

# MileStone 2 Wiki

2017030055 윤상현

## Design

구현환경은 java 언어를 사용한 Spring Boot 프레임워크를 사용한 back-end 구현과, Mysql workbench에서 db를 사용하였고, thymeleaf를 사용하여 front-end를 구현하였습니다.

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 교

신하는 Controller 파트, 그리고 데이터베이스와 데이터를 주고 받을 때 데이터를 저장할 DTO(data translate object), 상세한 기능을 구현할 Service 파트로 구분하였습니다.

Spring boot 에선 JPA라는 틀을 사용하여 Class 객체가 table, 각 Class들의 엔티티 객체들이 튜플에 대응하여 자동으로 DDL 언어가 자동 생성됩니다. 또한 일반적이지 않은, 혹은 정형화 하기 힘든 쿼리들의 경우 별도의 @Query라는 어노테이션을 통해 정의 해줘야 하지만, find by Id, delete, save 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  sanghyunyon \*
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.

The diagram illustrates the following tables and their attributes:

- User**: User\_ID (PK), Email, Login\_id, Password, Phone\_number, Role, UserName, Major\_id
- Room**: Room\_id (PK), Building\_name, Max\_person, Room\_number
- Hope**: hope\_id (PK), User\_id (FK), Class\_id (FK)
- Take\_class**: Take\_id, Class\_id, User\_id
- Major**: Major\_ID (PK), Major\_name
- Course**: Course\_ID (PK), Course\_name, Major\_ID (FK)
- Classes**: Class\_id (PK), Class\_number, Cur\_student\_num, Max\_student\_num, Professor\_name, Day, Start\_time, End\_time, Average\_Score, Course\_id (FK), Room\_id (FK)

Relationships (indicated by arrows):

- User to Room: User\_id (FK) to Room\_id (FK)
- User to Major: User\_id (FK) to Major\_ID (FK)
- Major to Course: Major\_ID (FK) to Major\_ID (FK)
- Course to Classes: Course\_id (FK) to Course\_id (FK)
- Classes to Room: Room\_id (FK) to Room\_id (FK)
- Classes to User: User\_id (FK) to User\_id (FK)
- Classes to Hope: Class\_id (FK) to Class\_id (FK)
- Classes to Take\_class: Class\_id (FK) to Class\_id (FK)
- Take\_class to User: User\_id (FK) to User\_id (FK)
- Take\_class to Classes: Class\_id (FK) to Class\_id (FK)
- Hope to Classes: Class\_id (FK) to Class\_id (FK)
- Hope to User: User\_id (FK) to User\_id (FK)

Handwritten note: many → one

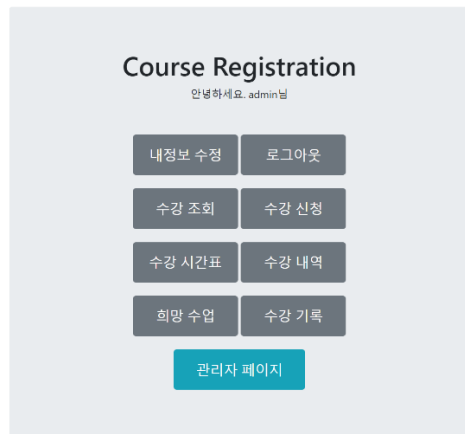
User 의 Role attribute 를 통하여 관리자와 사용자를 구분합니다. User은 major, take\_class 테이블을 참조하고 있는데, 이는 각각 그 학생의 전공, 그 학생이 듣는 수업 정보를 받아오게 됩니다. 복수전공의 가능성을 배제하고 구현할 예정이기에 1개의 전공이 다수의 학생을 가질 수 있게 학생과 전공(Major)의 연관관계를 Many to One 으로 설정해주었습니다.

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 1개의 강의실에서 여러 강의가 발생할 수 있기에 many 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.

## UI 디자인

### Home

Home  
수강신청 시스템



The image shows a UI design for a 'Course Registration' system. At the top, it says 'Course Registration' and '안녕하세요. admin님'. Below this, there are several buttons arranged in a grid: '내정보 수정', '로그아웃', '수강 조회', '수강 신청', '수강 시간표', '수강 내역', '희망 수업', and '수강 기록'. At the bottom, there is a button labeled '관리자 페이지'.

여러가지 기능을 모아 놓은 Home.html 입니다. 각 버튼을 누를 시, 해당 작업을 수행하는 Controller에 매핑 후, 필요하다면 정보를 처리해준 뒤, 다시 적절한 프론트 페이지(html)을 리턴 해줍니다. 맨 아래 관리자 페이지의 경우, 관리자 유저만이 클릭하여 접근 가능합니다.

