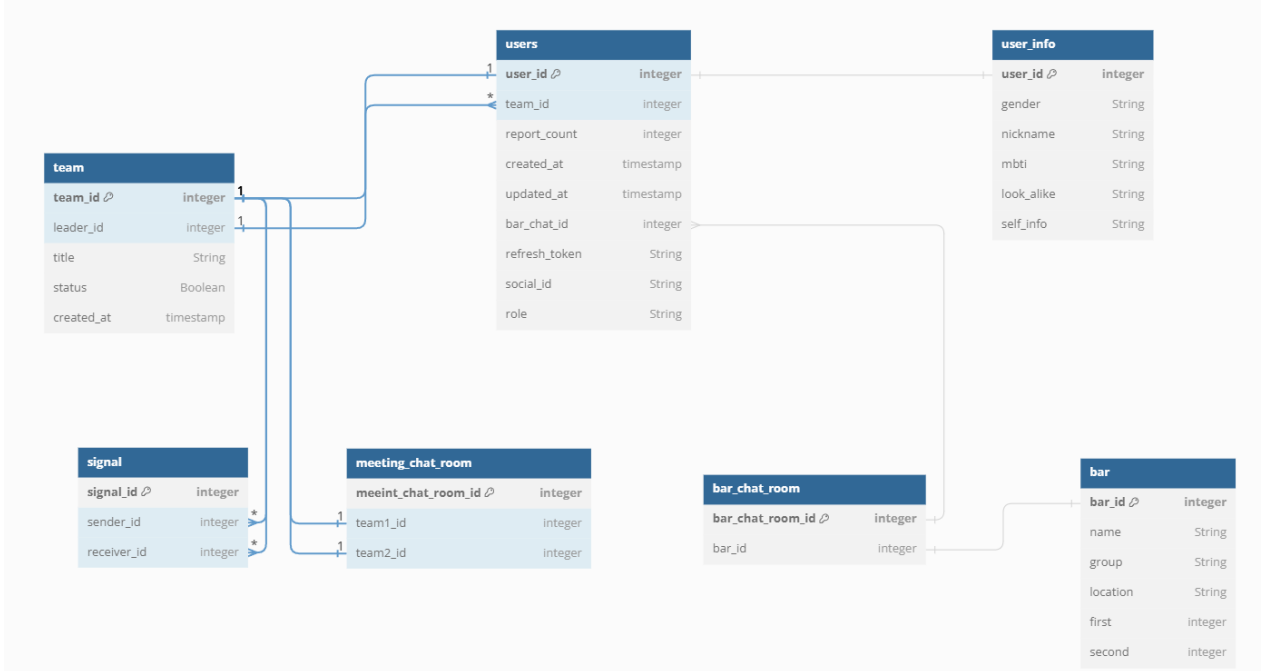
MySQL + MongoDB

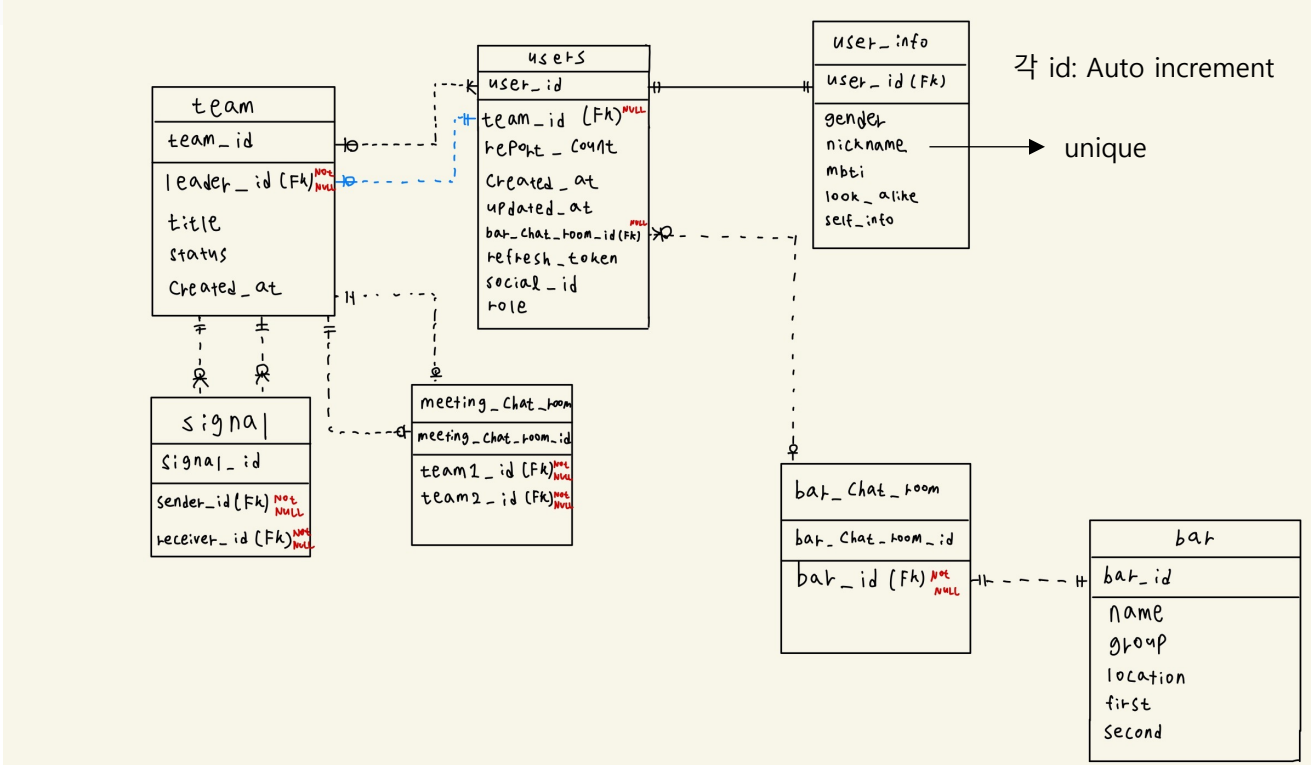
채팅 그 자체

TALK

DB 설계

1. [사용자]는 카카오 회원가입을 통해 [서비스]에 등록한다.
2. [사용자]는 [서비스]에 추가적인 정보를 기입한다.
3. 추가적인 정보로는 성별, 닉네임, MBTI, 닮은 상, 자기소개가 존재한다.
    a. 닉네임은 중복되지 않는다.
4. [사용자]는 [서비스]를 통해 매칭이 가능하다.
5. [사용자]는 인원 수를 선택하고, 팀원을 닉네임으로 초대한다. 해당 미팅의 제목을 기입한 이후 매칭을 등록한다.
6. [서비스]는 [사용자]에게 매칭 시 입력한 인원수와 맞는 매칭을 등록시간 기준 내림차순으로 정렬해 보여준다.
7. [서비스]가 [사용자]에게 제공하는 정보는 미팅방 제목, 인원 수 등 간단한 정보이다.
8. [사용자]가 미팅방을 클릭한다면 [서비스]는 [사용자]에게 상세정보를 제공한다.
9. [서비스]는 [사용자]에게 서로 다른 성별의 방을 제공하며, 매칭 대기 상태인 방만 제공한다.
10. [사용자]는 [서비스]가 제공하는 미팅 방에 좋아요(시그널)을 남길 수 있다.
    a. 좋아요는 다수의 팀에게 보낼 수 있다.
11. [서비스]는 [사용자]에게 좋아요를 보낸 상대와 관련된 진행사항을 알려준다.
