**[5회차 월말평가 - 관통 PJT]**

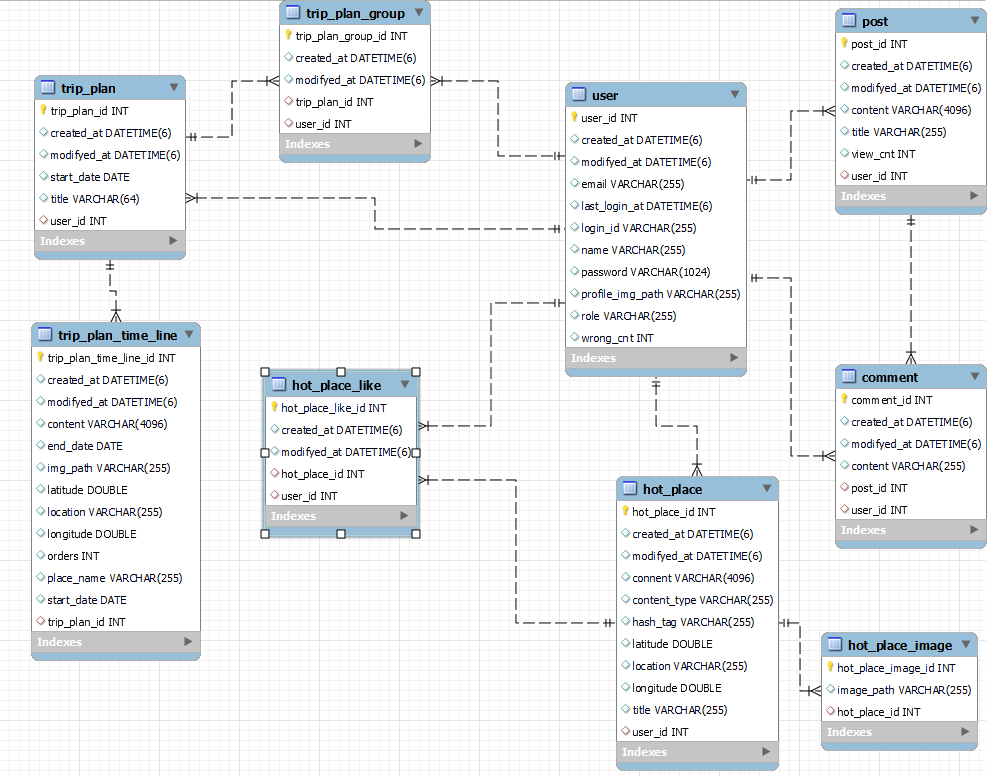
서울\_12반\_3조

팀장: 김한성, 팀원: 이정민  
gitlab 주소: https://lab.ssafy.com/anjfkrhdkdle/plan-t\_final-12-3

