Spring Security

2 개인 보안 노트 서비스 만들기

개인 보안 노트 서비스 만들기

Github

git clone https://github.com/kker5/spring-security-practice

프로젝트 목적

SpringSecurity가 필요한 상황을 경험해보고 직접 구현해 봅니다.
구현 내용을 토대로 SpringSecurity 아키텍처를 이해합니다.

프로젝트 요구사항

1. 유저는 본인의 노트(게시글)만 저장하고 삭제하고 볼 수 있습니다.
2. 다른 유저의 노트는 볼수 없습니다.
3. 어드민은 관리 차원에서 유저들의 노트 제목 리스트는 볼 수 있지만 내용은 볼 수 없습니다.
4. 어드민은 공지사항을 작성할 수 있고 일반 유저들은 이 공지사항을 볼 수 있습니다.

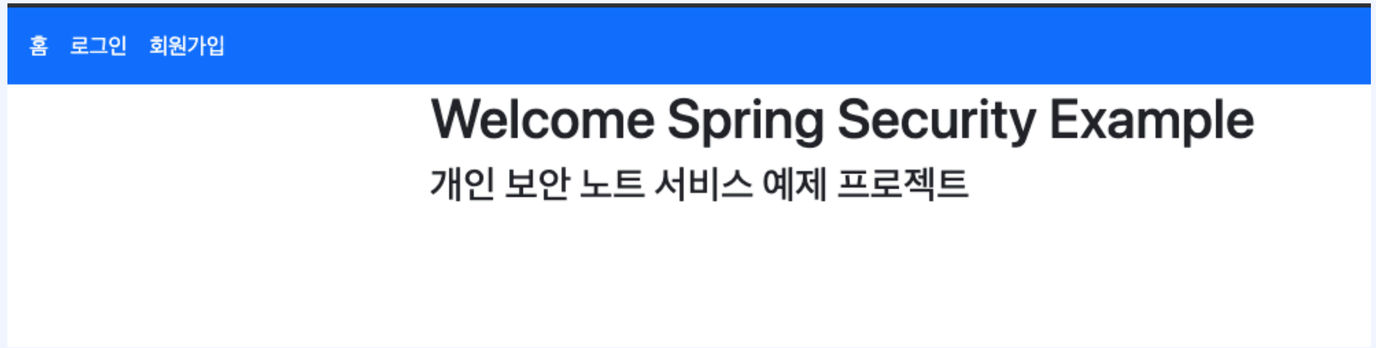
기술 스택

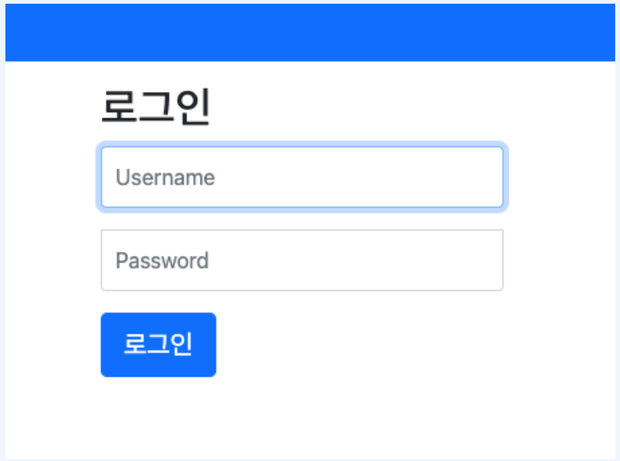
1. Spring WebMVC
2. Spring Security
3. Thymeleaf
4. Lombok
5. Spring Jpa
6. H2
7. Gradle