### Login

#### 로그인



The image shows a UI design for a login page. It has two input fields: '로그인 ID' and '비밀번호'. Below the input fields, there are three buttons: '로그인' (blue), '회원가입' (gray), and '취소' (gray). At the bottom, there is a copyright notice: '©2022 sanghyun'.

로그인 페이지의 경우, Home.html 의 login 버튼이나 Spring security 를 사용하여 이동할 수 있습니다. 전자의 경우 home 페이지의 login 버튼을 사용하면 되고, 후자의 경우, 만약 현재 로그인되어있지 않은 유저는 Spring Security에서 anonymous User로 분류되어 Home 화면, Login 화면, SignUp 화면을 제외한 화면에 대한 접근권한이 없고, 해당 페이지들에 접근하려는 시도가 발생하면 자동으로 Login 화면으로 매핑되어 login 을 유도합니다. 아이디와 비밀번호를 입력 후 로그인을 클릭시 html 에 tymeleaf의 form 에 채워진 정보들이 submit 되며 login을 처리하는 컨트롤러

에 Mapping 됩니다. 회원가입 버튼을 클릭 시 signup.html 페이지로 이동되며, 취소 버튼 클릭 시 다시 홈 화면으로 이동합니다.

## 수강편람

[Home](#)

수강신청 시스템

Computer Science

검색

#	전공명	과목명	분반 조회
1	Computer Science	Java Programming	분반 조회
2	Computer Science	database	분반 조회

©2022 sanghyun

수강편람의 경우, 수업조회 버튼을 클릭 시 이동됩니다. 왼쪽의 박스에서 전공을 선택하면 해당 전공에 맞는 과목들이 나타나고, 빈 창에 키워드를 검색하면, 해당 키워드가 포함된 과목만 나타나게 됩니다. 그리고 내가 보고 싶은 과목의 수업 조회 버튼을 클릭 시, 아래 그림과 같이 해당 과목에서 개설된 모든 수업 정보들이 나타나게 됩니다.

[Home](#)

수강신청 시스템

Back

Computer Science - Java Programming

과목명	교수	학수번호	정원	인원	시간	교실
Java Programming	KIM	01	4	3	요일: 1 시작: 12 종료: 16	ITBT 508호
Java Programming	LEE	02	2	1	요일: 2 시작: 12 종료: 16	ITBT 508호

여기서 시간과 요일, 학점에 대한 데이터에 대한 설명을 하고 넘어가겠습니다. 세 정보 모두 Long 타입으로 정의하였고, '요일'의 경우 1L ~ 7L 이 월~일 요일을 나타냅니다.

시간의 경우, 0~ 48 이 존재하고, 1= 30분 입니다. 즉 24 = 12시 이고, 25 = 12시 30분 입니다.

Grade도 이와 유사하게 0~8 이 존재하는데, 1= 0.5 학점입니다. 즉 F =0, B0 = 5 , A+ = 8 로 표현됩니다. 이는 float 등의 소수점을 포함한 type 들이 가지는 부동소수점 문제를 염두하여 설계한 것으로, 추후 직관적인 정보로 변환되게 추가할 예정이지만, 당장의 프로젝트에선 데이터가 올바르게 전달되는 것에 초점을 두어 직관적인 부분을 일부 포기하였습니다.

## 수강신청

[Home](#)

수강신청 시스템

Computer Science

검색

과목명	교수	학 수 번 호	정 원	인 원	시간	교실	희망수 업	신청
Java Programming	KIM	01	4	3	요일: 1 시작: 12 중 료: 16	ITBT 508 호	희 망 추 가	수 강 신 청
Java Programming	LEE	02	2	1	요일: 2 시작: 12 중 료: 16	ITBT 508 호	희 망 추 가	수 강 신 청

Home 페이지에서 수강신청 버튼을 클릭 시 Mapping 됩니다. 희망 추가 버튼을 클릭 시 해당 수업을 희망 수업 리스트에 추가합니다. 신청 버튼을 클릭 시 해당 수업을 신청하는데, 만약 이미 신청한 내역들 중 이번 학기에 수강하는, 즉 아직 성적이 기입 안된 수업들 중 이미 해당 수업과 똑 같은 과목을 수강했거나, 겹치는 시간대에 수업을 신청하였다면, 에러 메시지와 함께 수강신청이 되지 않습니다.

