

# 스프링 부트 웹 프로젝트

chapter10

## 로그인

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

# Contents

part.1

스프링 시큐리티 소개

part.2

스프링 시큐리티 설정

part.3

사용자 인증

part.4

사용자 정보 출력하기

# 스프링 시큐리티 개념 프로젝트 준비

해당 프로젝트는 기존 코드를 활용할 예정입니다.

이전에 작성한 "project9"을 복사하여 "project10"를 새로 생성하세요.

 Project9 Project10

프로젝트에서 에러가 발생했으면, setting.gradle 파일을 열어서 프로젝트의 이름을 변경하세요.

그리고 gradle을 리프레쉬하면 에러가 사라집니다.

```
▼ > project10 [spring-study main]
  > > src/main/java
  > > src/main/resources
  > > src/test/java
  > > src/main/generated
  > IRF System Library [JavaSE-17]
```

```
settings.gradle x
1 rootProject.name = 'project9'
```



```
settings.gradle x
1 rootProject.name = 'project10'
```

# 스프링 시큐리티 개념 프로젝트 준비

시큐리티 관련 라이브러리를 추가하세요.

build.gradle

```
...  
implementation 'org.springframework.boot:spring-boot-starter-security'  
implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6'  
testImplementation 'org.springframework.security:spring-security-test'
```

PPT에서 코드를  
복사하여 붙여넣으세요.

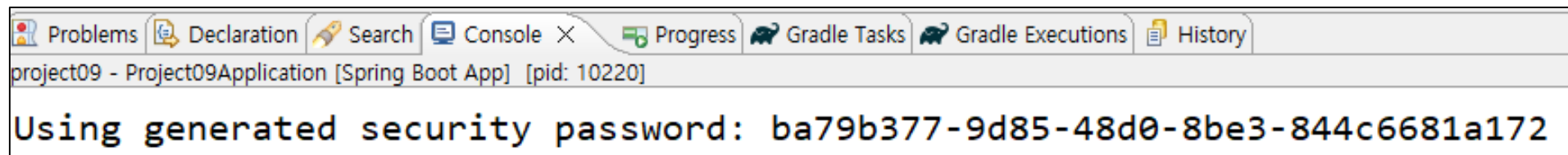
기존 테이블과 단위테스트를 모두 삭제하세요.

# 스프링 시큐리티 개념 프로젝트 준비

프로젝트를 실행한 후, 콘솔창에 비밀번호가 출력되었는지 확인하세요.

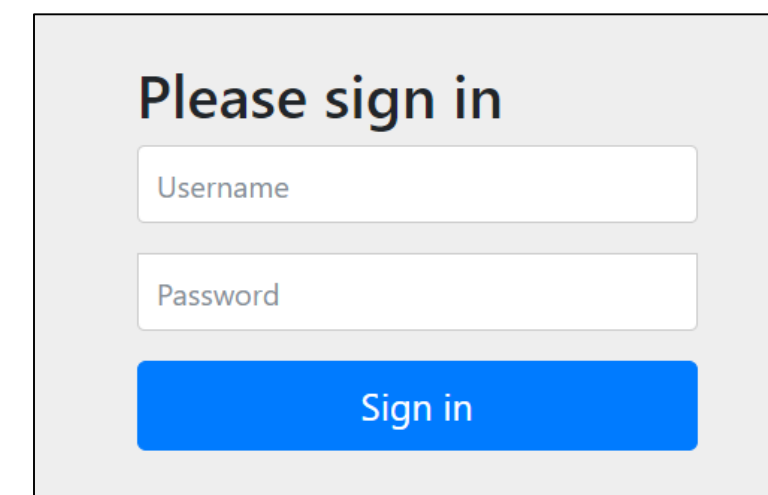
이 비밀번호는 DB에 사용자 계정이 하나도 없을 때 사용하는 임시 비밀번호 입니다.

1. 프로젝트를 실행한다.
2. 콘솔창에 비밀번호가 출력되었는지 확인한다.
3. 브라우저에서 localhost:8080 주소로 접속한다.
4. 메인화면 대신 로그인화면으로 이동하는지 확인한다.
5. user 계정을 사용하여 로그인한다. (username: "user", password: "임시비밀번호")



The screenshot shows an IDE console window with tabs for Problems, Declaration, Search, Console, Progress, Gradle Tasks, Gradle Executions, and History. The console output for 'project09 - Project09Application [Spring Boot App] [pid: 10220]' displays the message: 'Using generated security password: ba79b377-9d85-48d0-8be3-844c6681a172'.

콘솔창



The screenshot shows a login form with the title 'Please sign in'. It contains two input fields: 'Username' and 'Password'. Below the fields is a blue button labeled 'Sign in'.

로그인 화면

# 스프링 시큐리티 개념 Spring Security란?

프로그램의 보안 기능을 구현하기 위한 라이브러리이다.  
사용자 인증, 권한 부여, 다양한 보안 설정을 쉽게 처리할 수 있다.



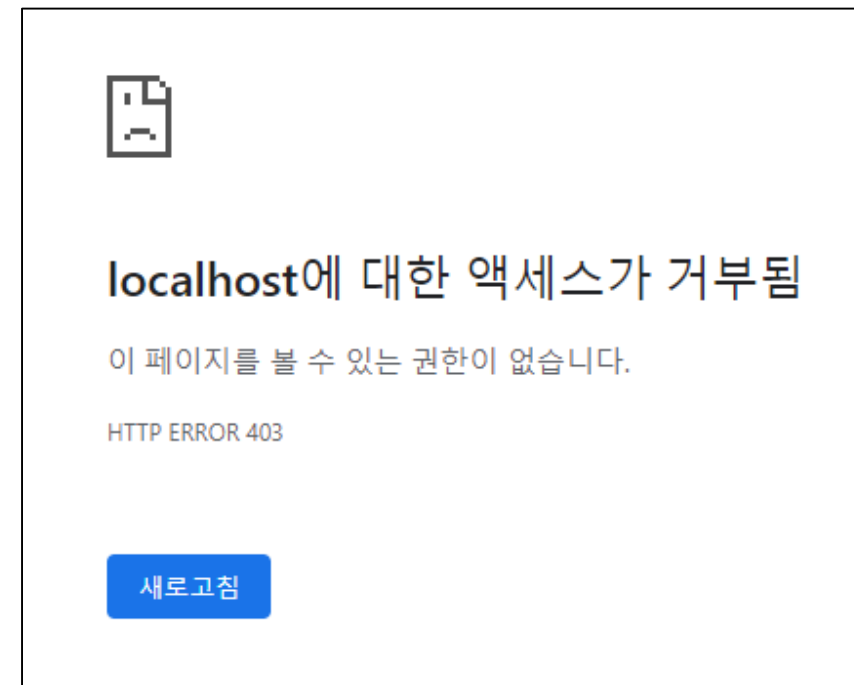
### Login

아이디

패스워드

Login

사용자 인증



권한이 없으면 접근 거부

# 스프링 시큐리티 개념 Spring Security의 필요성

프로그램의 보안을 위해 사용자의 권한을 확인해야 한다.

모든 컨트롤러 메소드에 권한 확인 로직을 넣어야 하지만, Spring Security를 사용하면 쉽게 처리할 수 있다.