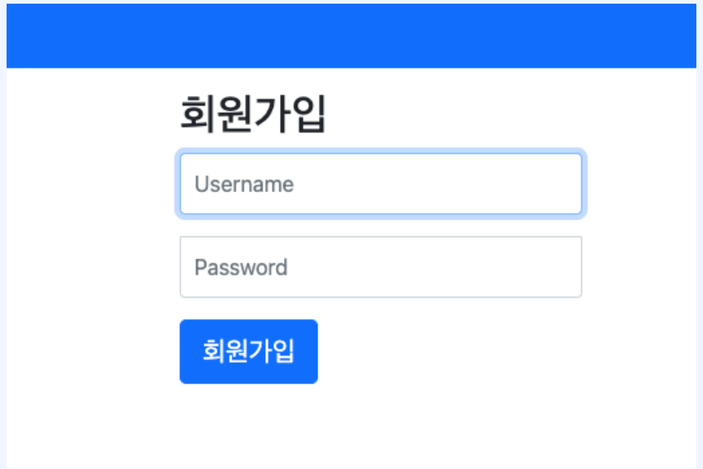
개인 보안 노트 서비스 만들기

로그인 없이 이용할 수 있는 서비스

홈
로그인
회원가입







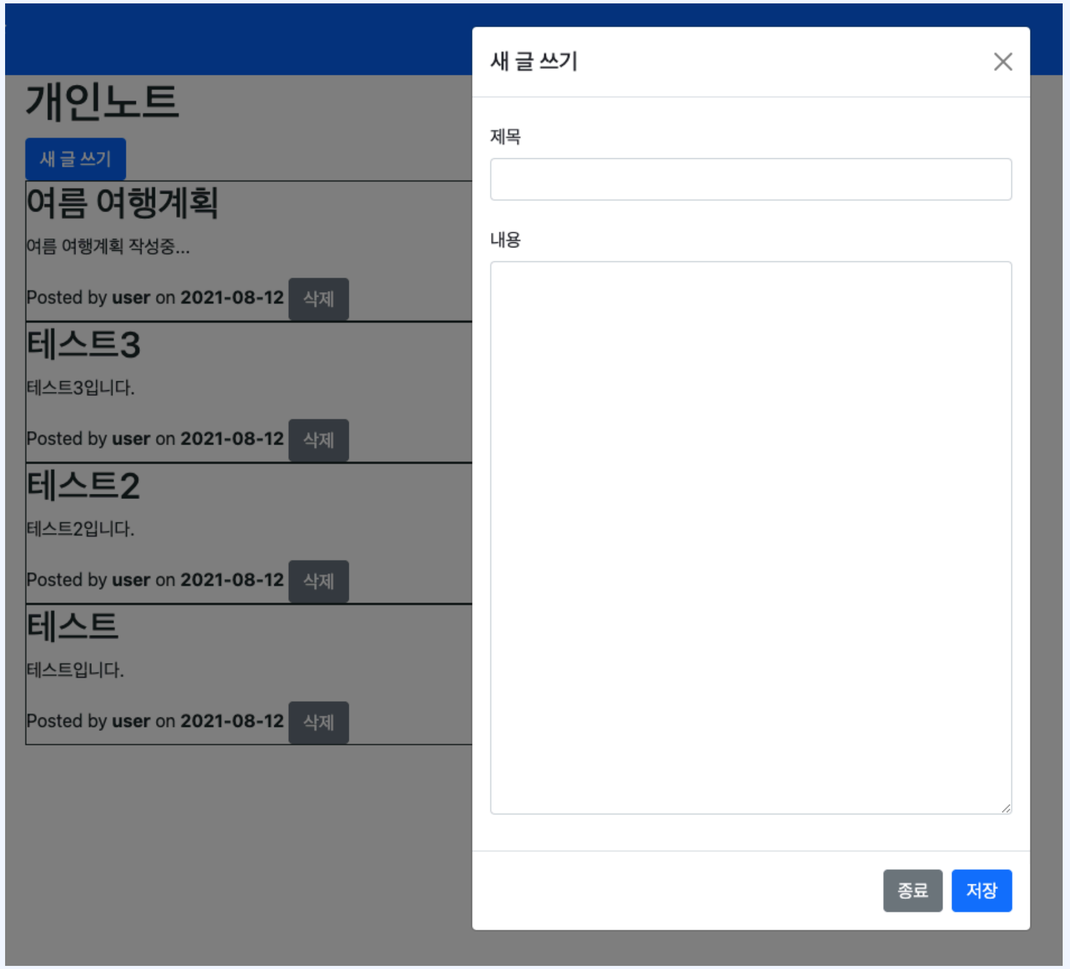
개인 보안 노트 서비스 만들기

유저가 이용할 수 있는 서비스

개인노트
본인이 쓴 노트 확인
노트 작성
노트 삭제

공지사항
공지사항 확인

계정
로그아웃



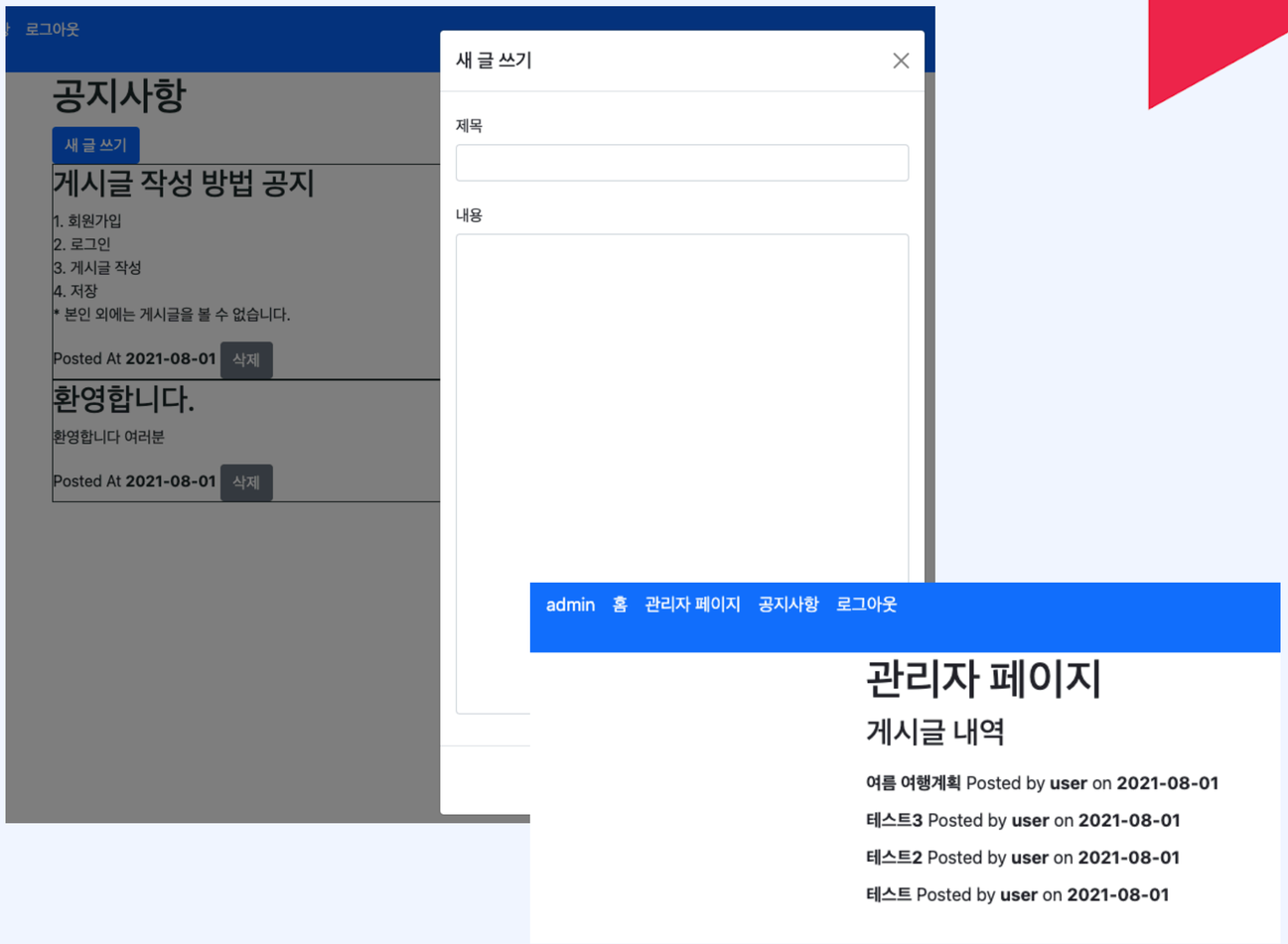
개인 보안 노트 서비스 만들기

어드민이 이용할 수 있는 서비스

관리자페이지
노트 생성 히스토리

공지사항
공지사항 확인
공지사항 작성
공지사항 삭제

계정
로그아웃



개인 보안 노트 서비스 만들기

프로젝트 생성

Gradle + Java

git 설정 추가

필요한 경우만 gitignore추가

build.gradle 추가

의존성 추가

SpringBootApplication.class 추가

main함수 추가

Thymeleaf

Thymeleaf 의 특징

서버사이드 템플릿 엔진입니다.

