Team : 과제 물리치고 5조

Member : 윤수빈, 조웅상, 서리하, 최강현, 임동혁

풋살 온라인 프로젝트

목차

Content

- 기획 개요
- 프로젝트 설계
  - API 명세서
  - Entity Relationship Diagram (ERD)
  - Wire Frame
  - 게임 구동
- 시연
- Trouble Shooting

1-1. 기획 개요

- Futsal Manager
  - 확률을 기반으로한 풋살 시뮬레이터

- 기획 특징
  - WebClient를 통해 플레이 가능
  - 거리 기반 확률 : 선수들 간의 거리, 골대 거리를 기반으로 확률이 결정됨
  - 뽑기로 선수 획득

2-1. API 명세서

- [API 명세서 노션 링크]

API 명세서

담당(정)	담당(부)	진행상황	파트	Aa 기능	우선순위	method	URL
윤수빈	최강현	완료	계정	토큰 재발급	선택	POST	api/refreshToken
윤수빈	최강현	완료	계정	계정 생성	필수	POST	api/account/regist
윤수빈	최강현	완료	계정	계정 조회	필수	GET	api/account
윤수빈	최강현	완료	계정	계정 수정	필수	PATCH	api/account
윤수빈	최강현	완료	계정	계정 로그인	필수	POST	localhost:3333/api/account/login
최강현	서리하	완료	선수	선수 생성	필수	POST	localhost:3333/api/players
최강현	서리하	완료	선수	선수 목록 조회	필수	GET	localhost:3333/api/players
최강현	서리하	완료	선수	특정 선수 조회	필수	GET	localhost:3333/api/players/:playerName
최강현	서리하	완료	선수	선수 데이터 수정	선택	PUT	localhost:3333/api/players/:playerName
최강현	서리하	완료	선수	보유 선수 판매	필수	DELETE	localhost:3333/api/roasters
최강현	서리하	완료	선수	보유 선수 조회	필수	GET	localhost:3333/api/roasters
조웅상	임동혁	완료	덱	팀 편성 선수 추가	필수	POST	localhost:3333/team/add
조웅상	임동혁	완료	덱	팀 편성 선수 제외	필수	PUT	localhost:3333/team/exclude
조웅상	임동혁	완료	덱	팀 편성 선수 초기화	선택	PUT	localhost:3333/team/empty
조웅상	임동혁	완료	덱	팀 편성 선수 조회	필수	GET	localhost:3333/team/search/:accountId
조웅상	임동혁	기획 변경	덱	팀 포지션 변경	선택	POST	