Spring Security는 중앙에서 보안을 설정하기 때문에, 각 컨트롤러에 일일이 보안 로직을 추가할 필요가 없다.

```
/* 등록폼화면 */
@GetMapping("/register")
public String registerForm() {

    return 사용자 권한 확인
}

/* 등록처리 */
@PostMapping("/register")
public String register(BoardDto boardDto, RedirectAttributes rttr) {

    BoardDto boardDto = service.register(boardDto);
    rttr.addAttribute("registerNo", registerDto.getNo());
    return "redirect:/board/list";
}
```

각 컨트롤러에 보안 로직을 넣은 경우

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) {

        http.authorizeHttpRequests()
            .requestMatchers("/register").permitAll()
            .requestMatchers("/assets/*", "/css/*").permitAll()
            .requestMatchers("/").authenticated()
            .requestMatchers("/board/*").hasAnyRole("ADMIN", "USER");
    }
}
```

스프링 시큐리티로 보안 로직을 설정한 경우

# 스프링 시큐리티 개념 Spring Security의 주요 기능

---

## 1. 사용자 인증

로그인 여부 및 회원 등급 확인

폼 기반 로그인, OAuth2, JWT 토큰 등 다양한 인증 방법 지원

## 2. 권한 부여

사용자 접근 권한 관리

## 3. 보안 설정

CSRF(Cross-Site Request Forgery) 방어 기능

세션 만료 기간 설정

CORS(Cross-Origin Resource Sharing) 설정을 통한 크로스 도메인 요청 제어



# 스프링 시큐리티 개념 Spring Security의 주요 기능

스프링 시큐리티의 메소드

메소드 또는 객체	설명
hasRole( [role] )	해당 권한이 있으면 허용
hasAnyRole( [role,role2])	여러 권한들 중에서 하나라도 해당하는 권한이 있으면 허용
permitAll	모든 사용자에게 허용
denyAll	모든 사용자에게 거부
authenticated()	인증된 사용자면 허용
principal	현재 사용자 정보 (인증객체)

# 스프링 시큐리티 설정 사용자 비밀번호와 암호화

## 사용자 비밀번호 암호화

사용자의 개인정보를 보호하기 위해 비밀번호는 안전하게 저장해야 한다.

비밀번호는 암호화 엔진을 사용하여 해시코드로 변환한 후, 데이터베이스에 암호화하여 저장한다.



id	mod_d...	reg_d...	name	password
user1	2023-...	2023-...	둘리	1234
user2	2023-...	2023-...	또치	1234
user3	2023-...	2023-...	도우너	1234

잘못된 비밀번호

password
\$2a\$10\$AB9rUDITtasWMd9IsosSc..kFHatuoXyB...
\$2a\$10\$xD/qXNXvxxPmjARyj4.l/OVFp1ep2oraeL...
\$2a\$10\$M04oCqeLG2j8G5INqgkZCO/U379ecAm...

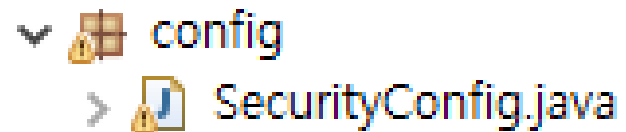
올바른 비밀번호

# 스프링 시큐리티 설정 사용자 비밀번호와 암호화

## 시큐리티 설정 클래스 만들기

프로젝트에 맞는 시큐리티 설정을 추가하기 위해, SecurityConfig 클래스를 생성한다.  
시큐리티 설정을 하기 위해 @EnableWebSecurity 어노테이션을 설정한다.

패키지



```
▼ config  
  > SecurityConfig.java
```

SecurityConfig

```
@Configuration  
@EnableWebSecurity  
public class SecurityConfig{  
}
```

# 스프링 시큐리티 설정 사용자 비밀번호와 암호화

## 패스워드 인코더 등록

패스워드 인코더를 등록하는 메소드를 추가하여, 사용자 패스워드를 암호화하는데 사용한다.  
이 메소드에 @Bean 어노테이션을 붙여 스프링 컨테이너에 빈으로 등록한다.

SecurityConfig

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

# 스프링 시큐리티 설정 사용자 패스워드와 암호화

## 패스워드 인코더 테스트

단위테스트를 통해 비밀번호를 암호화하고, 암호화된 문자열이 "1234"와 일치하는지 확인한다.

PasswordTest

```
@Test
public void 암호화테스트() {

    String password = "1234";

    String enPw = passwordEncoder.encode(password);

    System.out.println("enPw: " + enPw);
}
```

# 스프링 시큐리티 설정 사용자 패스워드와 암호화

## 서비스에서 패스워드 암호화 처리

회원 등록 메소드를 수정하여 패스워드 인코더를 사용해 회원의 패스워드를 암호화 한 후, 변경된 회원정보를 다시 저장한다.

### MemberServiceImpl

```
@Override
public boolean register(MemberDTO dto) {
    PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    String hashpassword = passwordEncoder.encode(entity.getPassword());
    entity.setPassword(hashpassword);
    repository.save(entity);
    ...
}
```

### 회원 테이블

id	name	role	password
admin	도우너	ROLE_ADMIN	\$2a\$10\$6/YTDarz/9CllpXFfMusSOvK9k
user1	둘리	ROLE_USER	\$2a\$10\$uhaP5nAUfnRxiwlfJ/PVN.NqEI
user2	또치	ROLE_USER	\$2a\$10\$Dwl0e9EcNfD58xvJXMiqSur2.

# 스프링 시큐리티 설정 사용자 비밀번호와 암호화

## 서비스 테스트

단위테스트를 통해 새로운 회원을 등록하고, 비밀번호가 암호화되었는지 확인한다.

MemberServiceTest