    a. 진행사항은 확인전, 수락, 거절이 존재한다.
    b. 상대에게 좋아요를 받았다면 상대 팀의 정보 (MBTI, 자기소개 등)을 확인 가능하다
12. [서비스]는 [사용자]에게 미팅과 관련된 채팅방을 제공한다.
    a. 각 [사용자]들은 본인의 닉네임으로 자유롭게 채팅 가능하다.
    b. [사용자]가 채팅방에 진입 시 채팅방은 리스트에서 자동으로 숨김처리 된다.
    c. 하나의 팀은 하나의 팀만 채팅이 가능하다.
13. [서비스]는 [사용자]에게 주점 리스트 정보를 제공한다
    a. 주점 리스트로는 주점 이름, 주점 단체, 주점위치, 컨셉 등을 보여준다.
    b. 리스트를 통해 주점정보를 제공하여 하나의 창에서 관리 가능하다.
14. [사용자]는 주점 리스트 중 하나의 주점의 채팅방에 진입 가능하다.
15. [서비스]는 [사용자]가 주점별 채팅방에 진입 시 경고창을 제공한다.
    a. 해당 경고창에 따라, 적절하지 못한 채팅은 퇴장당할 수 있다.
    b. 해당 기준으로는, 특정 인물이나 단체를 향한 심한 욕설 및 비방, 성희롱과 같은 성 관련 언행이 존재한다.
16. [서비스]는 [사용자]에게 신고기능을 제공한다.
    a. [사용자]는 다른 [사용자]를 신고할 수 있으며, 신고 5회가 누적될 경우 신고를 당한 [사용자]는 밴된다.
17. [서비스]는 [사용자]가 참여하고 있는 미팅 채팅방 정보를 제공한다.