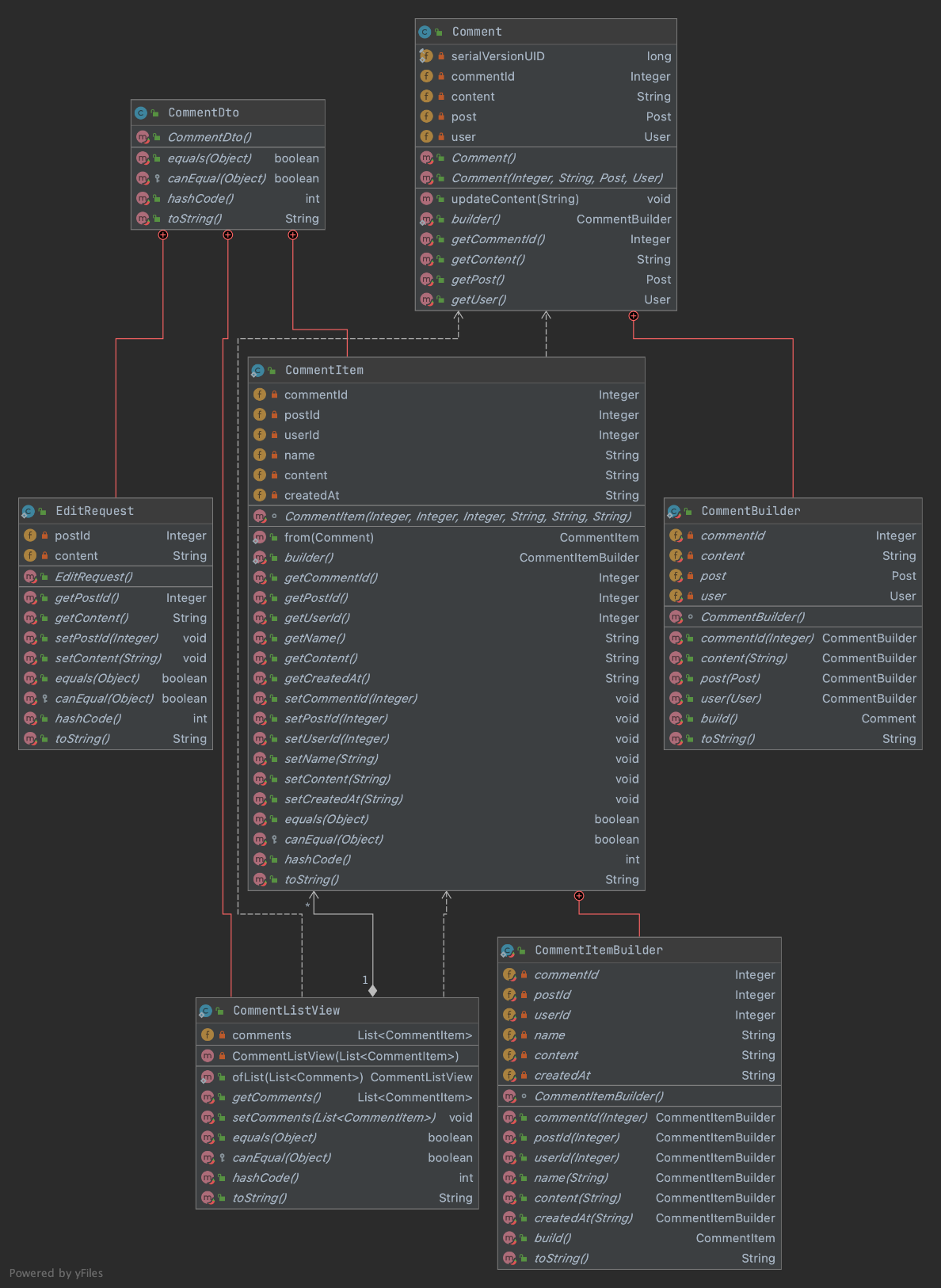
1) 요구사항 정의서

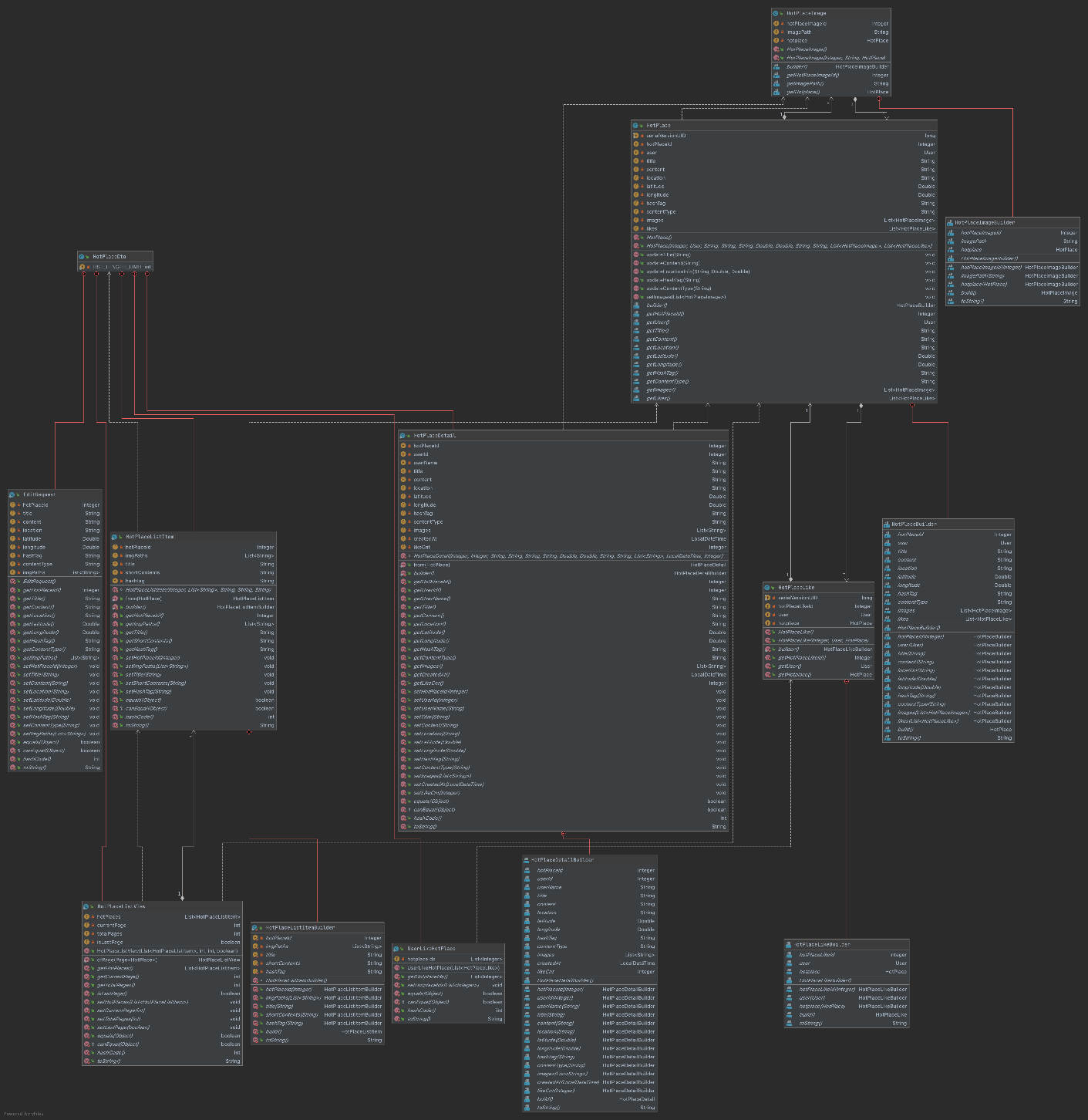
|  |  |  |  |
| --- | --- | --- | --- |
| **순번** | **요구사항명** | **요구사항 상세** | **우선순위** |
| **기능적 요구사항** | | | |
| B01 | 회원관리 | 회원가입 / 프로필 변경/ 별명 변경 | 필수 |
| B02 | 사용자 로그인 | JWS을 이용한 로그인 로그아웃 | 필수 |
| B03 | 사용자 아이디 찾기 | SMTP를 이용한 이메일을 통한 사용자 아이디 찾기 | 추가 |
| B04 | 사용자 임시 비밀번호 발급 | SMTP를 이용한 이메일을 통한 사용자 임시 비밀번호 발급 | 추가 |
| B05 | 여행 계획 관리 | 여행 계획 등록, 수정, 삭제, 조회 TMAP API 활용, 위도, 경도 정보를 수집이 가능해야함, 계획별 순서 설정 가능해야함 | 추가 |
| B06 | 여행 계획 권한 부여 | 다른 사용자에게 계획 수정,삭제 권한 부여 | 추가 |
| B07 | 여행 계획 복사 | 여행 계획을 복사해서 생성 가능해야함 | 추가 |
| B08 | 공유 게시판 | 게시글 등록, 수정, 삭제 조회  페이지 네이션이 가능해야함, 조건에 따른 검색이 가능해야함 | 필수 |
| B09 | 공유 게시판  덧글 관리 | 덧글 등록, 수정, 삭제, 조회 | 필수 |
| B10 | 핫플레이스 관리 | 핫플레이스 등록, 수정, 삭제, 조회 복수개의 이미지 업로드 가능, 카카오맵 활용, 지도상의 위치 등록이 가능해야함 위도, 경도, 데이터 타입을 수집해야함 | 심화 |
| B11 | 핫플레이스 좋아요 | 1개의 핫플레이스에 대해 사용자는 1개의 좋아요를 선택해서 볼수 있어야함 | 심화 |
| F01 | HOT PLACE 페이지 | 리스트 조회, 상세조회, 등록, 수정 페이지 | 심화 |
| F02 | TRIP PLAN 페이지 | 여행계획 등록, 수정, 상세조회 페이지 | 추가 |
| F03 | 자유 게시판 페이지 | 게시글 등록, 수정, 삭제 및 덧글이 추가 가능한 게시판 페이지 구현 | 필수 |
| F04 | MY PAGE | 프로필 수정 / 스크랩한 게시글 조회 | 심화 |