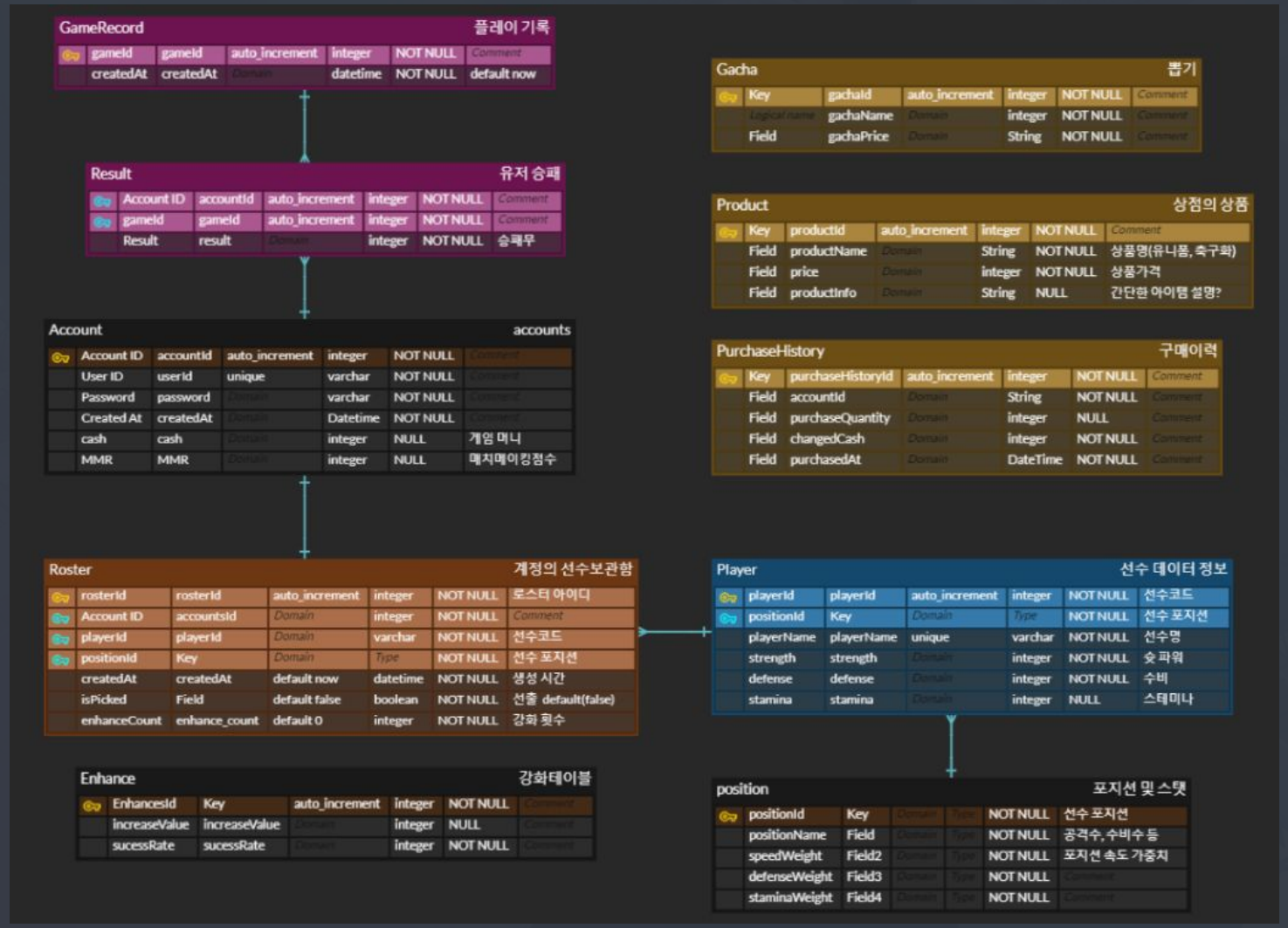
2-1. API 명세서

- [API 명세서 노션 링크]

임동혁	윤수빈	완료	매치	사용자 설정 게임	필수	POST	localhost:3333/api/custom
임동혁	윤수빈	완료	매치	경쟁전 시작	도전	GET	localhost:3333/api/match
임동혁	윤수빈	완료	매치	계정 랭킹 조회	도전	GET	localhost:3333/api/rank
임동혁	윤수빈	완료	매치	보유 선수 강화	도전	PATCH	
서리하	조응상	완료	상점	선수 단일 뽑기	필수	POST	localhost:3333/api/gacha/buy/:gachaTry
서리하	조응상	완료	상점	선수 다수 뽑기 열기	선택	POST	localhost:3333/api/gacha/buy/:gachaTry
서리하	조응상	완료	상점	상점 아이템 구매	필수	POST	localhost:3333/api/product/:productId
서리하	조응상	완료	상점	게임 머니 구매	필수	PUT	localhost:3333/api/account/cash
서리하	조응상	완료	상점	상점 아이템 전체조회	선택	GET	localhost:3333/api/product
서리하	조응상	완료	상점	상점 아이템 세부조회	선택	GET	localhost:3333/api/product/:productId
서리하	조응상	완료	상점	구매이력 생성	선택	POST	localhost:3333/api/product/purchasehistory
서리하	조응상	완료	상점	구매이력 조회	선택	GET	localhost:3333/api/product/purchasehistory
서리하	조응상	완료	상점	가챠상품 생성	필수	POST	localhost:3333/api/gacha
서리하	조응상	완료	상점	가챠상품 조회	선택	GET	localhost:3333/api/gacha

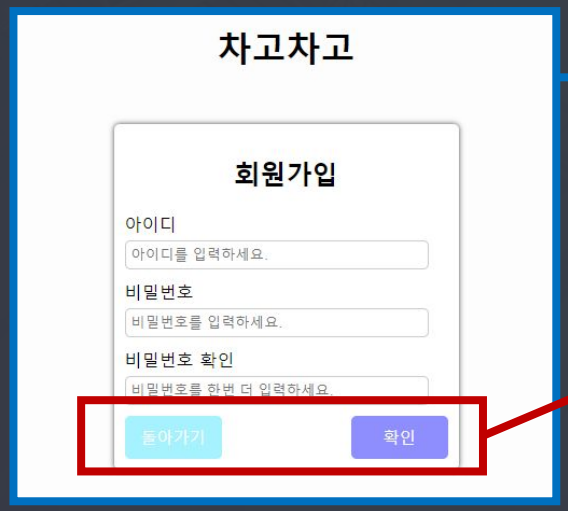
2-2. Entity Relation Diagram (ERD)