이제 뭐하지...?

- 설계가 너무 빨리 끝나버림....
- 내 할일 끝...???

API를 더 단순하게 짤 수 있도록 해보자!

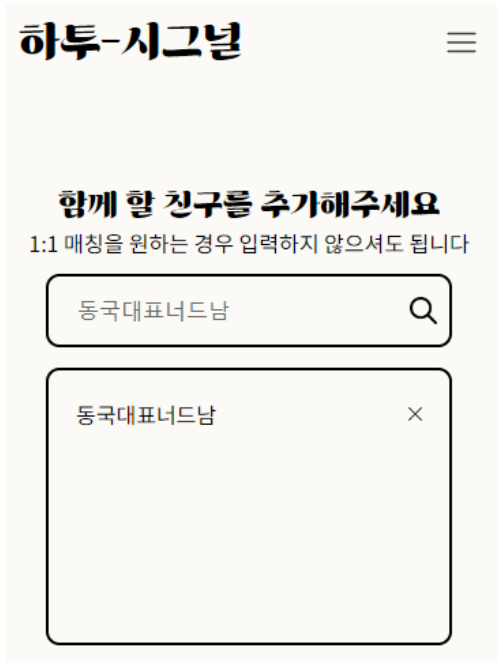
DB 프로시저 언어란

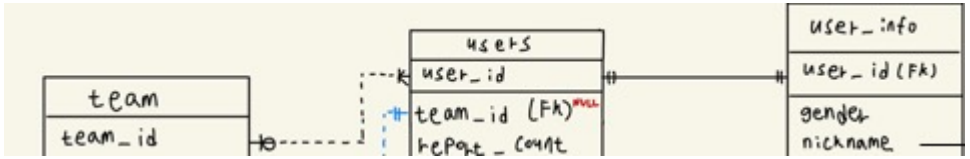
- DBMS 에서 작성되는 프로그래밍 언어.
- 한 번의 호출로, 여러 SQL 문 실행 가능
  - 하나의 API에서 발생하는 연쇄적인 쿼리문 한번에 처리 가능
- 사전에 정의하여, 재사용 가능.
- Oracle: PL/SQL
- MySQL: Stored Procedure Language

프로시저 이용하기

- 우리 서비스가 제공하는 기능에 따르면, 팀 생성, 미팅방 생성의 처리를 하기 위해선, DB에 연쇄적인 데이터 입출력이 필요했음
- Ex) 팀 생성

Client





DataBase

팀 생성 API 호출
"동국대표너드남"

Server

닉 존재해?
ㅇㅇ
팀 없어?
ㅇㅇ
팀생성: team_id=1
해당 닉네임의 user 찾아, team_id를 1로..

1
1
1
1

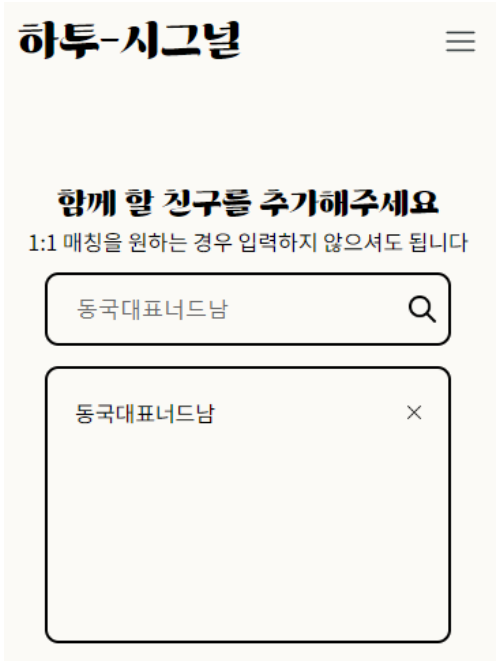
MySQL

아무리 최적화를 해도 최소 4번의 쿼리 필요
-> '팀 생성'이라는 API 로직 자체도 호흡이 더 길어지고, 복잡해질 것

프로시저 이용하기

- 앞선 로직을 단 한번의 쿼리로 처리하게 해보자!
- createTeam()이라는 프로시저를 생성.
  - 팀의 리더 아이디, 팀 제목, 팀원들 닉네임 리스트를 인자로 받아와,
  - 닉네임들이 실제 존재하는지 예외처리
  - 닉네임을 가진 유저가 이미 팀에 속해있는지 예외처리
  - 팀을 생성
  - 각각의 닉네임과 맵핑되는 user의 team_id를, 생성된 팀의 team_id로 바꿈

Client



팀 생성 API 호출
"동국대표너드남"

Server

call createTeam()

DataBase

1

API 로직이 단순해짐!