2) 테이블 구조도

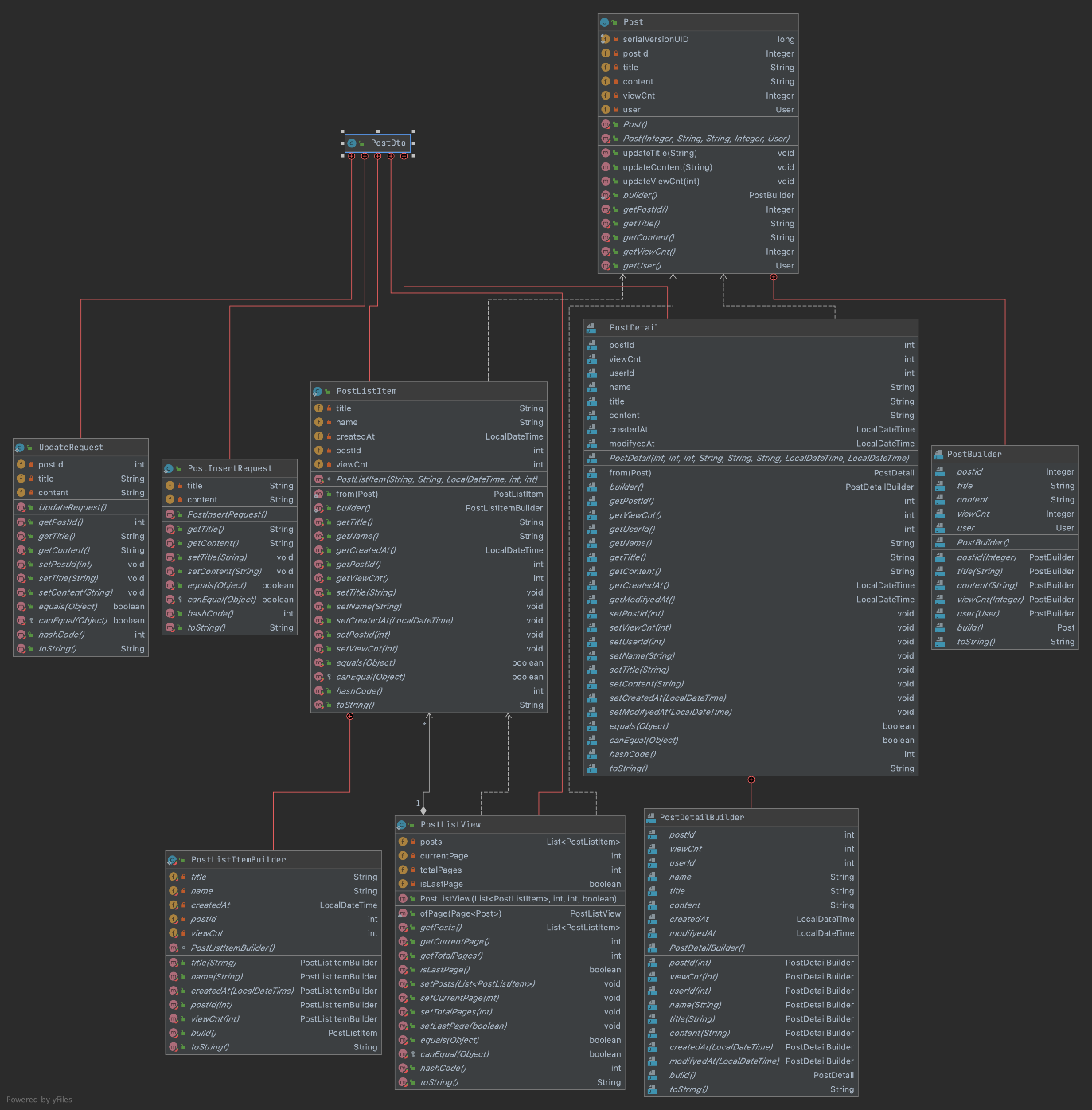


3) CLASS DIAGRAM – (데이터 설계)   
1)덧글

