Bank Project

<1일차> 초기세팅

<2일차> DTO, Entity 작성

| **이름** | **역할** | **어디에 사용?** |
| --- | --- | --- |
| DTO(Data Transfer Onject) | 데이터 전달용 객체 | Controller ↔ Service |
| Entity | DB 테이블과 매핑되는 객체 | Repository, DB 저장용 |
| Message | API 응답 메시지 구조체 | Controller → Client |
|  |  |  |

Entity는 스키마를 따라가기 때문에 잘 수정되지 않는다. 가져올 데이터들이 정의되어 있음.

그 데이터를 가공하거나 response 내려주는거는 DTO 사용

**-실무에서 유지관리에서 중요한 부분임-**

스키마를 entity에 구성함

val accessToken: String?= null,

이런 표현은 초기값을 null로 지정함. ?가 붙어서 string이지만, null 가능해짐

사용하는 경우 : 로그인 안 했을 때, 나중에 값을 넣을 예정일 때 등등

* @ManyToOne(fetch = FetchType.*LAZY*)  
  @JoinColumn(name = "user\_ulid", nullable = false)  
  val user: User,
* **@ManyToOne**: Account가 여러 개일 수 있고, 하나의 User에 연결됨. 즉, **N:1 관계**.
* **fetch = FetchType.LAZY**: 지연 로딩 방식. 실제로 user 필드를 참조할 때까지 DB에서 데이터를 안 가져옴 → 성능 최적화.
* **@JoinColumn(name = "user\_ulid")**: 외래 키 컬럼 이름 지정. 실제 테이블에 user\_ulid라는 칼럼으로 저장됨.
* **user: User**: User 엔티티와 연결되어 있음. 이게 객체 지향적으로 연관 관계를 표현하는 방식임.
* @Column(name = "balance", nullable = false, precision = 15, scale = 2)  
  var balance : BigDecimal = BigDecimal.ZERO
* 금액(balance)을 저장. BigDecimal은 돈 계산할 때 부동소수점 오차가 없어서 사용함.
* **precision = 15, scale = 2**: 최대 자릿수 15자리 중 소수점 이하 2자리.
* var: 변경 가능. 즉, 계좌 금액이 바뀔 수 있음.