- ERD Cloud 사용



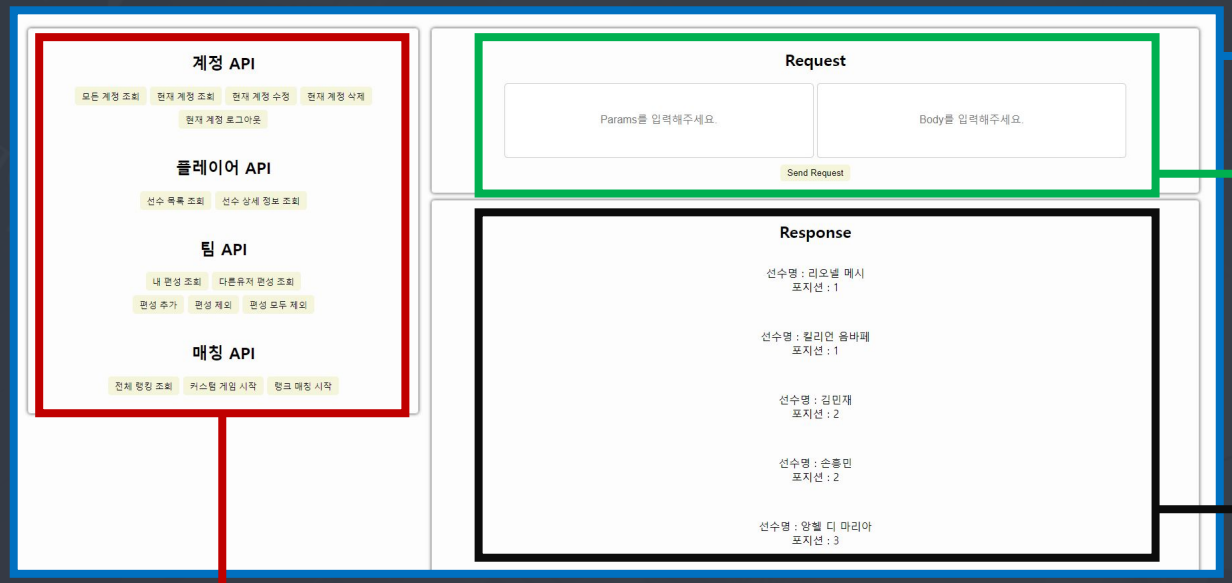
2-3. Wire Frame

- 경로 -> (domain)/
- 계정 생성 혹은 확인 버튼으로 로그인을 할 수 있습니다.



- 경로 -> (domain)/join
- 로그인 화면으로 돌아가거나 확인 버튼으로 계정 생성이 가능합니다.

2-2. Wire Frame



- 경로 -> (domain)/category
- 요청할 때 필요한 param과 body를 작성하는 공간입니다.

- 응답한 결과가 출력되는 공간입니다.
- 각 API의 응답에 따라 정제하여 출력할 수 있습니다.
- 게임 시작 시, 게임과 게임 상황이 출력됩니다.

- API 요청 카테고리 목록입니다.
- 각 API 버튼을 눌러 우측에 Send Request 버튼을 생성합니다.
- Send Request 버튼의 ID를 각 API 버튼 ID를 가져와 고유아이디를 붙여줍니다.

2-4. 게임 구동

1. 참여 유저 선수 정보를 획득
2. 선후공 선택 (코인토스)
3. 선공 팀의 최전방 선수에게 공 지급
4. [드리블, 패스, 슛] 3가지 선택지 중 가장 확률이 높은 행동을 선택.
   a. 드리블 : 상대 수비수 거리에 반비례해 확률이 달라짐.
   b. 패스 : 동료 거리에 반비례해 확률이 달라짐.
   c. 슛 : 골대 거리에 따라 반비례해 확률이 달라짐.
   d. 수비 : 상대 공격수 거리에 반비례해 확률이 달라짐.
5. 선택된 행동 성공을 판정.
6. 판정 결과를 적용. [수비 성공, 실책, 득점 실패] >> 스태미나 소모, 공 소유권 이동
7. 골 수로 승패를 판정하고 승리시 MMR을 획득.