```
@Test
public void 회원등록() {

}
```

회원 테이블

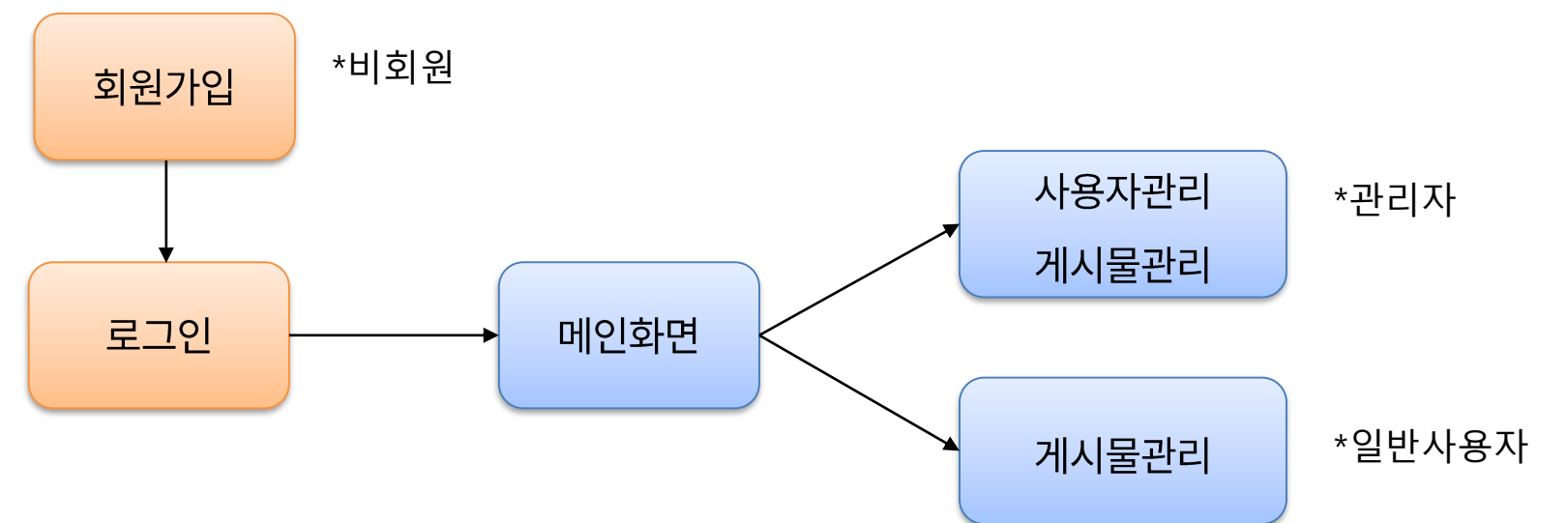
id	name	password
user1	둘리	\$2a\$10\$9IUDWXLrietZQat8

# 스프링 시큐리티 설정 메뉴 접근 권한 설계

관리자, 일반사용자, 비회원(로그인하지 않은 사용자) 세 가지 등급에 따라 접근할 수 있는 메뉴를 설계한다.

메뉴 접근 권한

메뉴	관리자	일반사용자	아무나
회원가입	O	O	O
로그인	O	O	O
회원관리	O	X	X
게시물관리	O	O	X





# 스프링 시큐리티 설정 회원 관리 URL 변경

## 회원 컨트롤러에서 주소 처리

스프링 시큐리티는 URL 주소를 기반으로 접근 권한을 설정할 수 있다.

회원가입은 아무나 할 수 있지만, 회원 조회는 관리자만 할 수 있다.

시큐리티 설정 클래스에서 해당 URL에 대한 접근 권한을 설정해야 한다.

### MemberController

```
public class MemberController {  
    @GetMapping("/member/list")  
    public void list()  
    @GetMapping("/register")  
    public String register()  
    @PostMapping("/register")  
    public String registerPost()  
    @GetMapping("/member/read")  
    public void read()  
}
```

메뉴 접근 권한

메뉴	관리자	일반사용자	아무나
회원가입	O	O	O
회원관리	O	X	X

등록 메소드의 중간 경로를 없애주세요!

# 스프링 시큐리티 설정 회원 관리 URL 변경

---

## 회원가입 화면에서 주소 처리

폼의 action 속성에 등록 처리 주소를 변경한다.

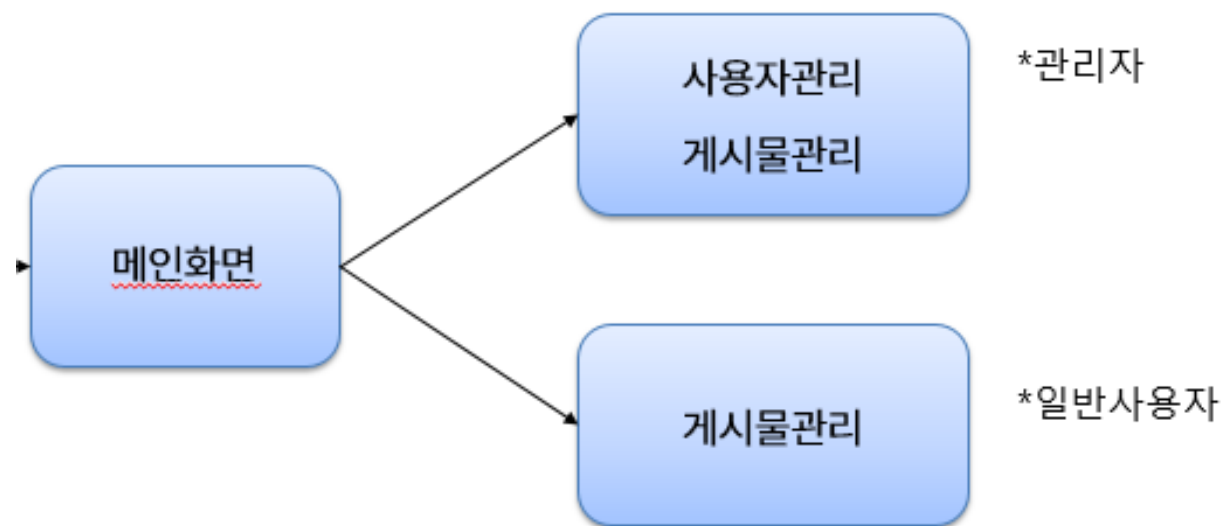
register.html

```
<form th:action="@{/register}" th:method="post">
```

# 스프링 시큐리티 설정 사용자 권한

## 사용자 권한

스프링 시큐리티에서는 사용자가 수행할 수 있는 작업을 구분하기 위해 권한(role)을 부여한다.  
이를 위해 사용자 정보에 권한 필드를 추가한다.



사용자 테이블

id	name	role
admin	도우너	ROLE_ADMIN
user1	둘리	ROLE_USER
user2	또치	ROLE_USER

# 스프링 시큐리티 설정 사용자 권한

---

## 회원 DTO와 엔티티에 권한 추가

### MemberDTO

```
public class MemberDTO {  
    ...  
    String role;  
}
```

### Member

```
public class Member {  
    ...  
    String role;  
}
```

# 스프링 시큐리티 설정 사용자 권한

## 회원 서비스에서 권한 처리

변경된 구조에 따라 변환 메소드를 수정한다.

dtoToEntity() 메소드에서는 DTO객체에서 엔티티객체로 권한을 옮긴다.

entityToDto() 메소드에서는 엔티티 객체에서 DTO객체로 권한을 옮긴다.

### MemberService

```
default MemberDTO entityToDto(Member entity) {  
    MemberDTO dto = MemberDTO.builder()  
        .role(entity.getRole())  
        ...  
}
```

```
default Member dtoToEntity(MemberDTO dto) {  
    Member entity = Member.builder()  
        .role(dto.getRole())  
        ...  
}
```

# 스프링 시큐리티 설정 사용자 권한

## 회원가입 화면에서 권한 처리

회원가입 화면에 사용자 권한 필드를 추가한다.

라디오버튼을 사용하여 일반 사용자와 관리자 권한을 선택할 수 있도록 한다.

각 입력 필드의 value 속성에는 "ROLE\_USER"와 "ROLE\_ADMIN"을 입력한다.

스프링 시큐리티에서는 권한 이름이 "ROLE\_"으로 시작하는 규칙이 있다.

### register.html

```
<div class="form-group">
  <div class="form-check form-check-solid">
    <input class="form-check-input" type="radio" id="user" name="" value="" checked>
    <label class="form-check-label" for="user">일반사용자</label>
  </div>
  <div class="form-check form-check-solid">
    <input class="form-check-input" type="radio" id="admin" name="" value="">
    <label class="form-check-label" for="admin">관리자</label>
  </div>
</div>
```

PPT에서 코드를 복사하여  
이름 필드 아래에 붙여 넣으세요.

이름

☒ 일반사용자  
☐ 관리자

드로

# 스프링 시큐리티 설정 사용자 권한

## 회원 상세 화면에서 권한 처리

회원 상세 조회 화면에 권한 필드를 추가한다.

입력 필드를 사용하여 사용자 권한을 출력한다.

read.html