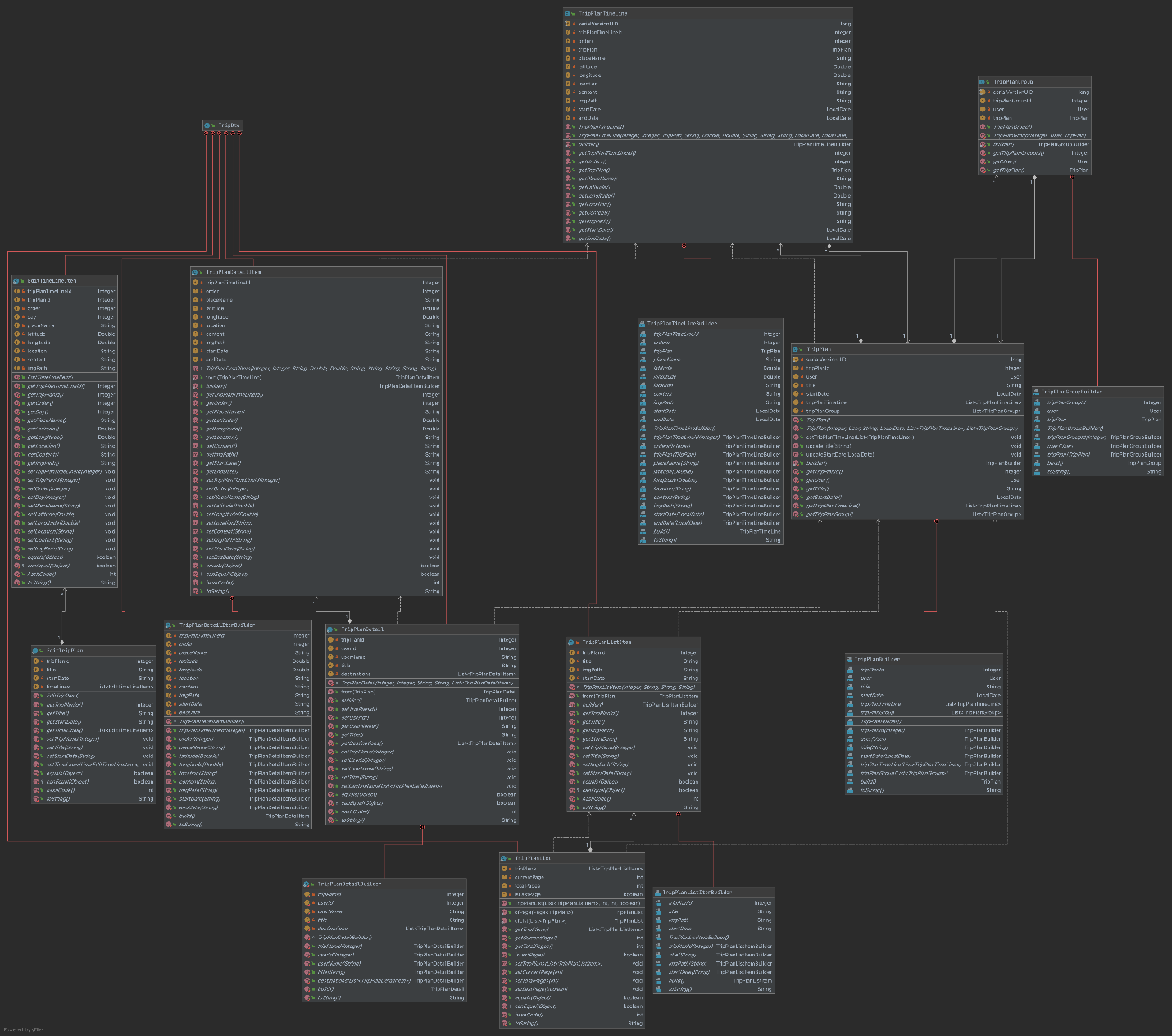


2)핫플레이스

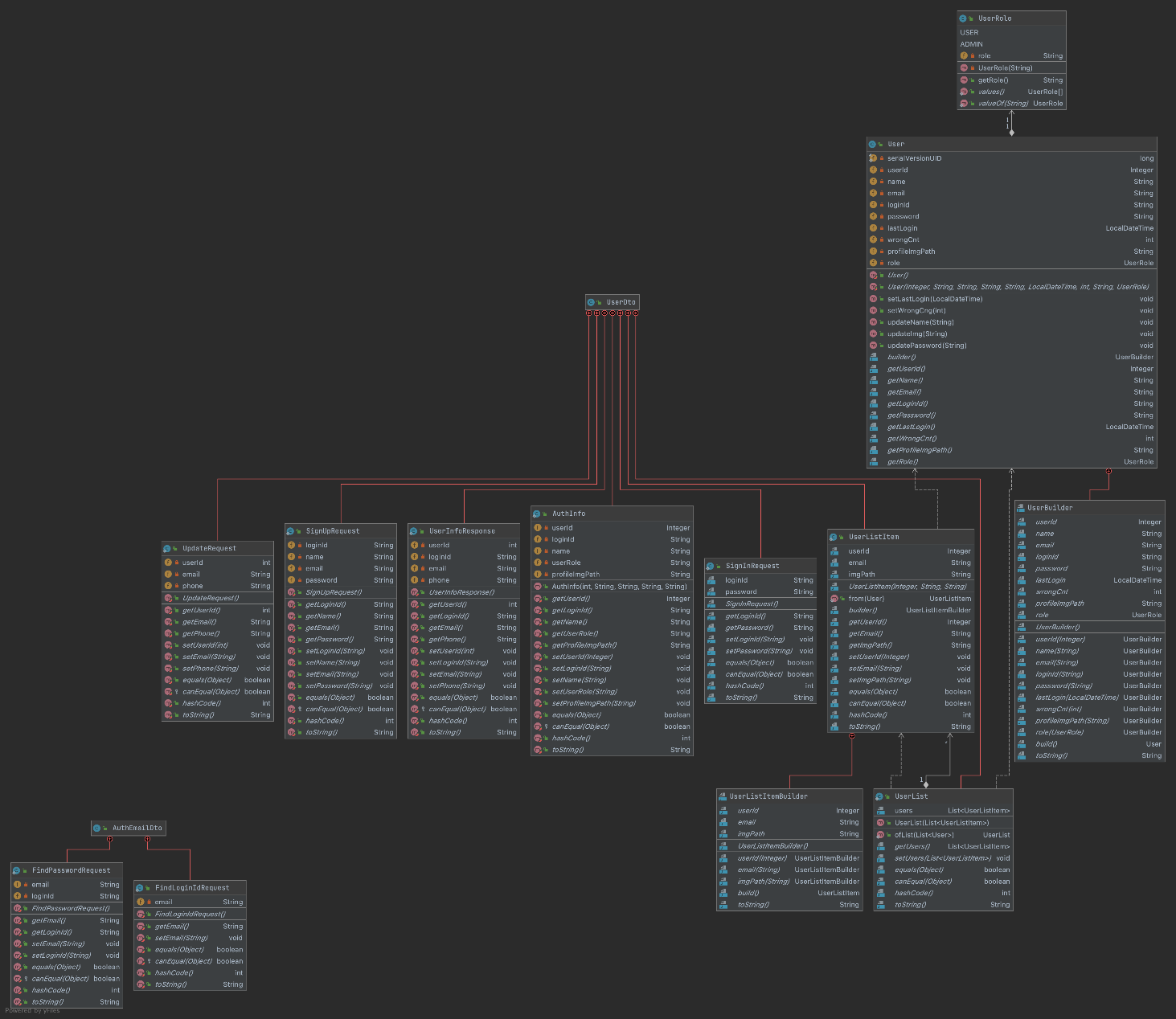
3) 게시글