## 희망수업

[Home](#)

[수강신청 시스템](#)

admin님의 희망수업 내역

과목명	교수	학수번호	수강신청	희망삭제
Java Programming	KIM	01	수강 신청	CANCEL
Java Programming	LEE	02	수강 신청	CANCEL
math1	KIM2	05	수강 신청	CANCEL
database	shy1	03	수강 신청	CANCEL

©2022 sanghyun

희망추가 버튼을 클릭 시 해당 수업이 이곳에 담깁니다. 수강 신청 버튼 클릭 시 수강신청 란에서 수강신청 버튼을 클릭한 것 과 똑 같은 방식으로 수강 신청을 시도하고, 희망 삭제 버튼을 클릭 시 해당 수업을 희망 수업 목록에서 지웁니다.

## 수강기록

[Home](#)

수강신청 시스템

admin님의 수강기록 평균학점 4

과목명	교수	분반	학점	수강취소
math2	shy3	08	3	CANCEL
Java Programming	KIM	01	5	done
law2	aaa	00	7	done

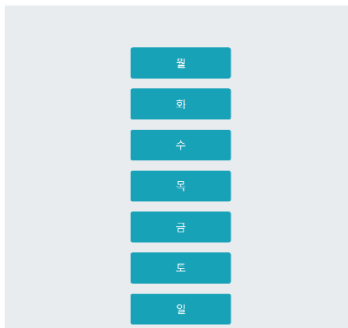
©2022 sanghyun

수강했던 수업들 중, 학점이 기입된, 즉 이수가 완료된 과목들입니다. 만약 수강한 과목의 학점이 B0 (5) 보다 낮다면 수강 취소(학점 포기) 가 가능하고, 이후 다시 수강신청 할 수 있게 됩니다. 하지만 B0 이상이라면, 수강 취소 버튼 대신 done 이라는 문구가 나타나며, 학점을 포기할 수 없고, 이에 따라 재수강 역시 불가능 하게 됩니다.

## 수강 시간표

[Home](#)

수강신청 시스템



Home 화면에서 수강 시간표 버튼을 클릭 시, 요일을 선택할 수 있는 페이지로 이동합니다.

해당 요일을 클릭 시, 해당 요일에 들어야 하는 수업들이 시간 순으로 표기 됩니다.

시간표 화면은 아래와 같습니다.

[Home](#)

수강신청 시스템

admin님의 1요일 수강 시간표

과목명	교수	교실	시간
math1	LEE2	HIT 100호	12 to 16
law1	Sam	ITBT 508호	16 to 20

©2022 sanghyun

‘수강 시간표’ 는 현재 듣고 있는 수업 기준으로 만들어지기에, 학점이 기입된 과목들은 모두 제외 되고, 아직 학점이 기입되지 않은 과목들만 나타납니다.

## 관리자 홈

[Home](#)  
수강신청 시스템



위 사진은 관리자가 홈 화면에서 관리자 페이지 버튼을 클릭 시 이동하는 화면입니다.

과목 추가 버튼은 새로운 강의를 신설할 때 사용되고, 과목 삭제 및 수정은 수업 정보( 수강인원)을 변경하거나 해당 수업을 폐강할 때 사용합니다. 학생 정보는 모든 학생들의 정보를 조회하고, 해당 학생이 수강완료한 수업의 점수를 기입하는 데에 사용됩니다. 여기서 수업 점수를 기입하면, 해당과목은 이수완료 된 것으로 처리되게 됩니다. 그리고 통계 는 OLAP 으로 써 통계자료를 보여줍니다.

## 수업추가

신규 수업 등록

수업 정보 등록	
최대 정원	<input type="text"/>
반 번호	<input type="text"/>
교수 명	<input type="text"/>
요일	<input type="text"/>
시작 시간	<input type="text"/>
종료 시간	<input type="text"/>
전공	<input type="text"/>
과목 선택	<input type="text"/>
교실 선택	<input type="text"/>
<input type="button" value="수업 추가"/> <input type="button" value="취소"/>	



수업을 추가하는 곳입니다. Db 안에 들어있는 과목, 교실들 중 하나를 select 한 뒤 수업 정보를 기입하여 수업 추가 버튼을 클릭 시 새로운 수업이 생성됩니다.