```
<!-- 등급 필드 -->
<div class="form-group">
  <label>등급</label>
  <input type="text" class="form-control" name="role" value="" readonly>
</div>
```

PPT에서 코드를 복사하여  
이름 필드 아래에 붙여 넣으세요.

이름

둘리

등급

ROLE\_USER

# 스프링 시큐리티 설정 필터체인이란?

## 필터체인이란?

요청이 서버에 오면 여러 필터 체인을 순차적으로 거쳐 여러 보안 로직을 처리하는 구조이다.

스프링 시큐리티에서 필터는 순차적으로 적용되어 요청을 검사하고 처리한다.

필터 (Filter): 요청이 서블릿에 도달하기 전에 실행되며, 보안 검증을 수행

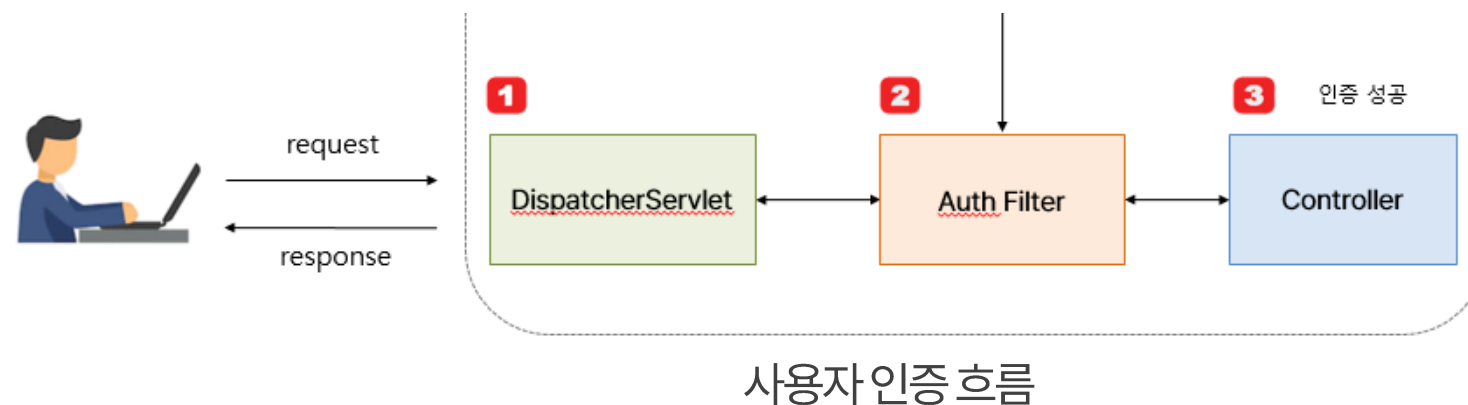
필터 체인 (Filter Chain): 여러 필터를 순차적으로 적용하는 구조

## 필터체인의 주요 기능

인증(Authentication): 사용자의 아이디와 비밀번호로 사용자가 누구인지 확인

인가(Authorization): 사용자가 특정 URL에 접근할 권한이 있는지 확인

로그아웃 처리: 사용자 로그아웃을 처리





# 스프링 시큐리티 설정 URL에 대한 접근제한 설정

스프링 시큐리티에 SecurityFilterChain을 정의하여 등록한다.

스프링 내부적으로 "ROLE\_USER"는 상수처럼 사용되며, 권한에 "ROLE\_"가 접두어로 붙는다.

1. 회원가입은 누구나 할 수 있다.
2. 메인화면은 로그인한 사용자만 접근할 수 있다.
3. 게시물 관리는 관리자와 일반사용자 모두 사용할 수 있다.
4. 회원관리는 관리자만 사용할 수 있다.

## SecurityConfig

```
public SecurityFilterChain filterChain(HttpSecurity http) {  
    http.authorizeRequests()  
        .requestMatchers("/register").permitAll()  
        .requestMatchers("/").authenticated()  
        .requestMatchers("/board/*").hasAnyRole("ADMIN","USER")  
        .requestMatchers("/member/*").hasRole("ADMIN");  
}
```

## 스프링 시큐리티 내부의 인증 매니저

```
public final class AuthorityAuthorizationManager<T> imple  
  
    private static final String ROLE_PREFIX = "ROLE_";
```

# 스프링 시큐리티 설정 로그인 및 로그아웃 설정

로그인과 로그아웃 기능을 처리한다.

시큐리티가 제공하는 로그인 화면이나 커스텀 로그인 화면을 사용할 수 있다.

커스텀 로그인 화면을 사용하려면 별도 설정이 필요하다.

메뉴바에 로그아웃 링크를 추가한다.

## SecurityConfig

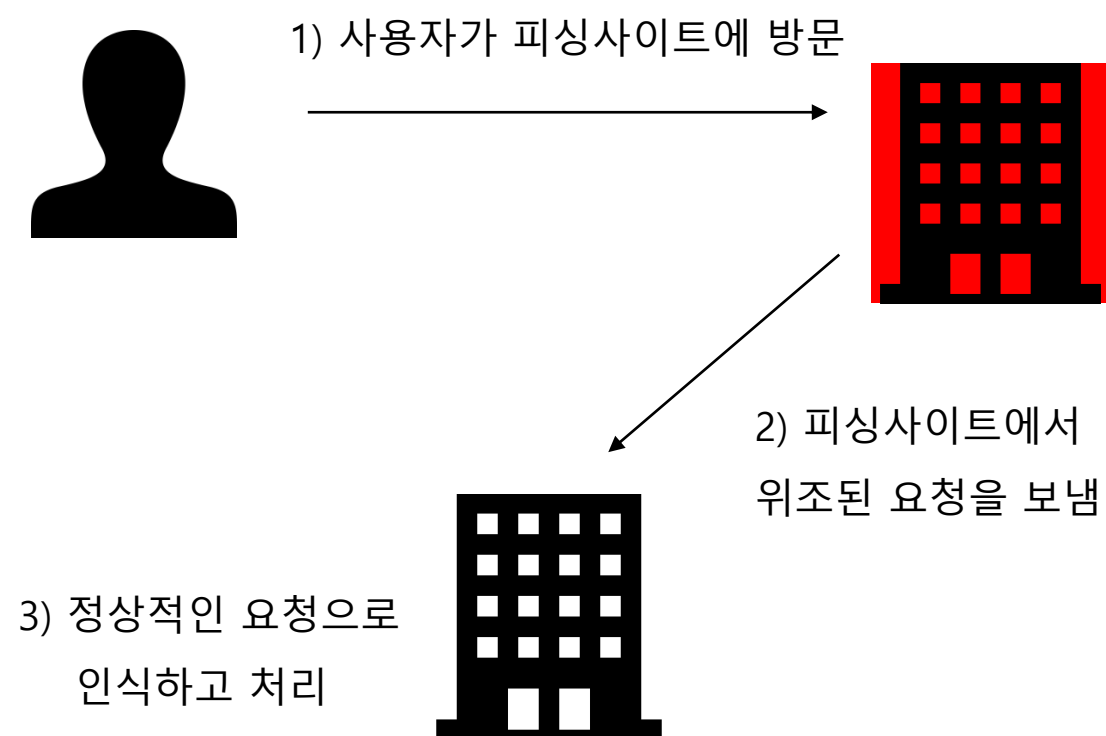
```
public SecurityFilterChain filterChain(HttpSecurity http) {  
    http.formLogin(withDefaults());  
    http.logout(withDefaults());  
}
```

## basic.html

```
<li class="nav-item"><a class="nav-link" href="/logout">Logout</a></li>
```

# 스프링 시큐리티 설정 CSRF 설정

스프링 시큐리티는 크로스 사이트 요청 위조(CSRF) 공격을 방어하기 위해 GET을 제외한 요청 방식을 기본적으로 막는다. POST, PUT, DELETE 요청 방식을 사용하려면 CSRF 토큰을 포함시켜야 한다. CSRC 토큰은 세션당 하나씩 생성되며, 계속 값이 변경된다. 매번 토큰의 값을 알아내야 해서 번거롭기 때문에, CRSF 기능을 비활성화 한다.



## 예시

사용자가 이 버튼을 클릭하면 자동으로 페이스북에 광고성 글을 올리게 된다..