4) 여행 계획

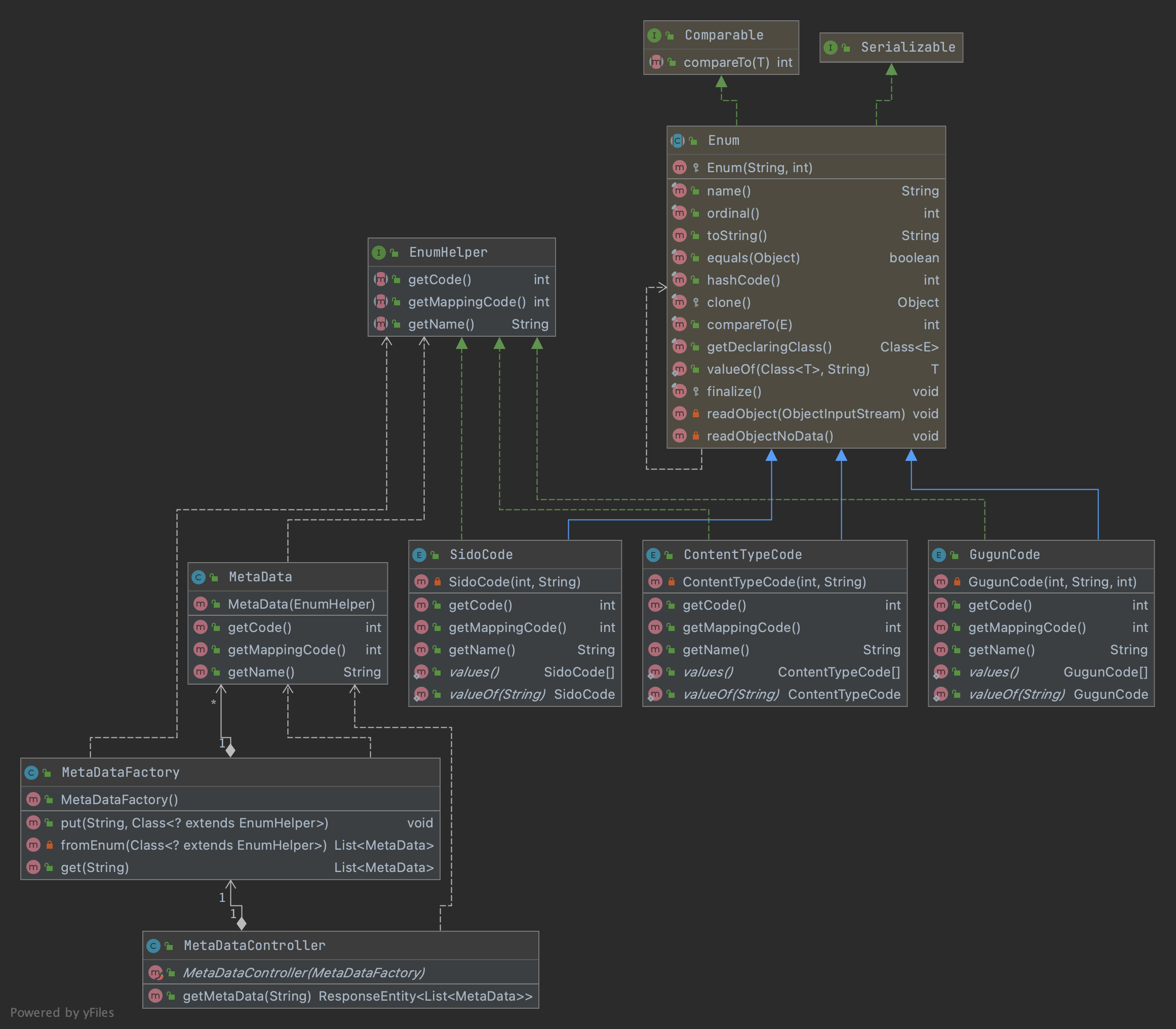


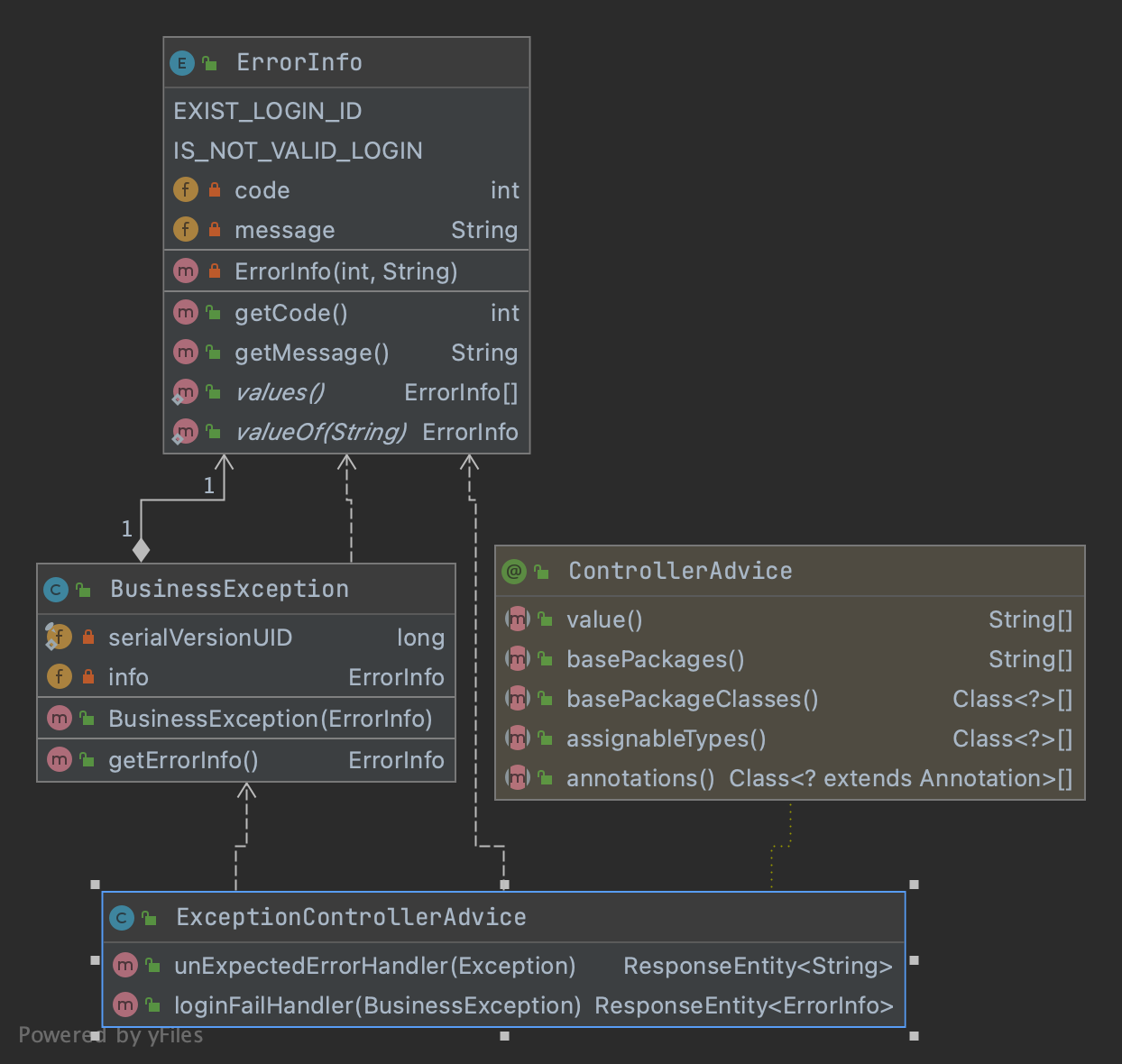
