MileStone 1

2017030055 윤상현

Design

구현환경은 java 언어를 사용한 Spring Boot 프레임워크를 사용한 back-end 구현과, JPA, Spring data JPA 를 통한 데이터베이스 관리, thymeleaf를 사용하여 front-end를 구현할 예정입니다.

전체적인 틀은 MVC 방식으로, Controller에서 Model을 Manipulate 하여 View 반영하는 방식을 가지고 있습니다. 프로젝트는 크게 6개의 파트로 나누어져 있습니다. 각 테이블을 Class 로 정의한 domain 파트, 데이터베이스에 접근하고 관리하게 되는 Repository 파트, html과 같은 front-end 파트, 그러한 front-end 파트와 최상단에서 교신하는 Controller 파트, 그리고 데이터베이스와 데이터를 주고 받을 때 데이터를 저장할 DTO(data translate object), 상세한 기능을 구현할 Service 파트로 구분하였습니다.

Spring boot 에선 JPA라는 툴을 사용하여 Class 객체가 table, 각 Class들의 엔티티 객체들이 튜플에 대응하여 자동으로 DDL 언어가 자동 생성됩니다. 또한 find by Id, delete from 과 같은 기본적인 쿼리 언어들은 JpaRepository 클래스에 정의 되어 있습니다. Repository 인터페이스를 만든 뒤, 해당 클래스를 extend하는 것으로 쿼리 언어를 메소드 명으로 사용 가능하기에 별도의 쿼리 언어를 작성하지 않고 사용합니다. 시작에 앞서 JPA에서 쿼리를 대체하는 방식들에 대해 간략하게 설명하고 넘어가겠습니다.

아래는 JPA의 Table Create 예시 입니다.

```
public class Classes {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "class_id", nullable = false)
    private Long classId;

    1 usage
    @ManyToOne(targetEntity = Course.class, fetch = FetchType.LAZY)
    @JoinColumn(name="course_id")
    private Course course;

    1 usage
    @Column(nullable = false)
    private int classNumber;

    1 usage
    @Column(nullable = false)
    private String professorName;
```

Classes 는 Classes table로써 동작하고, @Column 어노테이션을 통하여 각 attribute에 대한 정보를 구성합니다. 이를 토대로 자동으로 DDL 명령어가 실행되어 테이블을 create 합니다.

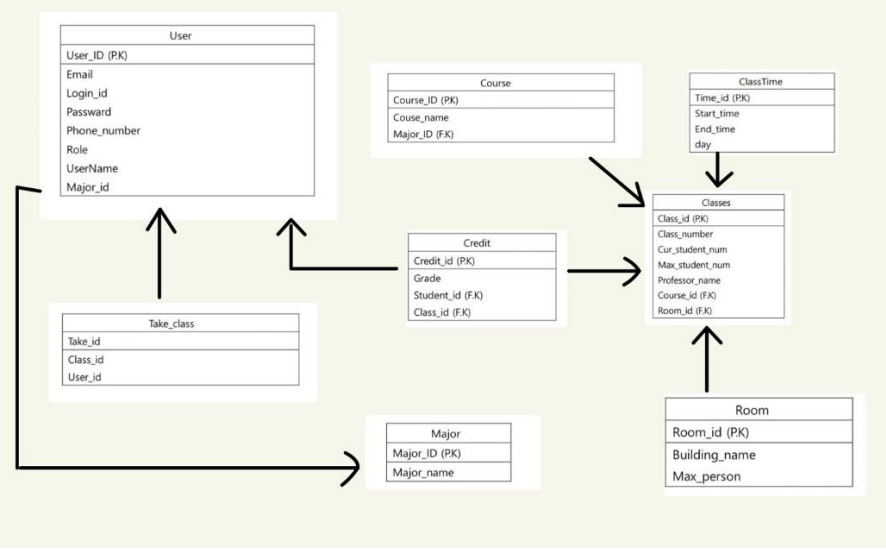
아래는 쿼리언어를 대체하는 쿼리 메소드의 예시입니다.

```
1 usage   sanghyunyoon *
public List<Classes> findByCourse(Long courseId) { //classdto로 받아줌.
    return classesRepository.findByCourse(courseId);
}
```

findByCourse() 라는 메소드를 통해 쿼리 언어를 대체합니다.