프로시저 코드 createTeam()

```
create
    definer = hs_admin@`%` procedure createTeam(IN leader_id bigint, IN title varchar(255), IN user_nicknames text)
BEGIN
    DECLARE cur_nickname VARCHAR(255);
    DECLARE leader_nickname VARCHAR(255);
    DECLARE temp_nick VARCHAR(255);
    DECLARE user_count INT DEFAULT 0;
    DECLARE cur_pos INT DEFAULT 1;
    DECLARE next_pos INT;
    DECLARE error_message VARCHAR(255);
    DECLARE leader_exists INT DEFAULT 0;
    DECLARE v_finished INTEGER DEFAULT 0;

    DECLARE cur CURSOR FOR SELECT nickname FROM temp_user_nicknames;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_finished = 1;

    SELECT COUNT(*) INTO leader_exists FROM users WHERE user_id = leader_id;

    IF leader_exists = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'leader_id가 Users table에 존재하지 않습니다.';
    END IF;

    SELECT user_info.nickname
    INTO leader_nickname
    FROM users
             JOIN user_info ON users.user_id = user_info.user_id
    WHERE users.user_id = leader_id;

    CREATE TEMPORARY TABLE IF NOT EXISTS temp_user_nicknames (
        nickname VARCHAR(255)
    );


    WHILE cur_pos < CHAR_LENGTH(user_nicknames) DO
            SET next_pos = LOCATE(',', user_nicknames, cur_pos);
            IF next_pos = 0 THEN
                SET next_pos = CHAR_LENGTH(user_nicknames) + 1;
            END IF;
            SET cur_nickname = TRIM(SUBSTRING(user_nicknames, cur_pos, next_pos - cur_pos));
            INSERT INTO temp_user_nicknames (nickname) VALUES (cur_nickname);
            SET cur_pos = next_pos + 1;
        END WHILE;

    SELECT COUNT(*) INTO user_count FROM temp_user_nicknames WHERE nickname = leader_nickname;
    IF user_count = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '리더의 닉네임이 팀원 목록에 포함되지 않았습니다.';
    END IF;

    OPEN cur;
    fetch_loop: LOOP
        FETCH cur INTO cur_nickname;
        IF v_finished = 1 THEN
            LEAVE fetch_loop;
        END IF;

        SELECT COUNT(*) INTO user_count
        FROM users
                 JOIN user_info ON user_info.user_id = users.user_id
        WHERE user_info.nickname = cur_nickname AND users.team_id IS NOT NULL;

        IF user_count > 0 THEN
            SET error_message = CONCAT('닉네임 "', cur_nickname, '"을 가진 사용자는 이미 팀에 속해 있습니다.');
            DROP TEMPORARY TABLE IF EXISTS temp_user_nicknames;
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END LOOP;
    CLOSE cur;

    INSERT INTO team (leader_id, title, status, created_at) VALUES (leader_id, title, 0, NOW());

    SET @last_team_id = LAST_INSERT_ID();

    UPDATE users
        JOIN user_info ON users.user_id = user_info.user_id
    SET users.team_id = @last_team_id
    WHERE user_info.nickname IN (SELECT nickname FROM temp_user_nicknames);

    DROP TEMPORARY TABLE IF EXISTS temp_user_nicknames;
END;
```

프로시저 이용하기

```
1 usage  DongKyeomJang +1
public void saveTeam(User leader, List<User> members, String title){
    Long leaderId = leader.getId();

    String userNicknames = members.stream() Stream<User>
                                .map(member -> member.getUserInfo().getNickname()) Stream<String>
                                .collect(Collectors.joining( delimiter: ","));

    StoredProcedureQuery storedProcedureQuery = entityManager.createStoredProcedureQuery( procedureName: "createTeam");
    storedProcedureQuery.registerStoredProcedureParameter( parameterName: "leader_id", Long.class, ParameterMode.IN);
    storedProcedureQuery.registerStoredProcedureParameter( parameterName: "title", String.class, ParameterMode.IN);
    storedProcedureQuery.registerStoredProcedureParameter( parameterName: "user_nicknames", String.class, ParameterMode.IN);

    storedProcedureQuery.setParameter( name: "leader_id", leaderId);
    storedProcedureQuery.setParameter( name: "title", title);
    storedProcedureQuery.setParameter( name: "user_nicknames", userNickna

    try {
        storedProcedureQuery.execute();
        log.info("팀 구성 완료");
    } catch (Exception e) {
        log.error("팀 구성 중 오류 발생", e);
    }

}
```