5) 사용자

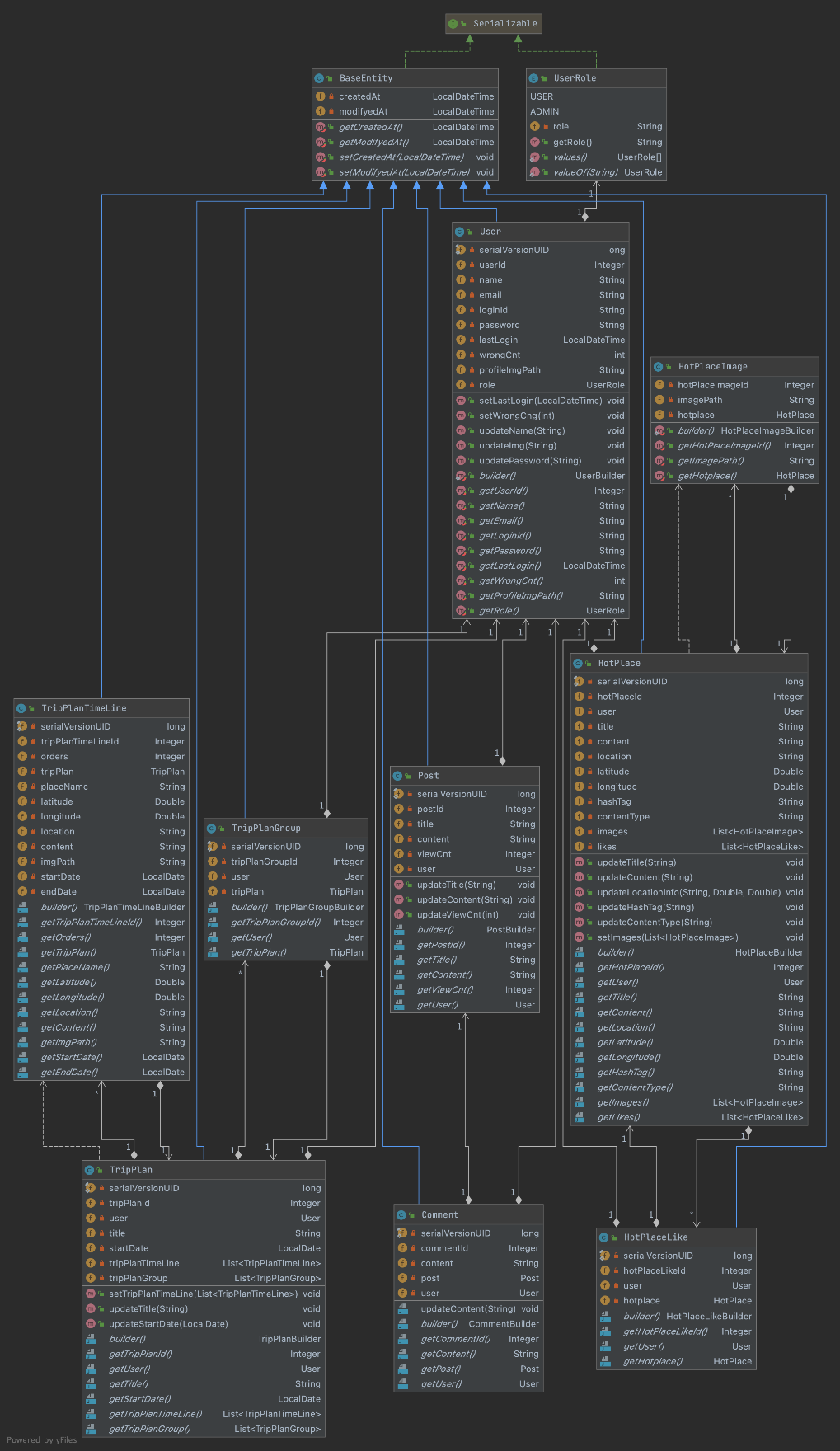


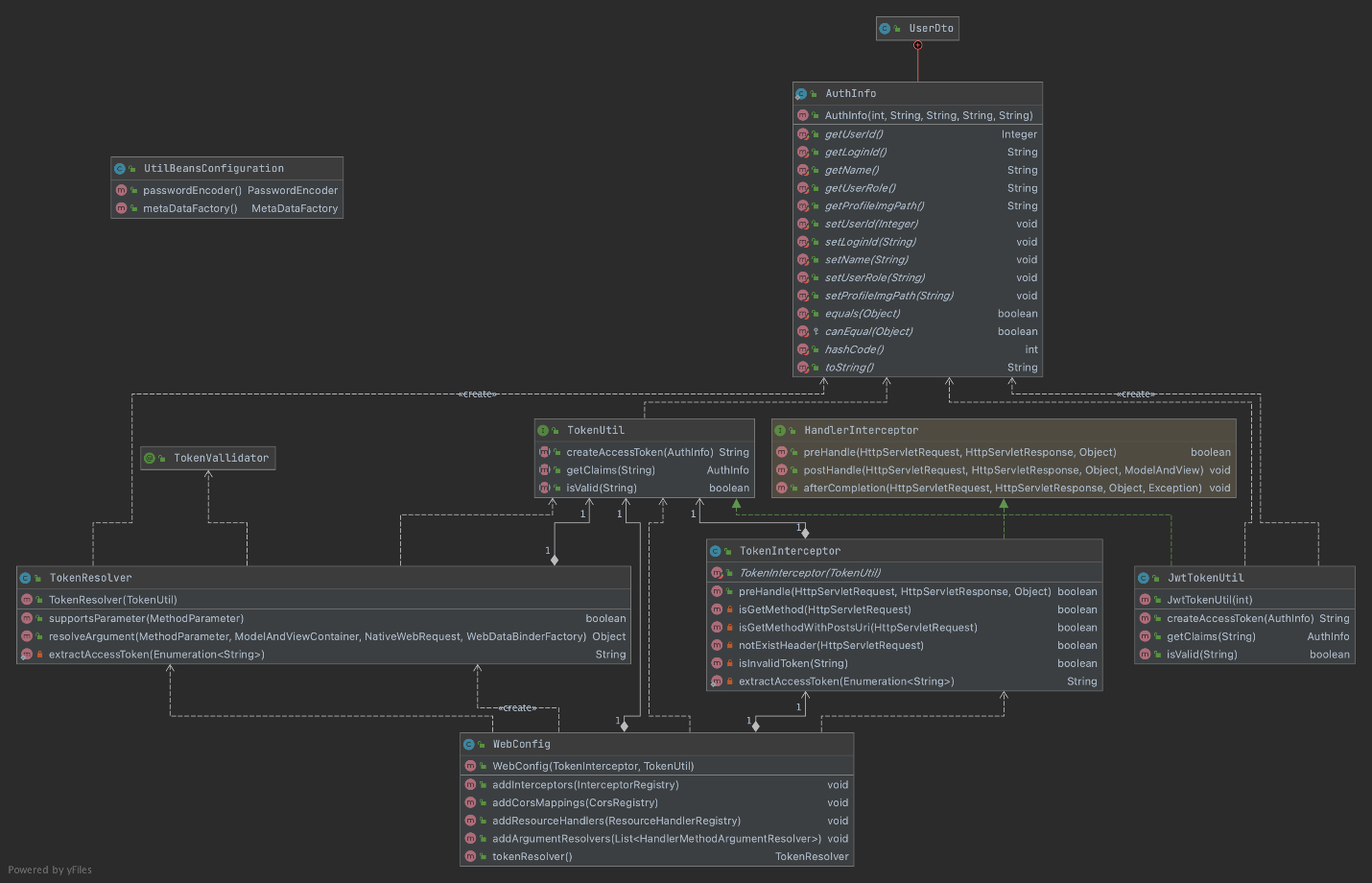
