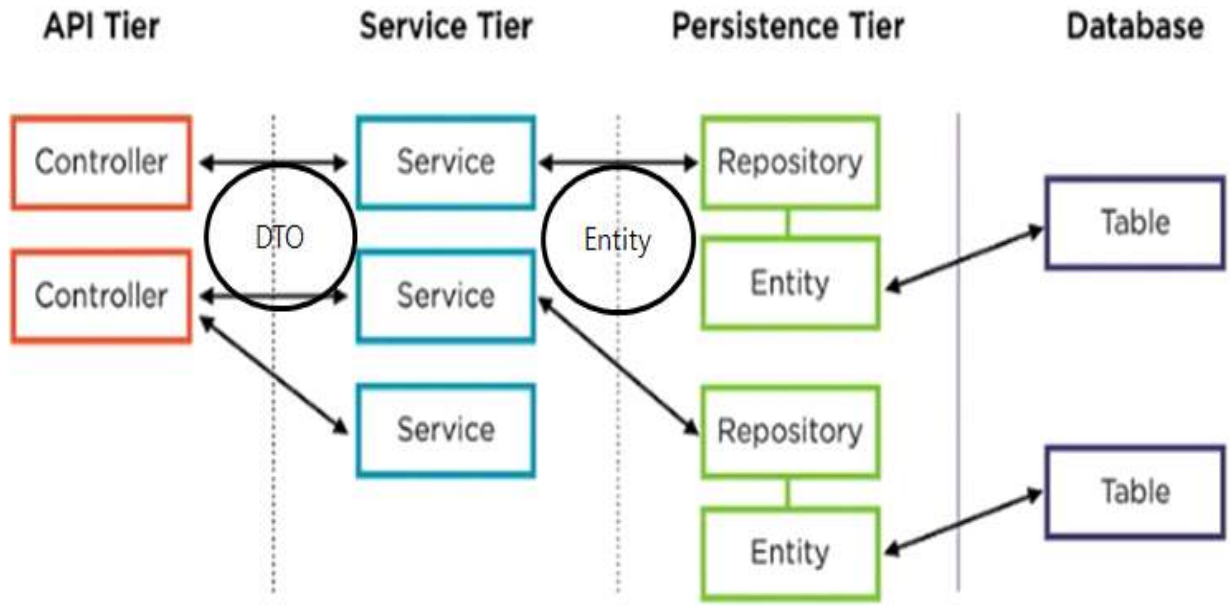
DTO & Entity 매핑



이미지 출처 : https://blog.devgenius.io/spring-data-jpa-c698d07a2a2b

1. DTO(Data Transfer Object)와 엔티티(Entity) 매핑

- DTO와 엔티티를 매핑하는 것은 보안, 성능, 유연성, 그리고 애플리케이션의 안정성을 도모하기 위해 필수적인 작업이다. 특히, 대규모 애플리케이션에서는 이러한 분리가 시스템의 복잡성을 관리하고 미래의 확장성 및 유지보수성을 보장하는 데 중요한 역할을 한다.

1) 보안

노출 제한 : 엔티티에는 비밀번호, 사용자 ID 등과 같이 외부에 노출되어서는 안 되는 정보가 포함될 수 있다. DTO를 사용하면 클라이언트에 반환되는 데이터를 제어할 수 있으므로, 민감한 정보의 노출을 방지할 수 있다.

2) 성능 최적화

필요한 데이터만 전송 : 엔티티 객체는 데이터베이스 테이블의 구조를 반영하기 때문에 필요 이상의 데이터를 포함할 수 있다. DTO를 통해 필요한 데이터만 클라이언트에 전송하면 네트워크 사용량을 줄이고, 성능을 개선할 수 있다.

지연 로딩 문제 해결 : 엔티티의 지연 로딩 속성으로 인해 필요하지 않은 데이터까지 로딩되어 성능 저하가 발생할 수 있다. DTO를 사용하면 이러한 문제를 피할 수 있다.

3) 유연성

레이어 간의 역할 분리 : DTO는 계층 간 데이터 교환을 위한 객체로, 프레젠테이션 계층과 비즈니스 로직 사이의 계약 역할을 한다. 엔티티와 DTO를 분리함으로써 각 계층의 역할을 명확히 분리하고 시스템의 유연성을 높일 수 있다.

API 스펙 변경의 용이성 : 클라이언트에게 노출되는 API 스펙을 변경할 필요가 있을 때, 엔티티 구조를 변경하지 않고도 DTO를 수정하는 것으로 충분하다. 이는 API와 데이터베이스 사이의 결합도를 낮추어 유지 보수성을 향상시킨다.

4) 애플리케이션의 안정성

데이터 무결성 보호 : 직접 엔티티를 클라이언트에게 노출하면 클라이언트가 엔티티의 상태를 예상치 못하게 변경할 위험이 있다. DTO를 통해 데이터를 주고받으면 이러한 위험을 줄이고 애플리케이션의 안정성을 보호할 수 있다.

2. 구현방법

- DTO와 엔티티의 매핑 방법은 여러 가지가 있으며 프로젝트의 요구 사항과 선호하는 코딩 스타일에 따라 적절한 방법을 선택할 수 있다. 생성자나 팩토리 메서드는 추가 라이브러리 없이 사용할 수 있는 반면 ModelMapper나 MapStruct는 라이브러리를 사용해야 한다.

1) 생성자를 통한 매핑

- DTO 클래스에 엔티티를 매개변수로 받는 생성자를 정의하여 엔티티 객체로부터 DTO 객체를 생성할 수 있다.

[예시]

```
public class BoardDTO {
    private Long id;
    private String title;
    private String content;

    public BoardDTO (Board board) {
        this.id = board.getId();
        this.title = board.getTitle();
        this.content = board.getContent();
    }

}
```

2) 정적 팩토리 메서드를 통한 매핑

- 클래스 내에 엔티티 객체를 매개변수로 받는 정적 팩토리 메서드를 정의하여 해당 메서드를 호출함으로써 매핑할 수 있다.

[예시]

```
public class BoardMapper {

    public static BoardDTO toDTO(Board board) {
        BoardDTO dto = new BoardDTO();
        dto.id = board.getId();
        dto.title = board.getTitle();
        dto.content = board.getContent();
        return dto;
    }

}
```

3) ModelMapper 라이브러리 사용

- ModelMapper는 자동으로 DTO와 Entity 사이의 매핑을 처리해주는 유용한 라이브러리이다.

[예시]

```
ModelMapper modelMapper = new ModelMapper();
BoardDTO boardDTO = modelMapper.map(board, BoardDTO.class);
```

4) MapStruct 사용

- MapStruct는 컴파일 타임에 동작하는, 타입 안전한 빈 매핑 라이브러리이다. MapStruct를 사용하면 매핑 로직을 인터페이스로 정의할 수 있으며, 구현체는 자동으로 생성된다.

[예시]

```
@Mapper
public interface BoardMapper {
    BoardMapper INSTANCE = Mappers.getMapper(BoardMapper.class);

    BoardDTO boardToBoardDto(Board board);
    Board boardDtoToBoard(BoardDTO boardDTO);
}
```