클라이언트가 동적으로 그리는 방식이 아니라 서버가 모든 html을 그려서 내려주는 방식입니다.

서버에서 (유저마다 동적으로 달라질수 있는)데이터들을 구해서 미리 정의된 템플릿에 넣고 서버에서 직접 html을 그려서 클라이언트(브라우저)에게 전달합니다.

th:text 와 같은 문법으로 html에 값을 주입하기때문에 매우 직관적이며 진입장벽이 낮습니다.

html로 작성하기 때문에 서버를 띄울필요 없이 바로 브라우저에서 확인할 수 있습니다.

우리 프로젝트에서 Thymeleaf 을 사용하면 좋은 점

Spring WebMVC와 통합이 비교적 쉽습니다.

spring-boot-starter-thymeleaf 를 따로 만들어 두었을 정도로 스프링과의 호환성이 매우 좋습니다.

간단한 코드로 Spring Security 결과물을 바로 확인할 수 있습니다.

Thymeleaf 문법

${name}

변수 name값 불러오기

```
@RequestMapping(value = "/example", method = RequestMethod.GET)
public String newPerson(Model model) {
    model.addAttribute("name", "김철수");
    return "example";
}
```

html 파일에서는 name이 있으면 name의 값이 들어가고 없으면 홍길동이 들어갑니다.

```
<p>당신의 이름은 <span th:text="${name}">홍길동</span> 입니다.</p>
```

th:with

변수값지정

```
<div th:with="temp=${name}" th:text="${temp}"></div>
```

th:text

text 수정

```
<div th:text="${name}">기본값</div>
```

Thymeleaf 문법

th:block / th:if / th:unless

If else와 유사함

```
<th:block th:if="${age < 30}"><p>당신은 30대가 아닙니다.</p></th:block>
<th:block th:unless="${age < 30}"><p>당신은 30대입니다.</p></th:block>
```

th:switch / th:case

Switch case와 유사함

```
<th:block th:switch="${name}">
  <p th:case="이철수"> 당신은 이씨 입니다.</p>
  <p th:case="김철수"> 당신은 김씨 입니다.</p>
</th:block>
```

th:each

반복문

```
<div th:each="data:${datas}">
  <h1 th:text="${data}"></h1>
</div>
```

Thymeleaf 문법

th:fragment

공통 layout을 나누고 사용함

```
<!-- 공통 헤더를 정의합니다. -->
<head th:fragment="commonHeader">
  <title>스프링 시큐리티 예제</title>
  <meta charset="UTF-8"/>
  <link
    rel="stylesheet"
    href="<https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css>"
  >
  <link
    rel="stylesheet"
    href="<https://use.fontawesome.com/releases/v5.0.6/css/all.css>"
  >
</head>
<!-- 위에서 정의한 fragment를 사용합니다. -->
<!-- commonHeader를 추가합니다. -->
<head th:insert="fragments.html::commonHeader"></head>
<!-- head를 commonHeader로 교체합니다. -->
<head th:replace="fragments.html::commonHeader"></head>
```

View 구현하기

메뉴바 (fragments.html)

권한에 따른 메뉴바 상태

인증 안됨

유저로 인증됨

어드민으로 인증됨

권한에 따른 활성화 방법

인증 받은 모든 사람

```
sec:authorize="isAuthenticated()"
```

인증 받지 못한 사람

```
sec:authorize="!isAuthenticated()"
```

관리자만

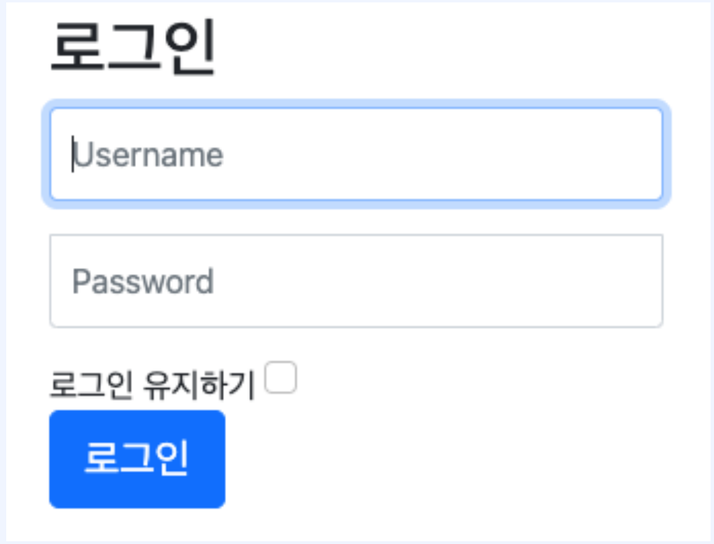
```
sec:authorize="hasAnyRole('ROLE_ADMIN')"
```