```
<form action="http://facebook.com/api/content" method="post">
  <input type="hidden" value="여기 가입하면 돈 10만원 드립니다." />
  <input type="submit" value="Click Me"/>
</form>
```

## SecurityConfig

```
public SecurityFilterChain filterChain(HttpSecurity http) {
    http.csrf(csrf -> csrf.disable());
}
```

# 스프링 시큐리티 설정 회원가입 테스트

1. localhost:8080/register 회원가입 페이지로 이동한다.
2. 등급 입력 필드가 나타나는지 확인한다.
3. 새로운 회원을 등록한다.
4. 회원정보에 등급과 비밀번호가 저장되었는지 확인한다.

회원가입 화면

## Sign Up

아이디

패스워드

이름

☐ 일반사용자  
☒ 관리자

등록

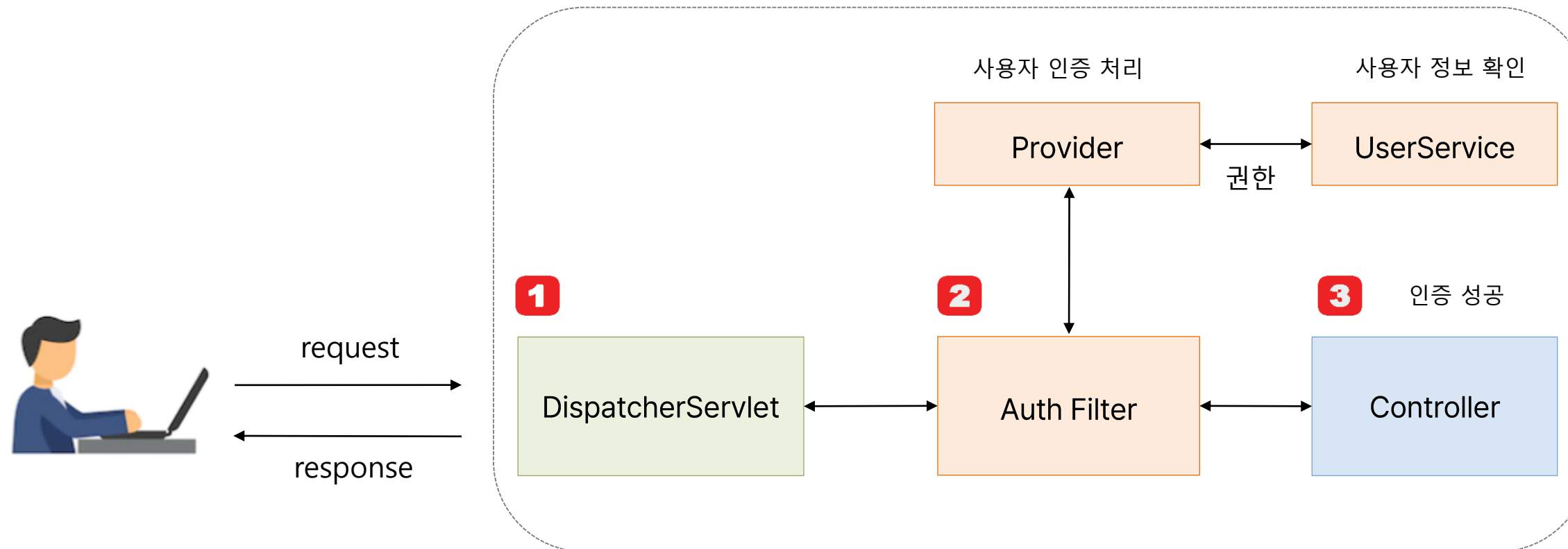
회원 테이블

id	name	role	password
admin	도우너	ROLE_ADMIN	\$2a\$10\$6/YTDarz/9CllpXFfMusSOvK9b
user1	둘리	ROLE_USER	\$2a\$10\$uhaP5nAUfnRxiwlfJ/PVN.NqEI
user2	또치	ROLE_USER	\$2a\$10\$Dwl0e9EcNfD58xvJXMiqSur2.2

# 사용자 인증

## 사용자 인증 흐름

1. 사용자가 서버에게 요청을 보낸다.
2. 인증 필터가 사용자의 접근권한이 있는지 확인한다.
3. 접근권한이 있다면 해당 컨트롤러에게 사용자 요청을 전달한다.



# 사용자 인증

## 인증객체와 인증서비스

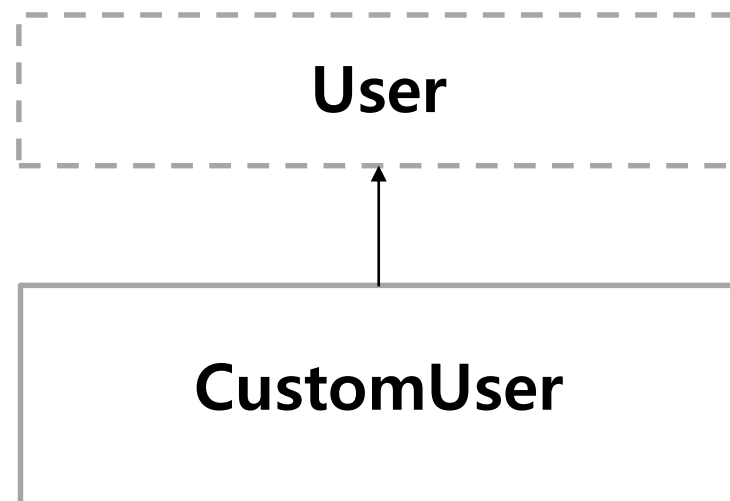
### 사용자 인증 객체

스프링 시큐리티는 인증 처리시 User라는 인증 객체를 사용한다.

이 객체는 사용자의 아이디, 비밀번호, 권한, 계정 사용가능 여부 등을 가지고 있다.

프로젝트에 맞게 User 객체를 확장하여 커스텀 인증 클래스를 만들 수 있다.

스프링 시큐리티의 User



사용자 인증 정보

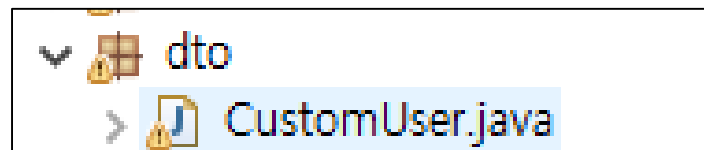
• this	CustomUser (id=151)
▢F accountNonExpired	true
▢F accountNonLocked	true
▼ ▢F authorities	Collections\$Unmodifiable
▼ ▢ [0]	SimpleGrantedAuthority
> ▢F role	"ROLE_ADMIN" (id=165)
▢F credentialsNonExpired	true
▢F enabled	true
> ▢ password	"\$2a\$10\$uHH8IP4bL9uuf
> ▢F username	"admin1" (id=110)

### 사용자 인증 객체

CustomerUser 라는 커스텀 클래스를 생성하고, 스프링 시큐리티의 User 클래스를 상속받는다.

부모클래스의 생성자를 호출하여 인증 객체를 생성하고, 권한은 시큐리티에서 사용하는 SimpleGrantedAuthority로 변환한다.

패키지



CustomerUser