## 폐강 및 수업 정보 수정

[Home](#)

수강신청 시스템

Computer Science - Java Programming						
과목명	교수	분반	정원	인원	폐강	정원 변경
Java Programming	KIM	01	4	3	폐강	정원변경
Java Programming	LEE	02	2	1	폐강	정원변경
Java Programming	KIM	01	3	0	폐강	정원변경
Java Programming	LEE	02	7	0	폐강	정원변경

©2022 sanghyun

관리자 페이지에서 수업 정보 수정, 폐강 버튼을 클릭 시 이동합니다. 폐강 버튼 클릭 시 해당 수업이 데이터 베이스에서 사라지고, 정원 변경 시 새로운 정원을 입력하는 페이지로 redirect 되며, 거기서 입력한 정원대로 해당 수업의 최대 인원이 수정됩니다.

## 학생정보 조회 및 성적 기입

[Home](#)

수강신청 시스템

이름	전공	연락처	이메일	평균학점	수업조회
admin	Computer Science	01	admin@n	4	수업조회
u1	Computer Science	01	u1@u	0	수업조회
u2	Mathematics	02	u2@u	4	수업조회
u3	law	03	u3@uu	1	수업조회
u4	chemical	04	u4@u	4	수업조회
u5	physics	05	u5@u	2	수업조회
u6	physics	01	s@n	0	수업조회

©2022 sanghyun

학생 정보 버튼을 클릭 시 해당 학생의 정보가 나타나고, 수업조회 버튼을 클릭 시 해당 학생이 들었던 수업들이 나타납니다

## admin님의 수강신청 내역

과목명	교수	학수번호	점수변경
Java Programming	KIM	01	점수변경
law2	aaa	00	점수변경
math2	shy3	08	점수변경
chemical1	aab	01	점수변경
math1	LEE2	06	점수변경
law1	Sam	09	점수변경

여기서 점수 변경을 클릭 시 해당 학생의 점수 정보를 변경하고, 이는 곧 점수 기입을 의미하기도 하여 만약 점수가 99(디폴트 값)에서 이외의 값으로 변경되었다면 해당 과목이 이수 완료된 것으로 처리됩니다.

## 통계화면

플래스 id	과목명	교수명	편차
10	math2	shy3	2
5	database	shy1	1
7	math1	KIM2	1
1	Java Programming	KIM	0
9	math2	KIM3	0
11	physic1	hajinpark	0
12	database	sanghyunyoona	0
13	chemical1	HominJoo	0
14	law1	ByunggunLee	0
15	physci2	eee	0

편차는 과목 평균 - 해당 과목을 수강한 학생들의 평균 학점의 평균 의 값으로, 그 값이 높을수록 학점을 짜게 주는 사람이 되고, 해당 페이지에선 가장 학점을 짜게 주는 10개의 수업을 순서대로 보여줍니다.

## implement

### 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.

```
1 senghyeyoon
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/home", "/login/**", "/css/**", "/signup/**").permitAll()
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .usernameParameter("loginId")
        .passwordParameter("password")
        .loginPage("/login")
        .loginProcessingUrl("/login/action")
        .successHandler(authSuccessHandler)
        .failureHandler(authFailureHandler)
}
```

위 사진은 Spring Security 관련 설정을 구현한 Security Config 의 일부분 입니다.

.antMatchers() 에 파라미터로 들어온 Url 들은 anonymous 유저도 접근 가능한 url로, 해당 url을 제외한 url에 anonymous 유저가 접근 시에, login 파트로 redirect 해줍니다.

사용자가 로그인하게 되면, 로그인 정보를 받아온 뒤, Spring Security 영역에서 비밀번호를 Decoding 하게 됩니다. 만약 올바른 로그인 정보가 들어왔다면, 사용자는 로그인에 성공하고, 이후부터 @AuthenticationPrincipal 과 같은 어노테이션을 사용하여 로그아웃하기 전까지 로그인 되어있는 유저의 정보를 언제든지 받아들 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 'aaaa' attribute를 통하여 구분합니다.

## 사용자 기능

로그인 후 Home화면으로 이동하게 되고, 그곳에서 수강신청, 신청내역 확인, 과목 조회 등 사용자 기능을 수행할 수 있습니다. html에서 버튼 클릭 등을 통하여 확인된 action 들은 URL 파라미터로 전달합니다.

```
<p th:unless="${user}" style="...">
  <a class="btn btn-lg btn-secondary" style="..." href="/signup">회원 가입</a>
  <a class="btn btn-lg btn-secondary" style="..." href="/login">로그인</a>
</p>
```

위 사진은 예시입니다. home.html에서 유저가 '회원 가입' 버튼을 클릭 시, /signup 으로 mapping 해주고, 이는 Controller 가 @GetMapping, @PostMapping 등을 통하여 전달 받습니다.

```
sanghyunyon
@GetMapping("/signup")
public String signup(Model model,
    @RequestParam(value="msg", required = false) String msg) {
    List<Major> majors = majorService.findAll();
    model.addAttribute( attributeName: "signupDto", new UserSignupDto());
    model.addAttribute( attributeName: "majors", majors);
    model.addAttribute( attributeName: "msg", msg);
    return "signup";
}
```

