Bank Project

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

()와 {}의 차이

| **기호** | **의미** | **예시** |
| --- | --- | --- |
| () | **생성자 정의** (받아야 할 값들 명시) | class Person(val name: String) |
| {} | **클래스 본문** (속성, 메서드 정의) | class Person { val name = "Unknown" } |

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.
* Val : 읽기 전용, var : 읽고 쓰기 전용

Enum class는 상수들을 고정된 집합으로 정의함. 요일, 방향, 상태, 에러 코드 등 사용

로깅 : 프로그램의 실행 중 일어나는 정보(이벤트, 오류, 상태 등)를 텍스트로 기록(log)하는 행위

로깅하는 이유 : 디버깅과 문제추적, 운영 호나경에서의 모니터링, 성능 분석, 보안 감시

AOP(Aspect-Oriented Programming) : 공통적인 기능(로깅, 트랜잭션, 보안 검사)을 핵심 비즈니스 로직과 분리하여 코드 중복을 줄이고 모듈화하는 프로그래밍 방식

디펜던시 인젝션(Dependency Injection) : Class가 직접 필요한 의존 객체를 만드는 것이 아닌, 외부에서 주입 받는 방식

사용 시 장점

* 느슨한 결합으로 클래스 간 의존성이 줄어듦
* 테스트가 용이함.
* 유지보수가 편리하고 코드 재사용률이 증가

| **요소** | **설명** |
| --- | --- |
| @Component | Spring Bean으로 등록됨 |
| OkHttpClient | HTTP 요청용 클라이언트, DI로 주입 |
| GET() | GET 요청 보내기, 헤더 포함 가능 |
| POST() | POST 요청 보내기, body 포함 가능 |
| resultHandler() | 응답 처리: 성공이면 내용 반환, 실패면 CustomException 발생 |
| **주의** | isSuccessful 체크는 잘못된 방향일 가능성 높음 (확인 필요) |

**싱글톤 패턴** : 특정 클래스의 인스턴스를 1개만 생성되는 것을 보장해줌.

Kotlin에서 object 키워드 사용해서 싱글톤 객체 생성

Json 인스턴스를 한 번만 생성하고 계속 재사용 가능

jsonUtil 같이 순수 기능(encode, decode)만 제공하는 클래스에 전역에서 바로 접근 가능한 형태가 됨. -> 훨씬 유용하고 병렬 환경에서도 스레드가 안전함.