3) CLASS DIAGRAM – (공통 로직 설계)

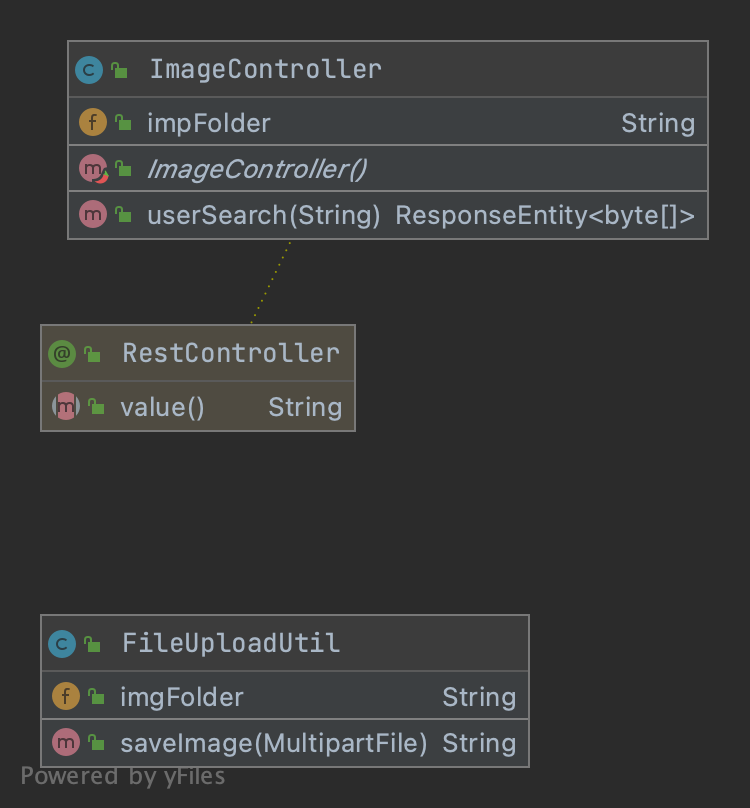
1) 메타데이터 조회 (시도/군구/컨텐츠 코드)