또한 JPA를 사용하여 Table을 생성시, Primary Key와 Foreign Key들은 설정해준 연관관계에 따라 (many to one, one to one 등) 자동으로 설정되기에, 제가 여태까지 구상한 스키마의 대략적인 틀만 임의로 만들어서 보여드리겠습니다.

Schema 설명



위 스키마가 제가 임의로 만든 스키마로, 각 유저들은 관리자, 혹은 사용자일 수 있습니다.

User 의 Role attribute 를 통하여 관리자와 사용자를 구분합니다. User은 major, take_class, credit 테이블을 참조하고 있는데, 이는 각각 그 학생의 전공, 그 학생이 듣는 수업, 그리고 수강한 수업의 학점 정보를 받아오게 됩니다. 복수전공의 가능성을 배제하고 구현할 예정이기에 1개의 전공

이 다수의 학생을 가질 수 있게 학생과 전공(Major)의 연관관계를 Many to One 으로 설정해주었습니다.

한 명의 학생은 여러 수업을 들을 수 있기에 학생과 듣는 수업(take_class)의 연관관계는

One to Many로 설정해주었고, 1명의 학생은 다수의 수업으로부터 다수의 점수(credit) 을 받을 수 있기에 역시 One to Many 관계로 설정해주었습니다.

Classes 테이블의 경우 Room 테이블에서 수업이 열리는 건물과 방 번호, 그리고 최대 수용 인원을 가져오고, ClassTime 테이블에서는 수업이 열리는 요일과 시작시간, 끝나는 시간을 받아오게 됩니다. 그리고 course 테이블에서 해당 class 가 어떤 과목을 배우는 수업인지에 대한 정보를 받아오게 됩니다. Classes와 Room 테이블의 관계는 편의를 위해 한 개의 수업이 한 개의 강의실에서만 열린다는 전제로 One to One 관계로 설정해주었고, Classes 테이블과 Course 테이블의 경우,

1개의 수업에는 1개의 과목만 가르치지만, 1개의 과목은 여러 수업으로 열릴 수 있기에

One to Many 관계로 설정하였습니다. 마지막으로 Class table과 ClassTime의 경우, 1개의 수업이 일주일에 2번이상 열릴 수 있기에 여러 시간대에 수업이 존재할 수 있게 One to Many 로 설정해주었지만, 추후 구현에 따라 이 부분을 수정될 수 도 있을 것 같습니다.

사용자 로그인

사용자 계정 관리(로그인, 로그아웃, 회원가입) 의 경우 Spring Security를 활용하여 구현할 예정입니다. 사용자 계정의 새로 생성하게 될 경우, Controller에서 Get-Mapping 방식을 통하여 url을 매핑 받은 뒤, Model에 비어 있는 DTO(Data translate object) 를 전달합니다. 그럼 login.html에서는 해당 DTO를 받은 뒤, 유저가 입력한 정보에 따라 DTO를 채우고, 유저가 '회원가입 완료' 버튼을 누르면 채워진 DTO를 다시 Controller에 전송합니다.

그럼 Controller는 다시 Post-Mapping 방식을 통하여 채워진 DTO를 받은 뒤, 데이터 베이스에 커밋하기 전, 정보를 Transaction 단위로 저장하는 영속성 컨텍스트에 관한 메소드들이 구현되어 있는 Repository 클래스에서 save 함수를 통하여 새로운 유저의 정보를 추가하게 됩니다.

추가된 정보들은 Spring Security에 의해 관리되며, Spring Security에서는 추가된 회원의 비밀번호를 암호화하는 역할과, 로그인하지 않은 anonymous 사용자는 로그인 뒤에 이용할 수 있는 서비스들을 사용하지 못하게 제한하게 됩니다. 만약 로그인이 되어있지 않아 anonymous 상태인 사용자가 다른 곳에 접근 시, login page로 redirect되어 로그인 정보를 입력하도록 유도합니다.