Service에 구현한 메소드 사용하고, Repository단에 정의 되어있는 Query를 날려 원하는 정보를 다시 Service로, 그리고 다시 Controller로 전달하여 Controller에서 요청 받은 데이터를 수신합니다. 이후 Controller에선 addAttribute() 메소드를 통하여 Model에 프론트에서 요청한 정보를 넣고, 프론트 페이지에 반환해줍니다. 이후 프론트에선 새롭게 추가할 데이터 ( 이 경우 회원 가입 정보) 를 입력 해준 뒤, 아래의 완료 버튼을 클릭합니다.

```
<div class="col">
  <button class="w-100 btn btn-primary btn-lg" type="submit">회원 가입</button>
</div>
```

해당 버튼 클릭 완료 시, 기존의 Url로 PostMapping 을 통해 Controller로 전달하여 후처리를 담당하게 됩니다.

```

@PostMapping("/signup")
public String create(UserSignupDto signupDto, Model model) {
    try {
        userService.join(signupDto);
    } catch (Exception e) {
        List<Major> majors = majorService.findAll();

        model.addAttribute("majors", majors);
        model.addAttribute("signupDto", signupDto);
        model.addAttribute("msg", e.getMessage());
        return "signup";
    }

    return "redirect:/";
}

```

userService에 정의한 Join 함수를 통하여 새로운 유저를 유저 테이블에 추가하게 됩니다.

```

@Transactional
public Long join(UserSignupDto signupDto) {
    userRepository.findByLoginId(signupDto.getLoginId())
        .ifPresent(user -> {
            throw new IllegalArgumentException("Failed: Already Exist ID!");
        });

    if (!signupDto.getPassword().equals(signupDto.getPasswordConfirm())) {
        throw new IllegalArgumentException("Failed: Please Check Password!");
    }

    User user = User.signupBuilder()
        .loginId(signupDto.getLoginId())
        .password(passwordEncoder.encode(signupDto.getPassword()))
        .username(signupDto.getUsername())
        .email(signupDto.getEmail())
        .phoneNumber(signupDto.getPhoneNumber())
        .major(majorRepository.findById(signupDto.getMajorId()).get())
        .build();
    return userRepository.save(user).getUserId();
}

```

Join 은 회원 정보를 담은 DTO를 받고, 그 정보를 Builder를 활용하여 새로운 객체를 생성한 뒤, Spring data JPA로 인하여 정의된 userRepository의 save 함수를 통해 영속성 컨텍스트에 저장합니다. @Transactional 어노테이션으로 해당 작업은 하나의 transaction 안에 들어오게 되고, 만약 중간에 에러 발생 시, transaction 단위로 rollback 되기에 데이터가 일부만 저장되어 오류를 유발하는 현상을 방지합니다. 여기서 userRepository.findByLoginId() or save() 메소드는 Spring data JPA가 미리 정의한 메소드로, 메소드명을 통하여 사용할 수 있습니다.

여기까지가 회원 가입 예시를 통해 알아본 전반적인 html( front) -Controller - Service - Repository - database 의 연관 관계이고, 이들이 서로 interact 하는 과정은 메소드마다 유사하기에 다른 메소드들의 자세한 설명은 위 설명으로 대체하겠습니다.

수강 신청의 경우 해당 과목을 검색한 뒤 수강신청 버튼을 누르면, 해당 정보를 영속성 컨텍스트에 저장하게 되는데, 이는 유저정보를 저장했던 방식과 유사하게 Repository에 Spring data JPA로 인하여 정의된 Save함수를 통하여 저장합니다. 그리고 수강 취소의 경우도 마찬가지로 Spring data JPA로 인하여 정의된 Repository에 정의된 delete함수를 통하여 구현하였습니다.