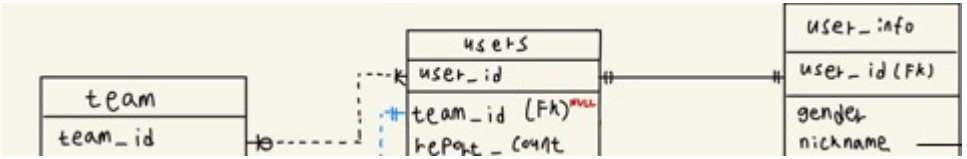
```
// 그룹 생성
1 usage  dlawjddn +1
public void makeTeam(User tempLeader, SaveTeamDTO teamInfo){
    User user = userService.findById(tempLeader.getId());
    if (!checkUserReport(user))
        throw new CustomException(ErrorCode.BANNED);
    List<User> members = new ArrayList<>(teamInfo.getNicknames().stream() Stream<String>
            .map(userInfoService::findByNickName).toList().stream() Stream<UserInfo>
            .map(userInfo -> userService.findById(userInfo.getId())).toList());
    members.add(tempLeader);

    log.info("팀 구성원 추출 완료");
    teamService.saveTeam(user, members, teamInfo.getTitle());
}
```

트리거란

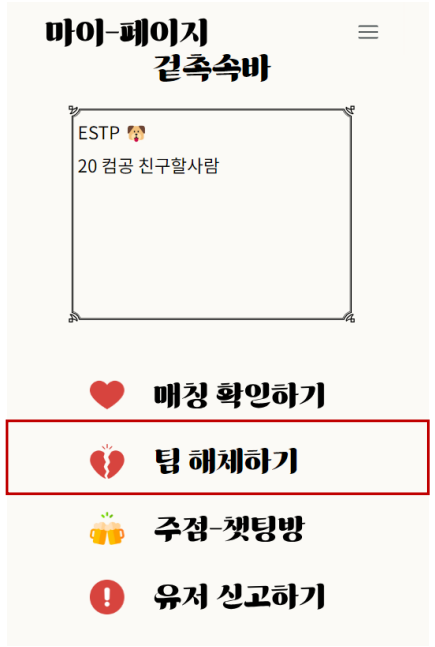
- 특정 이벤트의 발생에 따라 자동으로 작동하는 프로시저
- 이벤트 종류: 데이터 삽입, 수정, 삭제
- 이벤트에 따라 연쇄적으로 수행되어야 할 작업들을 자동화
  - Ex) 어떤 데이터가 삭제되면, 다른 데이터는 수정되어야 할 때

트리거 이용하기

- 우리 서비스가 제공하는 기능에 따르면, 팀 삭제, 미팅방 삭제의 처리를 하기 위해선, DB에 연쇄적인 데이터 입출력이 필요했음
- Ex) 팀 삭제



Client



팀 삭제 API 호출

Server

DataBase

팀 삭제 → 1

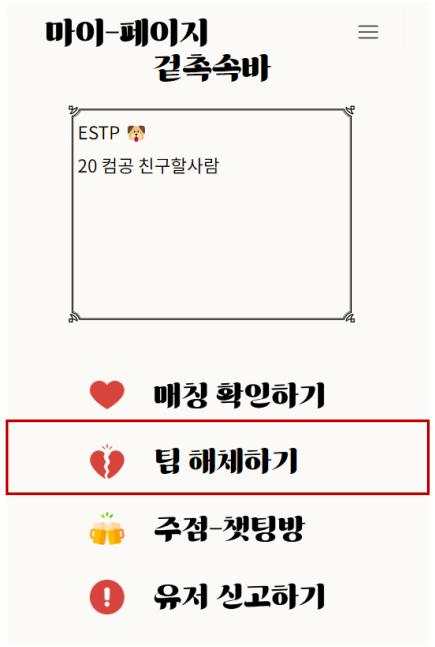
해당 user의, team_id를 NULL로.. → 1

단순히 팀을 삭제하고싶어도, users의 컬럼인 team_id에도 변경이 가해져야하므로, 쿼리문이 2개 필요함

트리거 이용하기

- 앞선 로직을 단 한번의 이벤트가 발생하면 자동으로 처리되게 해보자!
- before_team_delete 라는 트리거를 생성.
  - team테이블에 삭제 요청이 들어오면,
  - 그 전에 해당 team과 연결된 users테이블에 들어가서
  - 그 유저의 team_id를 NULL로 변경해준다!
  - 이후 팀 삭제

Client



팀 삭제 API 호출

Server

팀 삭제

1

DataBase

API 로직: "그저 지운다."