유저만

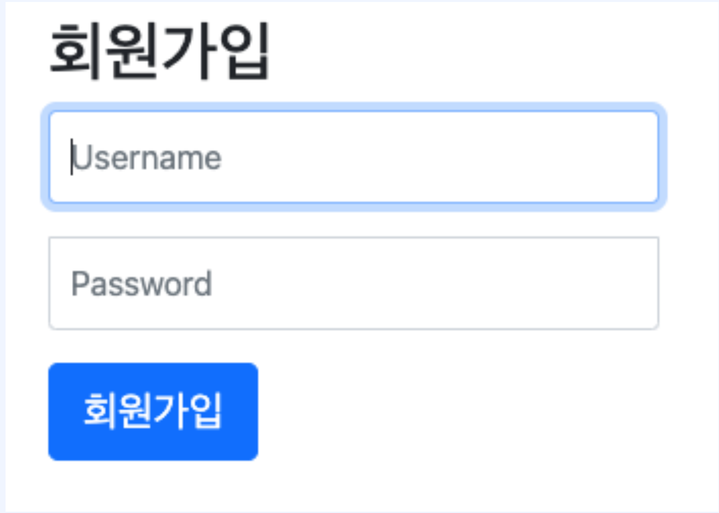
```
sec:authorize="hasAnyRole('ROLE_USER')"
```

View 구현하기

로그인 (login.html)



회원가입 (signin.html)



View 구현하기

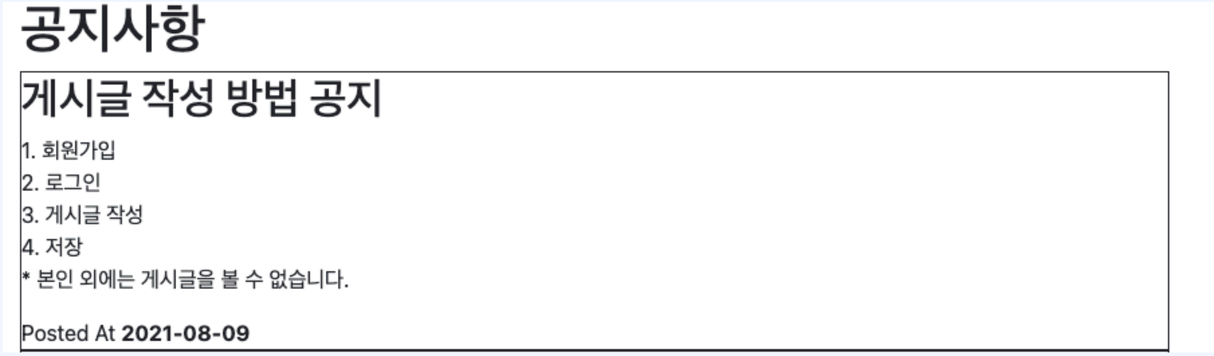
노트 (note/index.html)

View 구현하기

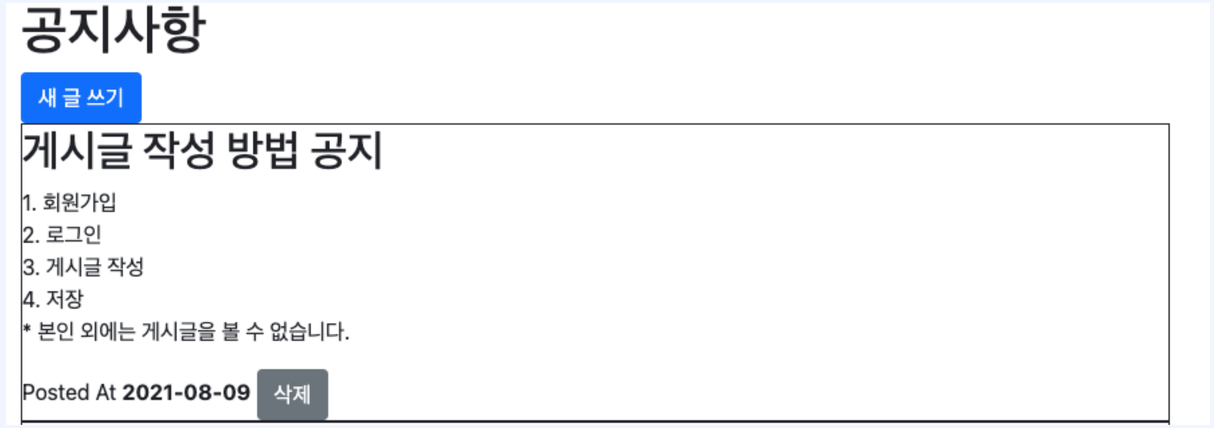
공지사항 (notice/index.html)