User는 Major(전공) 과 TakeClass 테이블을 참조하고 있고, Major 테이블을 참조하여 수강할 과목을 선택하고, Takeclass를 통해 수강신청한 과목을 관리할 수 있습니다. TakeClass는 User 테이블과

Classes 테이블을 참조할 수 있는 2개의 Foreign Key와 Grade 정보가 있는데, 이 Grade 값은 default 가 99L (Long type)으로 설정 되어있습니다. 이 99L 이라는 수치는 디폴트 값으로 분류되어 평균 성적 등에 반영 되지 않고, 성적이 관리자에 의해 기입되지 않았을 때 가지게 되는 값이므로, 성적 값에 99L 이라는 디폴트 값이 존재하는 경우, 해당 수업을 아직 이수하지 않은 것으로 판단하여 수강신청 내역에 노출되고, 수강 취소를 할 수 있게 됩니다. 하지만 만약 관리자에 의하여 성적이 기입된다면, 해당 수업을 이수한 것으로 판단하여 신청 내역에서 수강 기록으로써 역할을 하게 됩니다. 아래는 제가 설명한 것과 관련된 TakeClass Repository의 제가 정의한 Query 들입니다.

```
//해당 학생이 수강한 과목중 학점이 입력되었고, B0 이상이면 더이상 버릴 수 없다.
1 usage new *
@Query ("select tc from TakeClass tc where tc.user.userId =:userId and tc.grade >4L and tc.grade <> 99L")
List <TakeClass> findEndClass( @Param("userId") Long userId );

//해당 학생이 수강한 과목 중, 성적이 기입되어 해당과목을 이수했지만, 학점이 B0 아래라 재수강 가능한 과목
1 usage new *
@Query ("select tc from TakeClass tc where tc.user.userId =:userId and tc.grade <5L and tc.grade <> 99L")
List <TakeClass> findNotEndClass( @Param("userId") Long userId );

//해당 학생이 수강신청 하였지만, 아직 성적이 기입되지 않아 이번 학기의 수강신청 내역에 남아있는 수업
1 usage new *
@Query ("select tc from TakeClass tc where tc.user.userId =:userId and tc.grade = 99L")
List <TakeClass> findNotDoneClass( @Param("userId") Long userId );
```

TakeClass 테이블은 1개의 테이블로써 존재하지만, logical 하게 여러가지 의미를 담은 데이터들의 집합으로써 역할을 수행합니다. 유저가 수강 신청을 성공한 모든 Classes 들은 TakeClass 테이블에 들어가게 된 뒤, '이번 학기에 수강 중인 수업' 들로써 동작하지만, "Grade" attribute에 값이 기입되어 성적이 기입 완료 된다면, '수강 완료된 과목'으로써 저장되어지고, 그 중에서도 "Grade가 B0 이상이나, 미만이나" 로 나뉘는 뒤, 수강 취소 가능한 과목과 불가능한 과목으로 나뉩니다.

findEndClass는 성적 기입이 아직 안된 과목, 즉 이번 학기 수강 과목을 찾는 Query 이고,

findNotEndClass 는 성적 기입이 완료되었지만, 학점이 B0 아래라 재수강 가능한 과목,

그리고 findNotDoneClass 는 아직 성적이 기입되지 않은 과목을 찾는 역할을 수행합니다.

위 Query 들을 통하여 실제로는 TakeClass 라는 한 개의 테이블에 있는 정보들을 3가지의 상태로 구분 짓습니다.

## 관리자 기능

'aaaa' 라는 이름으로 정의한 Boolean-type attribute를 통하여 현재 로그인한 사용자가 관리자인지, 혹은 사용자인지 식별하게 됩니다. 'aaaa' 라는 attribute은 default 가 false 이고, 해당 attribute를 True로 바꾸기 위해선, 직접 연결된 Db 에 update DDL을 통하여 영역을 바꿔 주어야 합니다. 저는 테스트를 위해 User\_id 가 1인, 가장 처음 등록된 사용자의 'aaaa' attribute를 true로 바꾸어 관리자 기능을 테스트하였습니다.

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. 아래는 '관리자 페이지' 버튼을 클릭 시 Mapping 되는 컨트롤러 입니다.

```
sanghyunyeon
@GetMapping(value = {"/adminHome"})
public String home(Model model) {
    Object user = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    User us=(User) user;
    Log.info("{", user);
    if (user != "anonymousUser") {
        model.addAttribute( attributeName: "user", us);
    }

    if(!us.isAaaa()) ///관리자 아니면 home으로 다시 or 여러 문구 ( 문제: 로그아웃된 홈 화면으로)
        return "home";

    return "adminHome";
}
```

isAaaa() 메소드는 해당 유저 테이블의 'aaaa' attribute 가 True 인지 False 인지 구분하는 역할이고, 만약 'aaaa' attribute 가 True 라면 관리자이므로 adminHome 이라는 관리자 html 파일을 반환하고, 만약 False로 일반 유저가 클릭했을 경우, 관리자 페이지로 이동하지 않고 다시 기본 home 화면으로 돌아옵니다.