트리거 코드 before_team_delete

```
155  CREATE TRIGGER before_team_delete
156      BEFORE DELETE ON team
157      FOR EACH ROW
158  BEGIN
159      UPDATE users SET team_id = NULL WHERE team_id = OLD.team_id;
160  END;
```

트리거 이용하기

```
    public void deleteTeam(Team team) { teamRepository.deleteById(team.getId()); }


}
```

결과적으로

- 팀 생성 시 호출 쿼리 4-> 1
- 팀 삭제 시 호출 쿼리 2-> 1
- 미팅방 생성 시 호출 쿼리 5->1
- 미팅방 삭제 시 호출 쿼리 3->1

와 이거 효율 쩌는거 아니야???

…

잠 안오는김에 일단 짜놨는데...
이래도 되나?

단점은 뭐가있지??

오히려 더 혼란스럽게 만든거 아닌가?

나 혼자 이렇게 막 해도 되나

=> 찾아보자!!

저장 프로시저, 트리거 사용 시 단점

- 유지보수가 어렵다!!!!!!!!!!!
  - 규모가 크고, 기간이 긴 프로젝트에서 채택 안하는 가장 큰 이유
  - 데이터베이스 개발자가 짜놓고 도망갈 경우, 유지보수가 쉽지 않음
  - 배포 절차가 따로 없음->버전관리 어려움

- 이식성 문제
  - 프로시저와 트리거는 사용하는 데이터베이스에 종속적
  - 데이터베이스를 변경할 경우, 해당 데이터베이스에 맞게 다시 짜야함.

- 결합도 문제
  - 프로시저와 트리거는 데이터베이스와 강하게 결합되어있음.
  - 데이터베이스 내 테이블이나 컬럼에 변경이 생길 경우, 그에 대한 변경이 관련있는 모든 프로시저와 트리거에 영향.

- 테스팅 어려움
  - 서버단에서 테스팅을 진행할 때, 프로시저 혹은 트리거에서 발생하는 오류는 로그가 자세히 나오지 않음
  - 프로시저 혹은 트리거 내에서 모든 예외상황에 대한 자세한 예외처리가 필요

저장 프로시저, 트리거 사용 시 장점

- 네트워크 오버헤드 X
  - 서버와 데이터베이스 간의 통신 수를 효과적으로 줄임
  - 성능 최적화

- 데이터 처리 관련 수정이 발생한 경우, 배포가 간단해짐
  - 데이터베이스와 데이터 관련 로직의 결합도가 높아서 생기는 장점
  - 프로시저 or 트리거만 수정하여 배포하면 된다. 애플리케이션 코드의 배포 필요 X

- 데이터 무결성 강제
  - By 트리거
  - 데이터 무결성을 강제하므로, 서버단에서 무결성에 대한 별다른 고민 없이 로직 작성 가능

⇒ CRUD(Create,Read,Update,Delete) 기반의 애플리케이션,
⇒ 데이터의 무결성 보증이 중요한 애플리케이션,
⇒ 규모가 작고 기간이 짧은 프로젝트

GOOD!

이후엔..