서비스 주소 : http://soobin.store:3333/

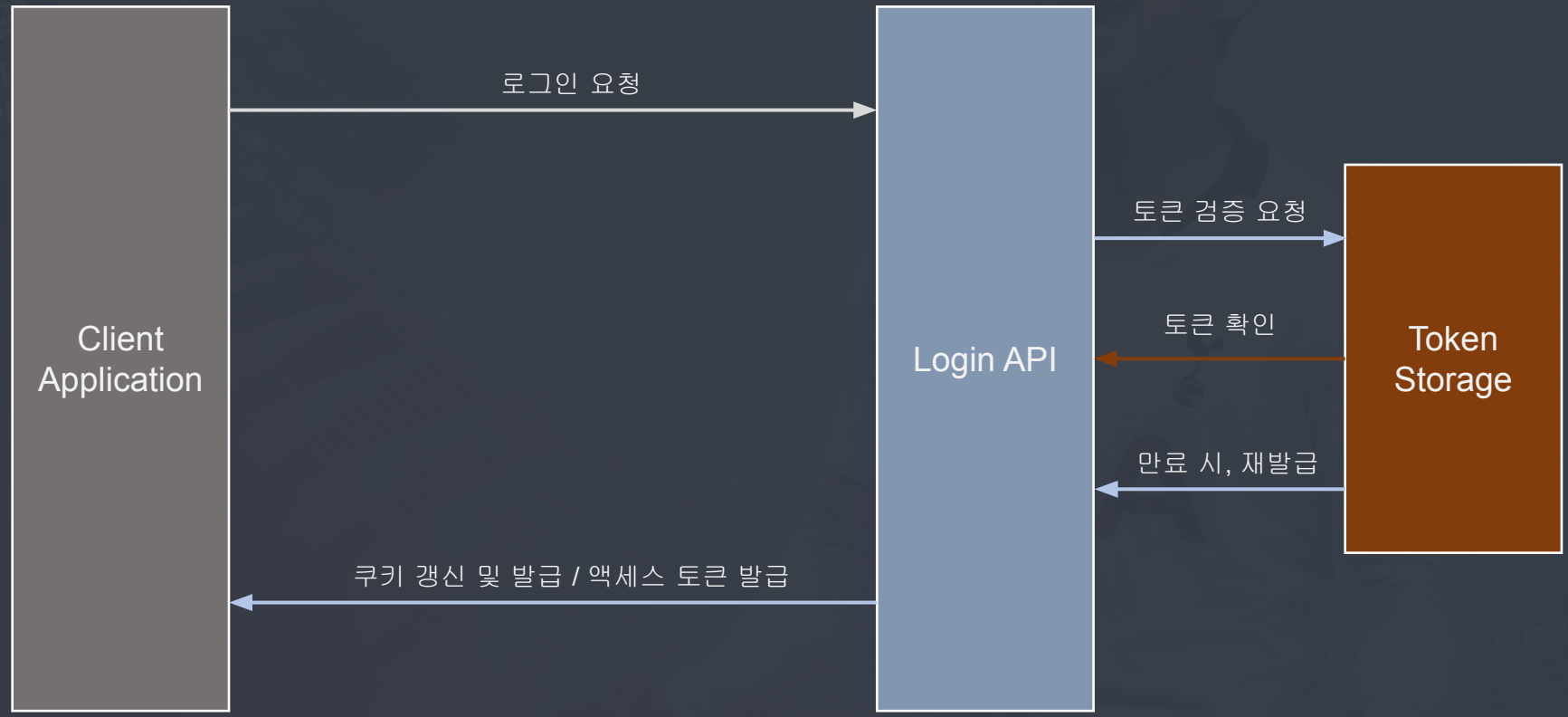
시연 영상 링크 :