새로운 수업 추가의 경우, 새로운 유저를 추가하거나 새로운 수업을 수강신청하는 것과 유사하게 동작합니다. 새로운 수업 정보를 입력 받고 완료 버튼을 클릭 시, 입력한 정보가 submit 된 뒤 Controller 에 매핑 되고, Controller 에선 입력된 정보를 토대로 Domain에서 정의된 Builder 메소드를 통해 새로운 tuple을 생성하고 Service에 구현된 Save 메소드를 통하여 이를 영속성 컨텍스트에 저장하게 됩니다. 폐강 역시 수업 신설과 유사하게 동작 되며, save() 대신 delete() 를 사용하여 튜플을 제거합니다.

```

1 usage
@OneToMany(mappedBy = "classes", orphanRemoval = true)
private List<TakeClass> takeClasses;

@OneToMany(mappedBy = "classes", orphanRemoval = true)
private List<HopeClass> hopeClasses;

```

단, 명세에 수업이 폐강되면 수강신청 내역, 희망수업에서 사라져야 하기에, Classes 도메인에서 TakeClass, HopeClass 와 이어주는 Foreign 키들을 설정 할 때 orphanRemoval 속성을 True로 체크하여 수업 객체가 delete 되면 연결된 TakeClass 객체들과 HopeClass 객체들이 사라지게 됩니다.

## OLAP

```

1 usage 1 sanghyunyeon
@Query("select avg(tc.grade) from TakeClass tc where tc.user.userId = :userId and tc.grade <= 99L group by tc.user.userId")
Optional<Long> findUserAverage(@Param("userId") Long userId); // "userId" 와 Long userId 이름 일치해야!!!!!! 디폴트인 99일사 반영 x

1 usage 1 sanghyunyeon
@Query("select distinct avg(tc.grade) from TakeClass tc where tc.grade <= 99L and tc.classes.classId=:classId group by tc.classes.classId ")
Long findClassAverage(@Param("classId") Long classId);

1 usage 1 sanghyunyeon
@Query("select distinct avg(tc.user.averageScore) from TakeClass tc where tc.grade <= 99L and tc.classes.classId=:classId group by tc.classes.classId ")
Long findClassUserAverage(@Param("classId") Long classId);

```

가장 성적을 짜개 주는 Top 10개의 과목을 찾는 OLAP 을 구현하기 위해 위 3개의 query를 구현 하였습니다. findUserAverage는 해당 유저의 누적 학점 평균을 나타냅니다. findClassAverage는 해당 수업을 들은 학생들이 해당 과목에서 받은 학점의 평균입니다.

findClassUserAverage는 findUserAverage로 각 유저의 averageScore attribute에 누적 평균을 넣어 준 것을 활용하여 해당 수업을 듣는 학생들의 누적 평균 학점의 평균을 구합니다.

즉 findClassAverage – findClassUserAverage 한 값이 저희가 구하고자 하는 학점을 짜개주는 기준 이 되는 편차가 됩니다.

## 기타

관리자의 경우, 최초에 1명의 관리자만을 insert 하게 됩니다. 이후로 추가적인 관리자가 필요한 경우엔, 일반적인 유저처럼 html에 회원가입 정보를 입력하여 추가하는 것이 아닌, 데이터베이스 에 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 구현할 수 있습니다.

## 빌드 과정

과목( 데이터 베이스 시스템 등) 이나 강의실 정보 는 관리자 페이지를 통해 추가하는 것이 아닌, mysql에서 DDL 을 통해 직접 insert 해줘야 하기에 적절하게 insert 정보를 기입해줍니다. 아래는 임의로 만든 DDL 목록입니다.

```
SET SQL_SAFE_UPDATES = 0;
insert into MAJOR (MAJOR_NAME) values ( 'Computer Science' );
insert into MAJOR (MAJOR_NAME) values ( 'Mathematics' );
insert into MAJOR (MAJOR_NAME) values ( 'Law' );
insert into MAJOR (MAJOR_NAME) values ( 'Chemical' );
insert into MAJOR (MAJOR_NAME) values ( 'Physics' );
insert into MAJOR (MAJOR_NAME) values ( 'Economics' );

insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Java Programming', 1);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Database', 1);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Math1', 2 );
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Math2', 3);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Law1', 4);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Law2', 5);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Chemical1', 4);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Chemical2', 4);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Physics1', 5);
insert into COURSE (COURSE_NAME, MAJOR_ID) VALUES ( 'Physics2', 5);

insert into ROOM (MAJ_PERSON, ROOM_NO, BUILDING_NAME) values ( 30, '30B', 'T101' );
insert into ROOM (MAJ_PERSON, ROOM_NO, BUILDING_NAME) values ( 30, '30B', 'NET1' );
```