API 명세작성 및 구현 완료

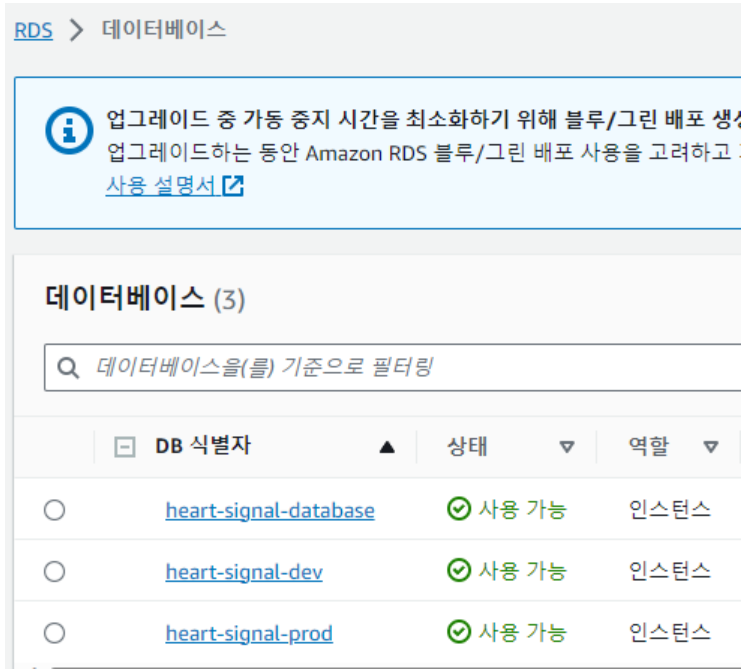
API 명세서

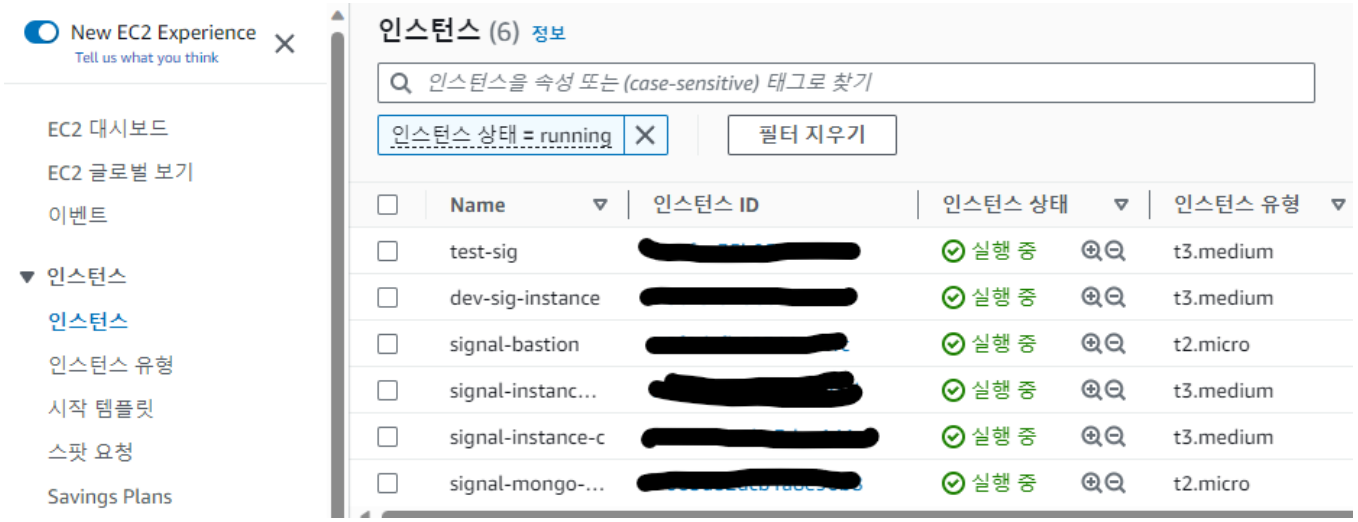
Aa 기능	Http 메...	API Endpoint	Status
카카오톡 로그인	POST	/oauth2/authorization/kakao	Done
닉네임 중복처리	GET	/api/v1/users/existed-nickname/{nickname}	Done
시작하기 열기 보)	POST	/api/v1/users/additional	Done
그룹화	POST	/api/v1/teams/building	Done
시그널 리스트 제공	GET	/api/v1/teams	Done
시그널 상세 정보 제공	GET	/api/v1/teams/{teamId}	Done
시그널 보내기	POST	/api/v1/teams/{teamID}/signal	Done
주점 채팅방 목록	GET	/api/v1/bars	Done
주점별 채팅방 입장	GET	/api/v1/chats/{barId}/chat?enterTime={enterTIme}	Done
매칭 확인하기	GET	/api/v1/teams/match/confirm	Done
마이페이지	GET	/api/v1/users/mypage	Done
시그널 거절하기	PATCH	/api/v1/teams/{teamID}/signal	Done
시그널 취소하기	DELETE	/api/v1/teams/{teamID}/signal	Done
유저 신고하기 → 닉네임 검색	GET	/api/v1/users/report/check-nickname/{nickname}	Done
유저 신고하기	PATCH	/api/v1/users/report	Done
메인 페이지	GET	/api/v1/users/main-page	Done
팀 삭제하기	DELETE	/api/v1/teams	Done
동성 닉네임 확인 (그룹화)	GET	/api/v1/users/same-gender/{nickname}	Done
미팅 채팅방 나가기 (삭제)	DELETE	/api/v1/chats/meeting-room/{Id}/chat	Done
미팅 채팅방 입장	GET	/api/v1/chats/meeting-room/chat	Done
미팅 채팅 소켓 전			Done