어드민이 공지하면 유저가 볼 수 있는 페이지

유저로 인증됨



어드민으로 인증됨



View 구현하기

관리자 페이지

유저가 등록한 노트들의 히스토리를 확인할 수 있는 페이지

노트 서비스 로직 구현하기

프로젝트 구조

config	JpaAuditorConfig	Jpa Auditor 설정
	InitializeConfig	최초 노트, 유저 추가
	MvcConfig	Mvc 설정
	PasswordEncoderConfig	Password Encoder 추가
	SpringSecurityConfig	스프링 시큐리티 설정
notice	Notice	공지사항 Entity
	NoticeController	공지사항 Controller
	NoticeRepository	공지사항 Repository
	NoticeService	공지사항 Service
note	Note	노트 Entity
	NoteController	노트 Controller
	NoteRegisterDto	노트 등록 Dto
	NoteRepository	노트 Repository
	NoteService	노트 Service
admin	AdminCotroller	관리자 Controller
user	AlreadyRegisteredUserException	등록된 유저에 대한 Exception
	SignUpController	회원가입 Controller
	User	유저 Entity
	UserRegisterDto	회원가입 Dto
	UserNotFoundException	유저를 찾을 수 없을 때 Exception
	UserRepository	유저 Repository
	UserService	유저 Service

노트 서비스 로직 구현하기

UserDetails

```
public interface UserDetails extends Serializable {
    Collection<? extends GrantedAuthority> getAuthorities();
    String getPassword();
    String getUsername();
    boolean isAccountNonExpired();
    boolean isAccountNonLocked();
    boolean isCredentialsNonExpired();
    boolean isEnabled();
}
```

User

```
public class User implements UserDetails {
    private Long id;
    private String username;
    private String password;
    private String authority;
```

노트 서비스 로직 구현하기

UserService

```
public class UserService {
    /**
     * 유저 등록
     */
    public User signup(
        String username,
        String password
    ) {
      }
    /**
     * 관리자 등록
     */
    public User signupAdmin(
        String username,
        String password
    ) {
      }
    public User findByUsername(String username) {
    }
}
```

SignUpController

```
@RequestMapping("/signup")
public class SignUpController {
    private final UserService userService;
    @GetMapping
    public String signup() {
        return "signup";
    }
    @PostMapping
    public String signup(
        @ModelAttribute UserRegisterDto userDto
    ) {
        userService.signup(userDto.getUsername(), userDto.getPassword());
        return "redirect:login";
    }
}
```

노트 서비스 로직 구현하기

Note

```
public class Note {
    private Long id;
    private String title;
    @Lob
    private String content;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_ID")
    private User user;
```

NoteService

```
public class NoteService {
    public List<Note> findByUser(User user) {
    }
    public Note saveNote(User user, String title, String content) {
    }
    public void deleteNote(User user, Long noteId) {
    }
}
```

노트 서비스 로직 구현하기

Notice

```
public class Notice {
    private Long id;
    private String title;
    @Lob
    private String content;
    @CreatedDate
    private LocalDateTime createdAt;
    @LastModifiedDate
    private LocalDateTime updatedAt;
}
```

NoticeService

```
public class NoticeService {
    public List<Notice> findAll() {
    }
    public Notice saveNotice(String title, String content) {
    }
    public void deleteNotice(Long id) {
    }
```