```
public class CustomUser extends User {  
    public CustomUser(MemberDTO dto) {  
        super(dto.getId(), dto.getPassword(), Arrays.asList(new SimpleGrantedAuthority(dto.getRole())));  
    }  
}
```

# 사용자 인증

## 인증객체와 인증서비스

### 사용자 인증 서비스

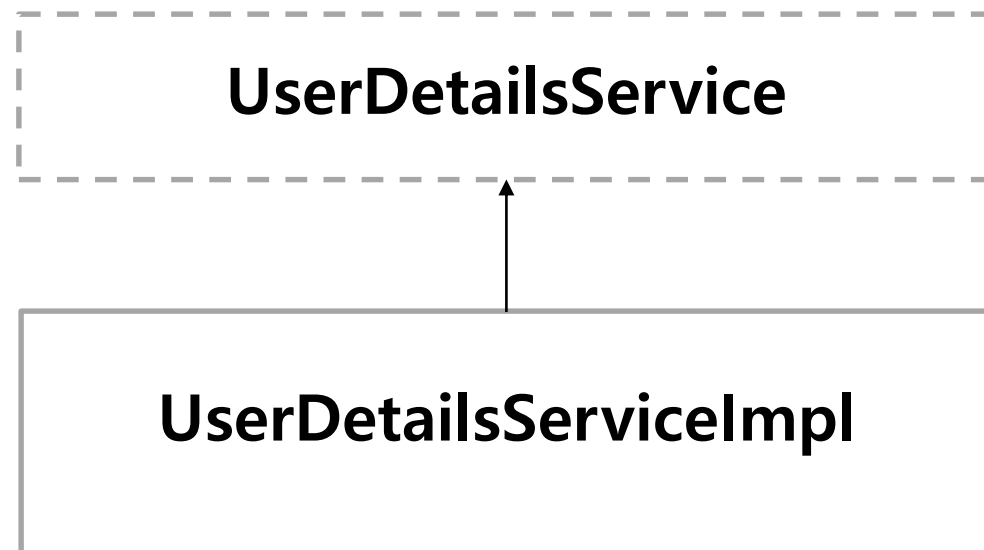
스프링 시큐리티는 인증 처리 시 UserDetailsService라는 인증 서비스를 사용한다.

UserDetailsService는 사용자 정보를 조회하여 반환하는 역할을 한다.

프로젝트에 맞게 UserDetailsService를 확장하여 커스텀 클래스를 만들 수 있다.

사용자 인증: 로그인 여부 및 회원 등급을 확인

스프링 시큐리티의 UserDetailsService



사용자 인증 처리

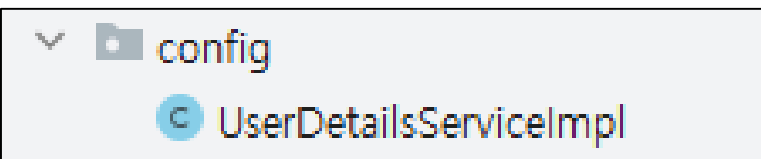
A user login form with a light gray background. It features the title "Please sign in" in bold. Below the title are two input fields: "Username" and "Password". At the bottom is a blue button labeled "Sign in".



### 사용자 인증 서비스

UserDetailsServiceImpl 이라는 커스텀 클래스를 생성하고, 스프링 시큐리티의 UserDetailsService 인터페이스를 상속받는다.

패키지



UserDetailsServiceImpl

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    MemberService service;

}
```

### 사용자 인증 서비스

부모 클래스에서 상속 받은 loadUserByUsername 메소드를 재정의한다.

시큐리티는 회원 아이디 대신 username이라는 용어를 사용한다.

이 메소드에서는 사용자 아이디를 확인하고, 아이디가 있다면 사용자 정보를 반환한다.

반환된 사용자 정보는 시큐리티에서 사용하는 인증객체(CustomerUser)로 변환한다.

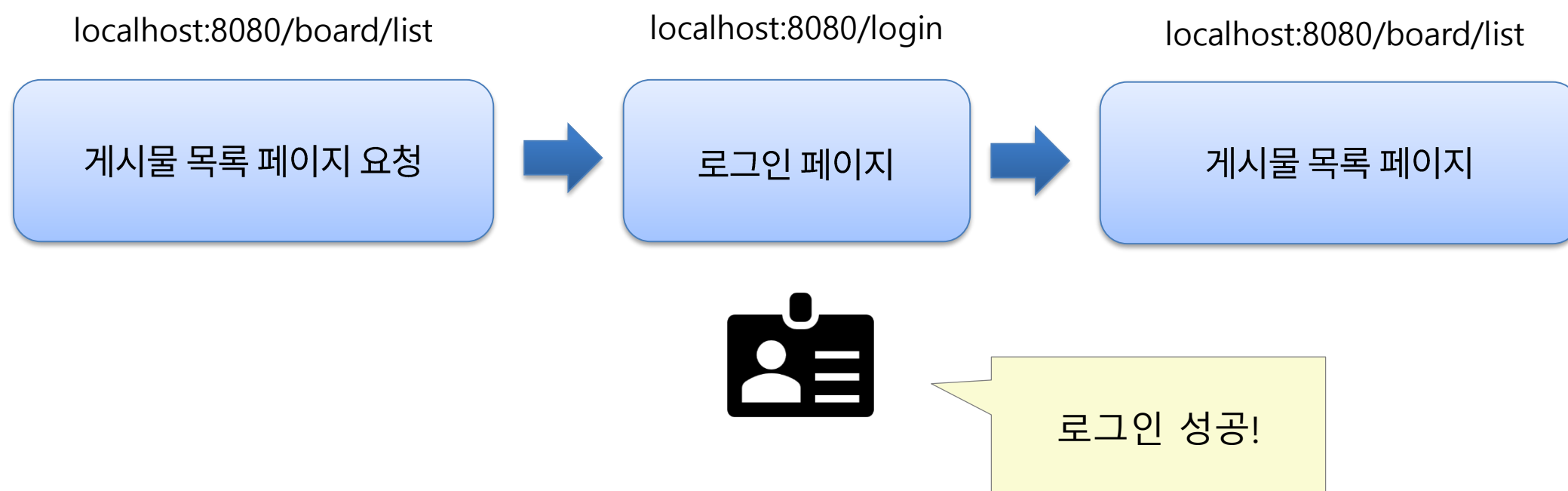
만약 해당아이디가 없다면 에러를 발생시킨다.