아래는 데이터베이스와 연결하는 설정을 해준 application.yml 파일입니다.

```
datasource: #더블트인 h2 는 별도 설정없이도 된다.
  driver-class-name: com.mysql.cj.jdbc.Driver
  url: jdbc:mysql://localhost:3307/registerdb?serverTimezone=Asia/Seoul
  username: root
  password: root

jpa:
  hibernate:
    ddl-auto: update #기본 create and drop, update(db 유지) create: 테이블이 이미 있다면 drop하고 다시 creat
  properties:
    hibernate:
      format_sql: true # 실행되는 query를 보여줌

session:
  store-type: jdbc
  jdbc.initialize-schema: always
```

datasource: 아래의 정보는 데이터 베이스로 쓸 Mysql에 관련한 정보를 입력하였고,

Ddl-auto를 update로 하여 서버를 내려도 데이터가 유지되게 설정하였습니다.

이후 main 디렉토리 내부에 있는 CourseRegistration System를 실행하면 빌드 됩니다.

## 요구사항 결과

UI 와 관련된 실행 결과는 UI 디자인 파트에서 보였기에 그 외 디테일한 요구사항들의 결과를 첨부하겠습니다.

## 검색기능

Computer Science

Ja

검색

#	전공명	과목명	수업 조회
1	Computer Science	Java Programming	수업 조회

Computer Science

da

검색

#	전공명	과목명	수업 조회
1	Computer Science	database	수업 조회

©2022 sanghyun

위와 같이, Major를 선택하여 해당 전공에 관한 수업만 검색되게 할 수 있고, keyword 검색으로 결과를 받을 수 있습니다.

## 수강신청

성적 B0 이상일 시 재수강 불가.

localhost:8080/register?msg=Failed:%20above%20B0%20cannot%20re%20attend!

localhost:8080 내용:  
Failed: above B0 cannot re attend!

확인

이미 해당 과목을 수강했거나, 하고 있는 경우

localhost:8080 내용:  
Failed: Already Registered!

확인

해당 과목이 기존 과목과 시간대가 겹칠 경우

localhost:8080 내용:  
Failed: classTime is repeated

확인

이번 학기에 이미 18학점 이상을 수강한 경우.

localhost:8080 내용:  
Failed: cannot attend Over 18!

확인

## 수강취소

admin님의 수강신청 내역

과목명	교수	학수번호	수강취소
math1	LEE2	06	CANCEL
law1	Sam	09	CANCEL

localhost:8080/myCourses?msg=Success!

localhost:8080 내용:  
Success!

확인

[Home](#)

## 수강신청 시스템

admin님의 수강신청 내역

과목명	교수	학수번호	수강취소
law1	Sam	09	CANCEL

©2022 sanghyun

위와 같이 취소 시 메시지 반환과 함께 수강이 취소된다.

## Trouble Shooting

이번 과제에서 Spring boot를 통한 웹 개발을 프로젝트로 진행해 보았는데, 기존에 클론 코딩, 혹은 이론 공부만 해왔기에 제가 생각한 아이디어를 사전에 미리 디자인 하는 부분에서 미흡했습니다. 이로 인하여 점점 뒤로 갈수록 문제가 발견되었고, 그것을 수정하려면 여태 구현했던 모든 코드에서 해당 부분을 해결하기 위해 변경한 사항을 반영해 주어야 했기에 수정에 많은 시간이 소요 되었고, 시간적인 문제로 논리적으로 말이 안되거나 조금 불편한 방식으로 문제를 처리해야 하는 경우도 생겨 아쉬움이 남았습니다. 하지만 데이터 베이스 시간에 배운 data Schema에 관련 된 내용으로 Schema를 조금 다듬은 뒤 그것을 기반으로 구현을 하기 시작하며 메소드 간의 충돌이 줄어들어 설계의 중요성에 대하여 느끼게 되었습니다. 제가 가장 애를 먹었던 문제는 바로

TakeClass 와 관한 내용이었는데, 테이블이 여러가지 측면으로 해석될 수 있게 설계하여 어려움이 있었지만, 데이터 스키마를 다듬어 본 결과, 스키마 자체로 구분하는 것 말고 Grade 값을 기준으로 Logical 하게 의미단위를 쪼개는 행위를 DML 단에서 수행하여 이를 조금 더 명확하게 나누어 구현할 수 있었고, 문제를 해결할 수 있었습니다.