2) 에러 처리  
  


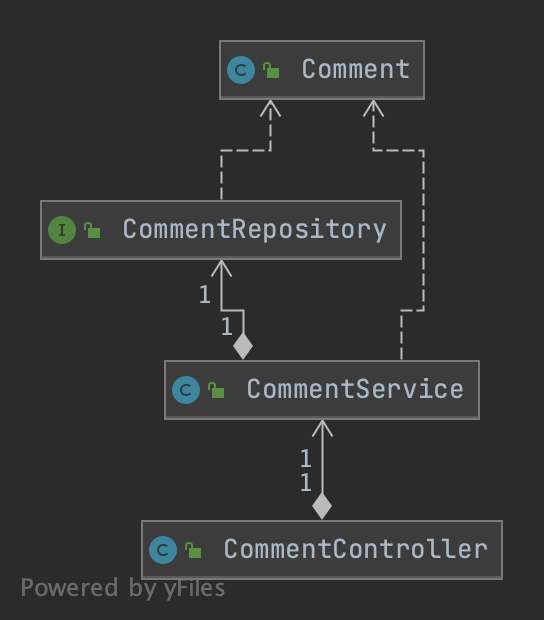
3) 엔티티 클래스간 의존성   


4) 인증 인가 관련  


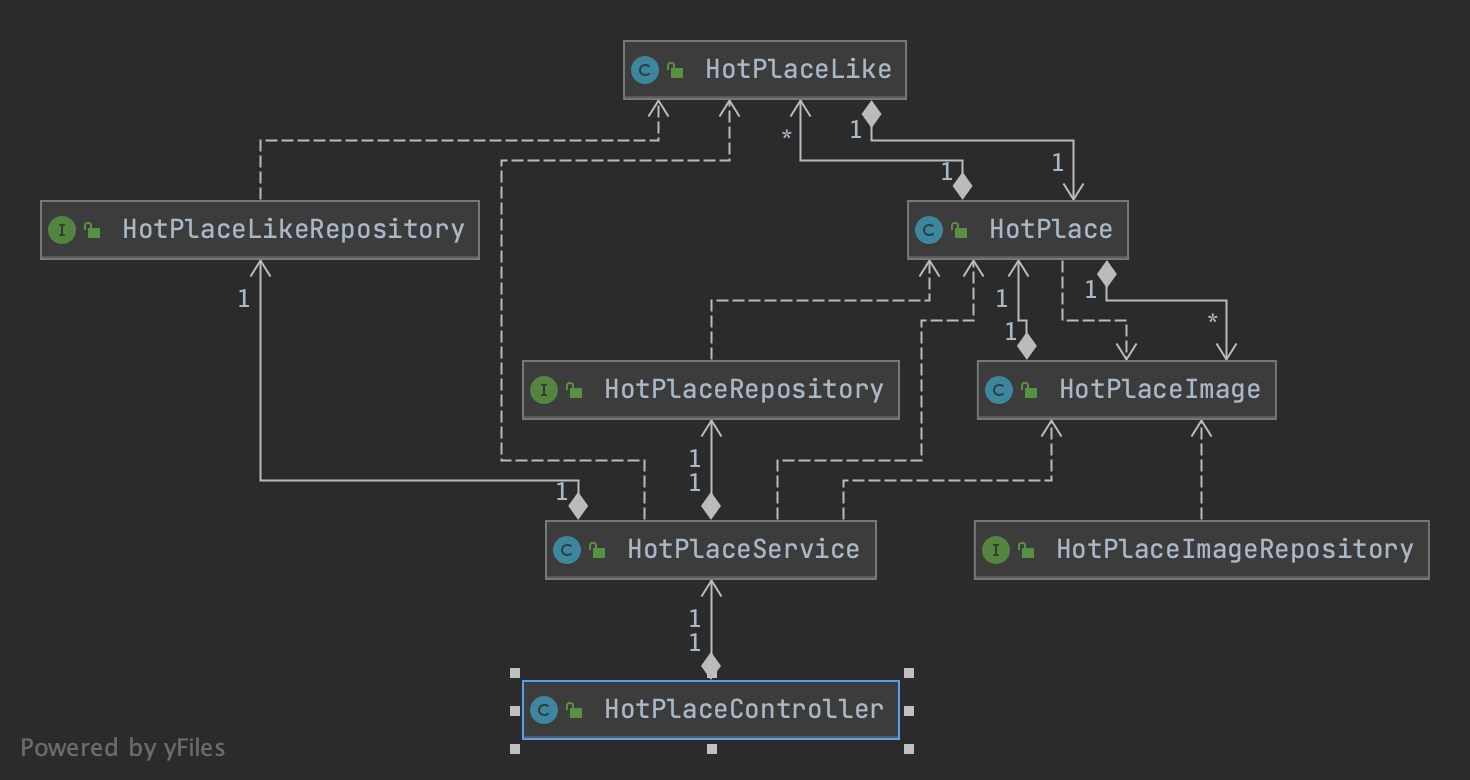
5) 파일 이미지 처리   


## 

3) CLASS DIAGRAM – (MVC 패턴)  
1) 덧글

****

2) 핫플레이스 관련



## 3) 게시글 관련 4) 여행 계획

## 5) 사용자

## 설계 관련 DTO 관리

Entity가 많아지고, 그 관계가 복잡할 수록 단순 DTO의 네이밍만으로 어떤 도메인과 연관되있는지 파악하기 어렵다 생각했습니다.

따라서 도메인과 관련된 하나의 큰 DTO로 묶고, inner class로 관리하여 응집성을 높이고자 했습니다.

**시군구, 컨텐츠 타입 코드**

사실상 거의 변화하지 않은 코드인데 매번 DB조회는 효율적이지 않다고 판단하여 별도의 enum으로 관리하였습니다.

그러나 군구 코드의 경우 개수가 상당히 많아, DB로 관리하는 것도 좋았다고 생각이 듭니다.

**인증 관련**

토큰 관련하여 Argument Resolver와 Intercepter를 두어 매번 컨트롤러에서 validation없이 한 곳에서 코드가 관리되도록 구성하였습니다.

**interface 관련**

여행 관련 어플리케이션에서 요구사항이 변동된다면 인터페이스 설계 자체가 바뀌어야하는 경우가 대부분일 것이고,

여러 구현체가 필요치 않다고 판단했습니다. 따라서 인증 인가를 제외하고는 별도의 인터페이스를 두지 않고 클래스를 두었습니다.

**예외 코드 관련**

비지니스 오류의 경우 별도의 서버 오류로 처리해야되기 때문에 에러코드를 관리하는 Enum객체와 이를 가지는 비지니스 Exception은 하나만 두어

예외관련 불필요한 클래스가 너무 많아지는 것을 방지하고자 했습니다. 또한 ExceptionAdvice를 통해 에러 처리를 한 곳에서 관리하고자 하였습니다.

**객체 Mapping**

Entity <-> 도메인간의 맵핑 로직이 Service단에 너무 많아지는 경우, 서비스의 핵심로직을 판단하는것을 방해한다 생각이 들었습니다.

따라서 이는 DTO에 팩토리 패턴을 적용하고 책임을 DTO에 두어, 서비스 레이어에서는 최대한 비지니스 로직만 남기려고 노력하였습니다.