노트 서비스 로직 구현하기

PasswordEncoderConfig

Password를 암호화할 수 있는 Encoder를 Bean으로 등록

SpringSecurityConfig ★

Security 상세 설정

Security의 어떤 기능을 끄고 킬것인지

‘개인노트’ 페이지는 유저만 접근가능

‘어드민’ 페이지는 관리자만 접근가능

‘공지사항’ 추가, 삭제는 관리자만 가능

로그인 & 로그아웃 설정

유저 관련 설정

SpringSecurity가 유저 정보를 가져와야할 때 어떻게 가져올 수 있는지 정의

노트 서비스 로직 구현하기

MvcConfig

WebMvcConfigurer

ViewController 추가

JpaAuditorConfig

@EnableJpaAuditing 추가

InitializeDefaultConfig

매번 회원가입하기 어려워서 기본 유저와 관리자 등록

유저등록, note 4개 등록

어드민등록, 공지사항 2개 등록

테스트 구현하기

AdminControllerTest	어드민 컨트롤러 테스트
NoticeControllerTest	공지사항 컨트롤러 테스트
NoteControllerTest	개인노트 컨트롤러 테스트
SignUpControllerTest	회원가입 컨트롤러 테스트
NoteServiceTest	개인노트 서비스 테스트
UserServiceTest	회원가입 서비스 테스트

테스트 구현하기

BDDAssertions

우리 프로젝트에서는 BDDAssertions 로 테스트 결과 (then절) 를 검증합니다.

```
then(user.getId()).isNotNull(); // id가 NotNull인지 검증
then(user.getUsername()).isEqualTo("user123"); // 유저명이 user123인지 검증
then(user.getPassword()).startsWith("{bcrypt}"); // 패스워드가 {bcrypt}로 시작하는지 검증
then(user.getAuthorities()).hasSize(1); // Authorities가 1개인지 검증
then(user.isAdmin()).isFalse(); // 어드민 여부가 False인지 검증
```

테스트 구현하기

MockMvc

우리 프로젝트에서 컨트롤러 테스트를 할때 씁니다.
Spring-Test에 포함되어 있으면 Controller테스트를 할때 보편적으로 가장 많이 씁니다.

```
@SpringBootTest
@ActiveProfiles(profiles = "test")
@Transactional
class AdminControllerTest {
    private MockMvc mockMvc;
    @BeforeEach
    public void setUp(@Autowired WebApplicationContext applicationContext) {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(applicationContext)
                .build();
     }
    @Test
    void getNoteForAdmin() throws Exception {
        mockMvc.perform(get("/admin"))
                .andExpect(status().is2xxSuccessful());
    }
```

테스트 구현하기

MockMvc

perform

요청을 전송하는 역할을 합니다. 결과로 ResultActions 를 반환합니다.

get, post, put, delete

perform() 안에 넣어서 요청할 http method를 정합니다. 인자로 경로를 적어줍니다.

ex) perform(get(“/hello”))

params

Key value 파라미터를 전달할수 있습니다.

여러 개일 때는 params, 한개면 param을 사용합니다.

andExpect

응답을 검증합니다.

ex) andExpect(*status*().isBadRequest())

status() 상태를 검증합니다. isOk(200), isNotFound(404)

view() 응답으로 받은 뷰 이름을 검증합니다.

redirect() 응답으로 받은 redirect를 검증합니다.

content() 응답 body를 검증합니다.

andDo

일반적으로 해야할 일을 표현합니다.

andDo(print()) 하면 결과를 print합니다.

테스트 구현하기

스프링 시큐리티의 테스트는 일반적인 컨트롤러 테스트와 약간 다릅니다.
그 이유는 유저가 로그인을 한 상태로 서비스를 이용했다는 가정하에
테스트를 진행할 수 있어야 하기 때문입니다.
시큐리티 테스트를 사용하면 테스트를 실행 전에 원하는 유저를 마치 로그인한 것 처럼 설정할 수 있습니다.

가짜 유저를 세팅하는 방법은 여러가지가 있지만 우리 프로젝트에서는 3가지 방법으로 구현하였습니다.

@WithMockUser

NoticeControllerTest에서 사용하였습니다.
특정 사용자가 존재하는 것처럼 테스트 진행할 수 있습니다.

@WithUserDetails

NoteControllerTest에서 사용하였습니다.
앞서 구현한 UserDetailsService 혹은 테스트를 위해 별도로 구현한 UserDetailsService를 참고해서 사용자를 가짜로 로그인할 수 있습니다.

~.with

AdminControllerTest에서 사용하였습니다.
직접 사용자를 mockMvc에 지정하는 방식입니다.