4-1. Access Token & Refresh Token

- 로그인

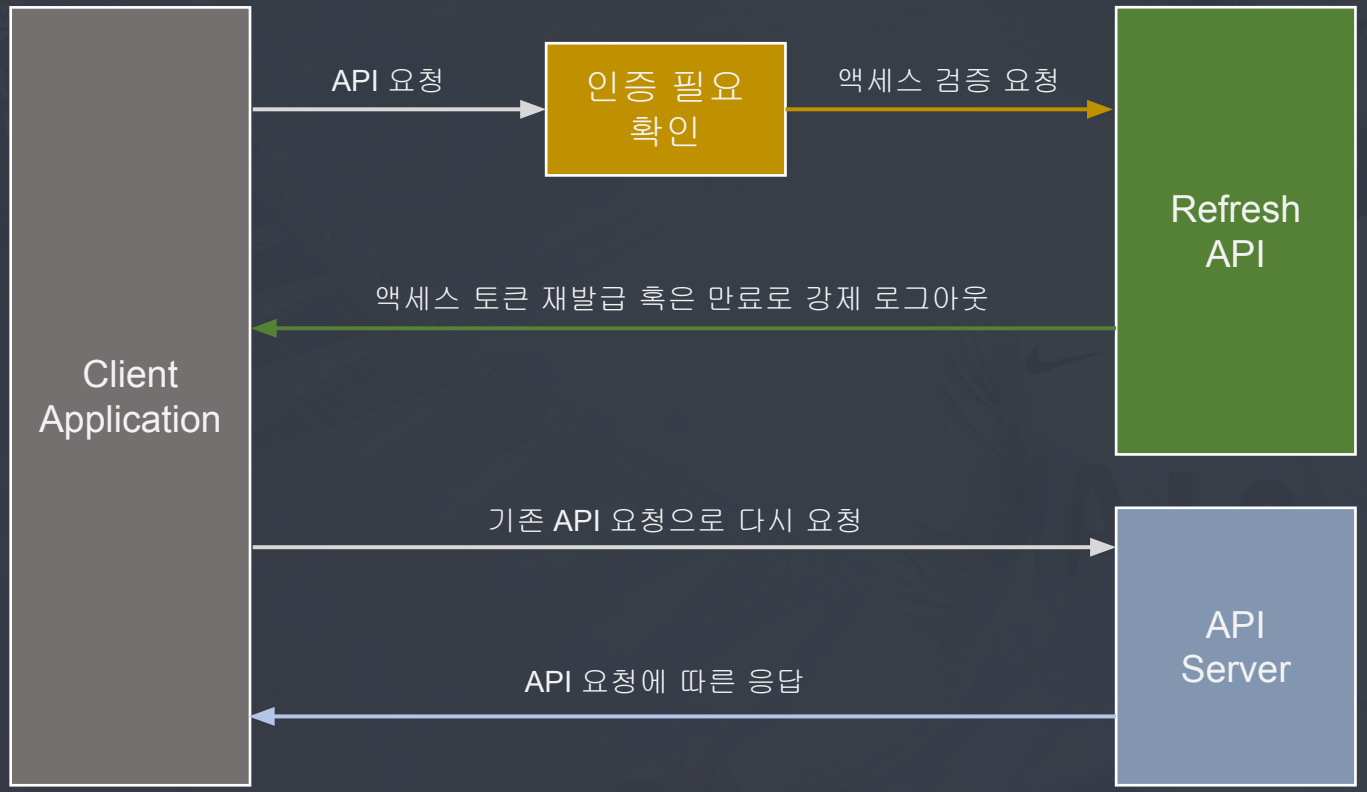
자세한 로그인 절차 및 코드는 아래 링크 참고

로그인 인증 절차



4-1. Access Token & Refresh Token

- API 요청



자세한 API 요청 절차 및 코드는 아래 링크 참고

[API 요청 인증 절차]

4-1. Access Token & Refresh Token

- 문제점
  - Access Token는 탈취 위험이 있어 보안이 미흡함.
  - 액세스 토큰과 리프레시 토큰의 저장 위치와 전달 방법에 대해 고민.
  - Refresh Token이 유효하지 않을 때 접근을 거부하지 못하는 문제.
  - 다른 브라우저로 이동해도 토큰이 유지되도록 고민함.
  - DB와 유저 쿠키 Refresh Token의 만료 기한이 일치하지 않는 문제.

- 해결 방안
  - XSS / CSRF 공격에 대응하여 최대한 Access Token의 생명주기를 짧게 함.
  - Refresh Token은 Access Token을 재발급하는 용도로만 사용하여 탈취 리스크를 제거.

    => Access Token은 메모리, Refresh Token은 쿠키로 발급.