#### UserDetailsServiceImpl

```
@Override
public UserDetails loadUserByUsername(String userName) {
    MemberDTO dto = service.read(userName);
    if(dto == null) {
        throw new UsernameNotFoundException("");
    } else {
        return new CustomUser(dto);
    }
}
```

### 로그인 처리 과정

1. 게시물 목록을 요청하면, 먼저 로그인화면으로 이동한다.
2. 로그인을 한다.
3. 로그인에 성공하면 게시물 목록 화면으로 이동한다.



### 로그인 상태 관리와 쿠키

- HTTP는 상태를 유지하지 않는 stateless 프로토콜로, 이전에 보냈던 요청 정보는 기억하지 않는다.
- 그래서 다른 페이지로 이동할 때 로그인 상태를 유지하기 위해, 쿠키라는 기술은 사용한다.
- 로그인을 하면 'JSESSIONID'라는 쿠키가 생성하여 브라우저에 저장한다.
- 쿠키는 일정 시간 동안 유효하고, 그 기간 동안 로그인 상태를 유지할 수 있다.
- 로그인 상태가 유지되는 동안에는 페이지를 이동하더라도 로그인을 다시 요청하지 않는다.
- 로그아웃을 하면 쿠키와 세션이 초기화된다.

요소 콘솔 소스 네트워크 성능 메모리 애플리케이션 보안 Lighthouse Recorder Per		
애플리케이션	필터	
매니페스트	이름	값
Service Workers	uuid	6f4a05336809fb1ac88704762c60fb4d5488771
저장용량	JSESSIONID	74D82E298780C5288427817CA25A1007
저장용량		
로컬 스토리지		
세션 스토리지		
IndexedDB		
웹 SQL		
쿠키		
http://localhost:8080		

#### 요청 헤더

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/svg+xml,\*/\*;q=0.8

Accept-Encoding: gzip, deflate, br

Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7

Cache-Control: max-age=0

Connection: keep-alive

Cookie: JSESSIONID=74D82E298780C5288427817CA25A1007

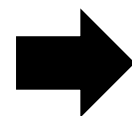
Host: localhost:8080

서버에서는 사용자 정보를 저장하기 위해 세션ID를 생성하고, 클라이언트와 연결한다.

이 세션ID는 클라이언트의 쿠키에 저장되며, 페이지를 이동할 때 브라우저는 쿠키에 저장된 세션 ID를 전송하여 세션을 유지한다.

로그아웃을 하면 세션ID가 무효화되고, 다시 로그인하면 새로운 세션이 생성된다.

Board List Page		게시물 등록
#	제목	
Name	Value	
JSESSIONID	A76251736B8A56016B180A944116A9B0	

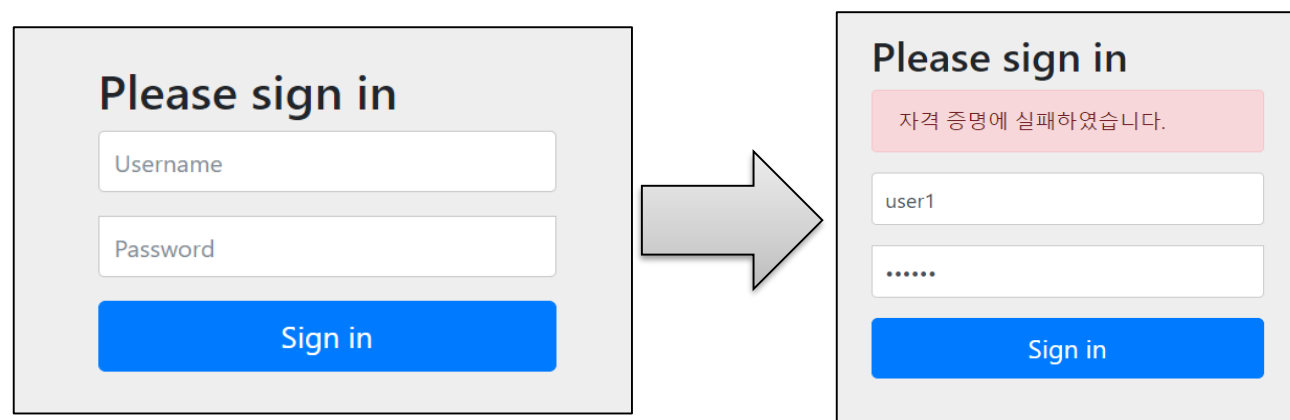


Member List Page	
아이디	이름
<a href="#">bb</a>	bb
<a href="#">aa</a>	aa
<a href="#">user1</a>	둘리
Name	Value
JSESSIONID	A76251736B8A56016B180A944116A9B0

## 로그인 테스트

1. 게시물 목록 화면에 접속한다.
2. 잘못된 계정을 이용하면 로그인페이지에 실패메세지가 표시된다.
3. 정상적인 계정을 이용하면 목록 화면으로 이동한다.

## 비정상적인 로그인

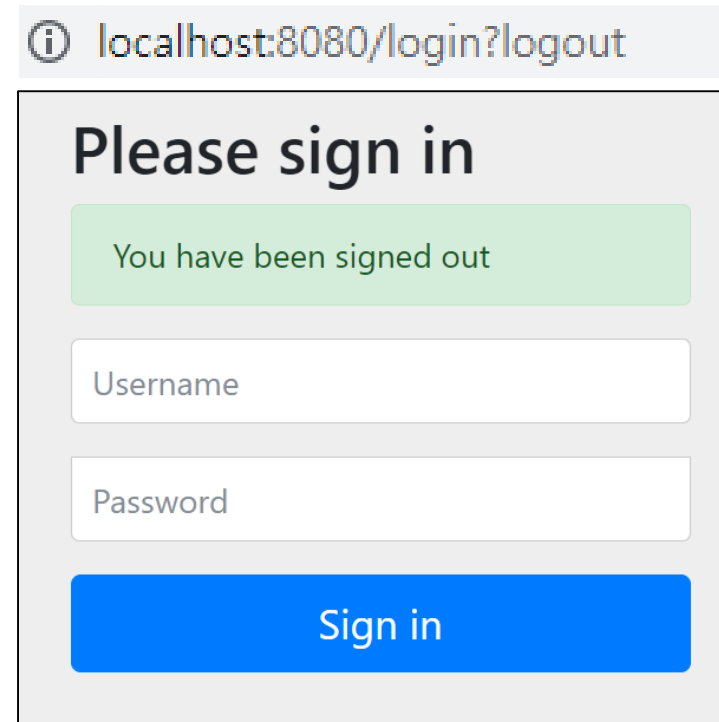
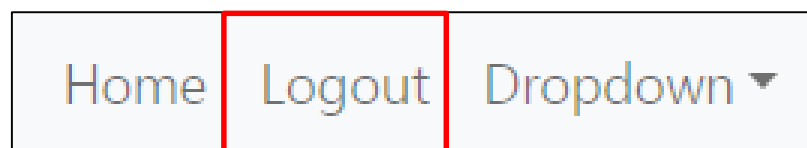