사용자가 로그인하게 되면, 로그인 정보를 받아온 뒤,

Spring Security 영역에서 비밀번호를 Decoding 하게 됩니다. 만약 올바른 로그인 정보가 들어왔다면, 사용자는 로그인에 성공하고, 이후부턴 @AuthenticationPrincipal 과 같은 어노테이션을 사용하여 로그아웃하기 전까지 로그인 되어있는 유저의 정보를 언제든 받아올 수 있습니다.

이를 통하여 수강신청, 수강신청 내역확인, 등 다양한 유저기능을 수행할 때, 자신의 유저정보를 손쉽게 받아 처리할 수 있습니다. 그리고 현재 사용자가 관리자인지, 사용자인지는 User table의

Role attribute를 통하여 구분합니다.

사용자 기능

로그인 후 Home화면으로 이동하게 되고, 그곳에서 수강신청, 신청내역 확인, 과목 조회 등 사용자 기능을 수행할 수 있습니다. 과목을 조회할 시에, 수업이름, 학수번호 등 정보로 검색을 할 수 있고, 이때 html에 입력한 검색정보는 URL 파라미터로 전달되어 Controller로 온 뒤, Service에 구현한 메소드를 통하여 Query를 날려 원하는 정보를 다시 html에 전달하여 검색한 정보를 노출합니다. 수강 신청의 경우 해당 과목을 검색한 뒤 수강신청 버튼을 누르면, 해당 정보를 영속성 컨텍스트에 저장하게 되는데, 이는 유저정보를 저장했던 방식과 유사하게 Repsoitory에 미리 정의한 Save함수를 통하여 저장할 예정입니다. 그리고 수강 취소의 경우도 마찬가지로 Repository에 정의된 delete함수를 통하여 구현할 예정입니다. User는 Major(전공) 과 TakeClass 테이블을 참조하고 있고, Major 테이블을 참조하여 수강할 과목을 선택하고, Takeclass를 통해 수강신청한 과목을 관리할 수 있습니다.

관리자 기능

Role attribute를 통하여 현재 로그인한 사용자가 관리자인지, 혹은 사용자인지 식별하게 됩니다. 관리자 기능의 경우, Home 화면에 "관리자 페이지" 라는 버튼이 존재하게 되는데, 해당버튼을 누르게 된다면, 관리자 기능들이 모여 있는 adminHome.html로 이동할 수 있게 됩니다.

하지만 해당 버튼을 누른 User의 Role이 관리자 (admin) 이 아닐 경우, 사용자는 관리자 페이지에 접근하지 못한 채 다시 홈 화면으로 돌아오게 됩니다. Class의 개설과 폐강, 최대 정원 수정은 User table에 tuple을 넣고 빼는 것, 그리고 User tuple을 수정하는 방식과 유사하게 구현할 예정입니다.

기타 명세

관리자의 경우, 최초에 1명의 관리자만을 insert 하게 됩니다. 이후로 추가적인 관리자가 필요한 경우엔, 일반적인 유저처럼 html에 회원가입 정보를 입력하여 추가하는 것이 아닌, 데이터베이스에 직접 접근하여 insert 명령문을 통하여 삽입해 주어야 합니다. 또한 기본적으로 존재하는 과목, Class, 전공 등등은 초기에 mysql에 DDL 명령어를 통하여 삽입해 주어야 합니다. 이번 프로젝트의 전반적인 설정은 기존의 application.properties 파일이 아닌, application.yml로 yaml 파일을 통하여 구현하였고, 여기서 주요한 사항으로는 database의 특성에 대한 정의가 있습니다.

DDL-auto : update가 기본적인 설정으로 적용 되어있는데, 이를 통하여 영속성 컨텍스트에 저장해 둔 정보들이 실질적으로 데이터 베이스에 commit되게 하여 project를 종료하더라도 변경사항이 업데이트 되어 저장합니다. 테스트 환경의 경우엔 DDL-auto: create 설정을 통하여 데이터가 create-drop 방식으로 동작하게 하여 테스트마다 데이터베이스 환경이 초기화되도록 해줄 예정입니다.