    => 인증이 필요하면 액세스 토큰을 헤더로 전달하고 쿠키를 보내는 방향 채택.
  - 다른 브라우저 이동 전 로컬스토리지에 저장하고 이동 완료 후 메모리에 다시 저장 후 로컬 스토리지를 비워 브라우저 이동에 대응함.
  - 쿠키는 토큰 DB에 저장된 Refresh Token의 만료기한(exp)을 가져와서 현재 시간으로 계산하는 과정으로 만료를 확인.

4-2. 선수 호출 API

- 호출 빈도 높은 API 개선 문제
  - 보유한 선수 전체를 조회하는 API는 테이블 데이터가 많아 응답 속도 저하를 우려해 성능적으로 개선할 수 있는 방안에 대해 고민함.
  - prisma를 사용할 경우 값을 호출한 뒤 처리가 한번 더 필요했던 중복 선수 확인 코드를 개선함.

- 해결 방법
  - 해당 코드를 RawQuery로 작성하여 쿼리 연산으로 한번에 처리되도록 함.
  - player 테이블과 roster 테이블을 조인해 중복선수를 그룹핑해 중복 선수를 카운트한다.

개선 전

```
/** 보유 선수 조회 **/
// 강화수치가 같은 중복 선수는 거름!!
router.get('/roster', authMiddleware, async (req, res) => {
    const { accountId } = req.user;

    const myPlayers = await prisma.roster.findMany({
        where: {
            accountId: +accountId,
        },
        include: {
            player: true,
        },
        distinct: ['playerId', 'enhanceCount'],
    });

    if (myPlayers.length === 0) {
        return res.status(404).json({ message: '보유한 선수가 없습니다.' });
    }
```

개선 후

```
// 각 중복선수 카운트
const countPlayer = await prisma.$queryRaw`
   SELECT
    roster.rosterId,
    roster.playerId,
    roster.isPicked,
    roster.enhanceCount,
    player.positionId,
    player.playerName,
    player.playerStrength,
    player.playerDefense,
    player.playerStamina,
    COUNT(*) as playerQuantity
  FROM roster
  JOIN player ON roster.playerId = player.playerId
  WHERE roster.accountId = ${+accountId}
  GROUP BY  roster.isPicked , roster.playerId, roster.enhanceCount
  ORDER BY  roster.isPicked DESC, roster.playerId ASC, roster.enhanceCount DESC;
```

4-2. 게임 기록 테이블 구성

- 가챠 로직 구현 문제
  - create를 통해 뽑기 등장 목록을 등록하고 랜덤하게 뽑는 기능 구현에 어려움 있었음.
  - create를 for문을 통해 동작시키고 있어 DB 부하의 위험이 있었음.

- 해결 방법
  - 선수 DB에서 인덱스번호를 랜덤추출로 부여 로직 적용
  - createMany를 사용해서 로직을 개선함.

```
export default async function (gachaTry, accountId) {
    const answer = [];

    // 선수 목록 조회
    const players = await prisma.player.findMany({
        select: {
            playerId: true,
            playerName: true,
        },
    });

    for (let i = 0; i < gachaTry; i++) {
        let randomPlayer = Math.floor(Math.random() * players.length - 1) + 1;
        answer.push({
            playerId: players[randomPlayer].playerId,
            playerName: players[randomPlayer].playerName,
            accountId: +accountId,
        });
    }
```

```
const gachaTransaction = await prisma.$transaction(
    async tx => {
        const resultGacha = await gacha(gachaTry, accountId);
        const createGacha = resultGacha.map(({ playerName, ...rest }) => rest);
        const findName = resultGacha.map(({ playerName }) => playerName);

        const cashGo = await prisma.account.update({
            data: {
                cash: req.user.cash - gachaPrice,
            },
            where: {
                accountId: +accountId,
            },
        });
        // 위에서 뽑은 결과로 createMany
        await prisma.roster.createMany({
            data: createGacha,
        });

        return [cashGo, findName];
    },
```

4-3. 게임 기록 테이블 구성

- 게임 기록 테이블 구성 문제
  - 게임 기록 테이블에 참가한 유저들과 승패 결과를 기록하고자 함.
  - 게임 기록 테이블에 AccountId 라는 고유키가 여러번 들어갈 수 있게 하는 방법을 고민함.

- 해결 방법

  GameRecord 테이블과 Account 테이블 사이에 결과를 저장하는 테이블을 두는 것으로 1:N:M:1 형태를 구성함.