로컬 API 테스팅

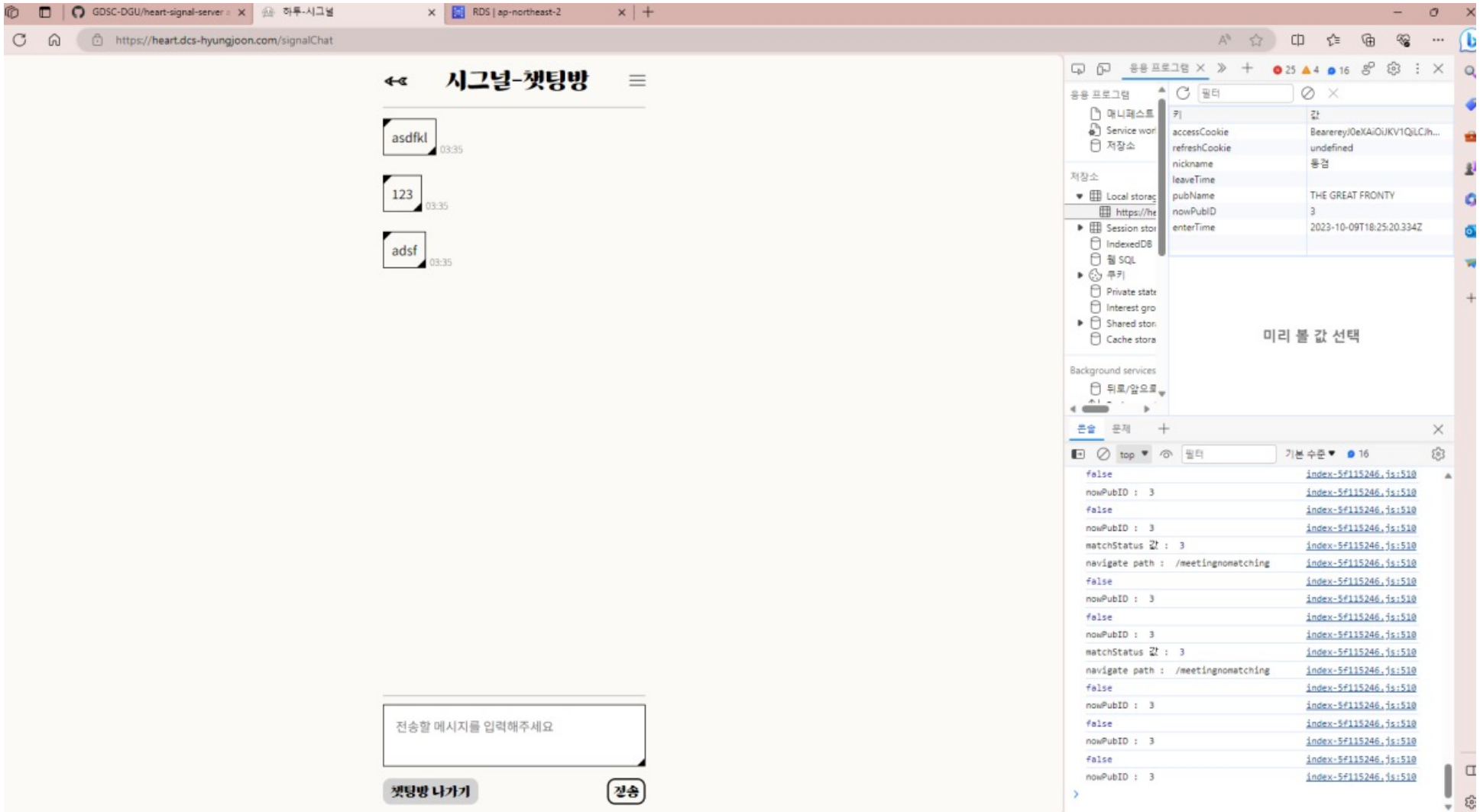
클라우드 DB 생성, 가상 서버 테스팅

- RDS 클라우드 DB 생성 후 셋팅
- EC2 인스턴스에 스프링 애플리케이션을 도커 컨테이너로 배포
- 해당 서버에서도 테스팅 성공적으로 수행





클라이언트 – 서버 통합테스팅



10/11 성공적인 배포


heartu-signal.com
하투-시그널
메인페이지
시그널-리스트
주점-리스트
마이-페이지
제작자들
인숫하그램
©GDSC HEART SIGNAL. All rights reserved.

감사합니다!

사실 원카드 중에 포착된 장면

속았죠?!