## 정상적인 로그인



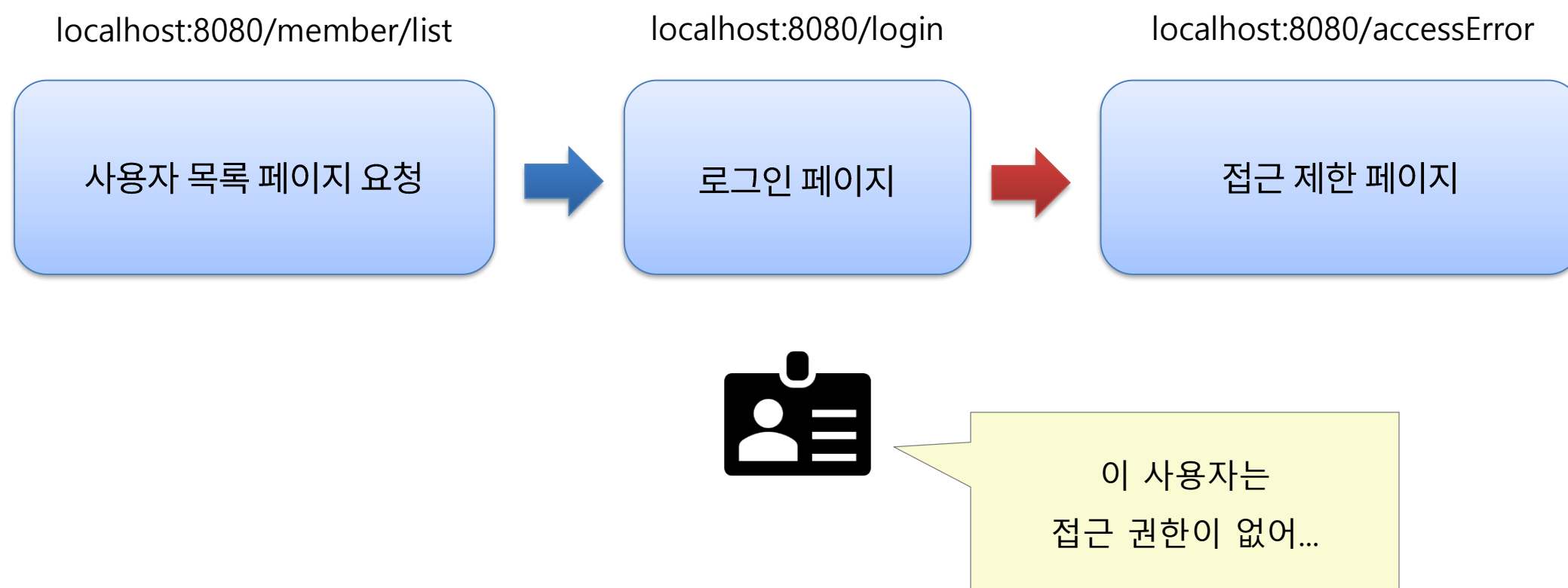
### 로그아웃 테스트

1. 메인페이지에 접속한다.
2. [Logout] 버튼을 클릭한다.
3. 로그인페이지로 이동하는지 확인한다.



### 접근제한 처리 과정

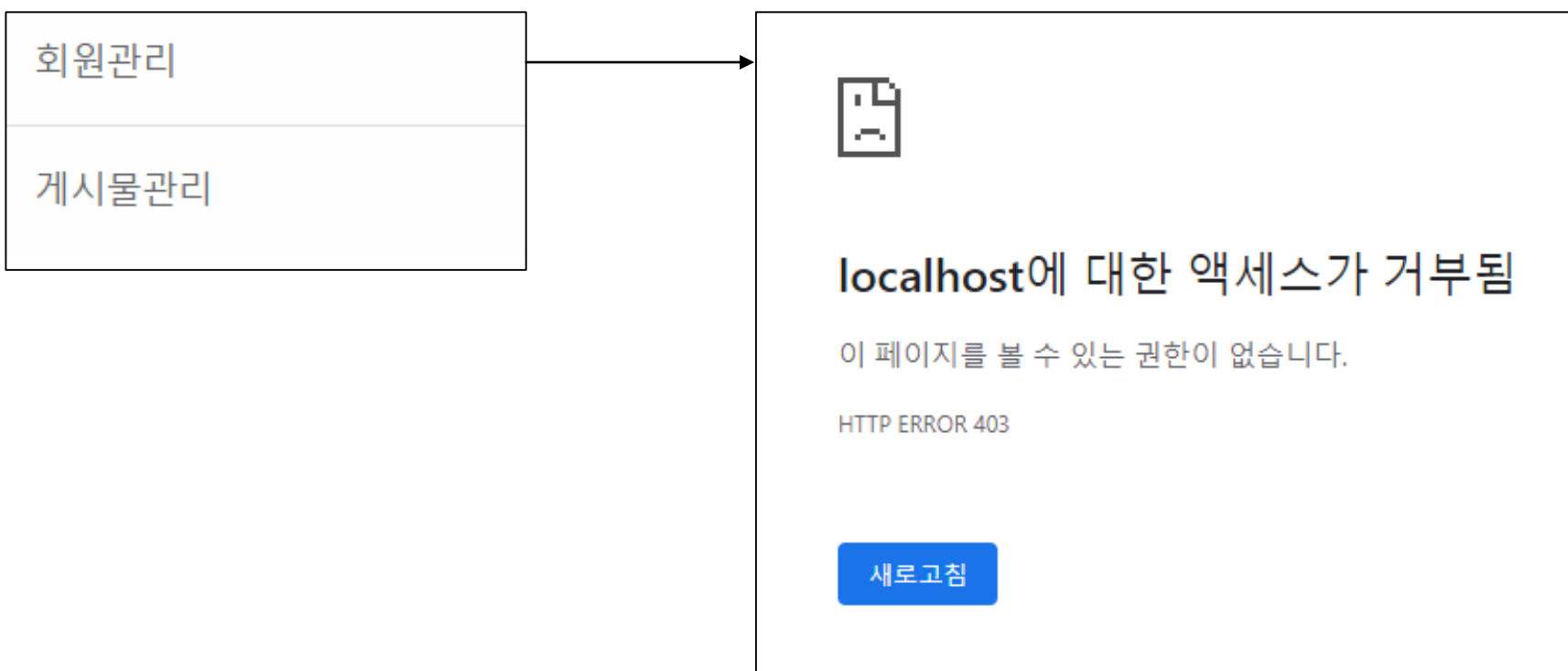
- 사용자에게 권한이 있다면, 원하는 페이지로 이동한다.
- 사용자에게 권한이 없으면, 접근 제한 페이지로 이동한다.





### 테스트

1. 일반사용자로 로그인한다.
2. [회원관리] 메뉴를 클릭한다.
3. 접근제한 페이지로 이동하는지 확인한다.



### 테스트

1. 관리자 계정으로 로그인한다.
2. 회원 상세 화면으로 이동한다.
3. 사용자의 권한이 나타나는지 확인한다.

Member Read Page	
아이디	user1
패스워드	.....
이름	둘리
등급	ROLE_USER

### 설정 클래스에서 로그인 처리

filterChain() 메소드를 수정하여 커스텀 로그인 페이지를 설정한다.

로그인 페이지 주소와 로그인 후 이동할 주소를 설정한다.

#### SecurityConfig

```
public SecurityFilterChain filterChain(HttpSecurity http) {  
    http.formLogin(  
        form ->  
            form.loginPage("/customlogin")  
                .loginProcessingUrl("/login")  
                .permitAll()  
                .successHandler((request, response, authentication) -> {  
                    response.sendRedirect("/");  
                })  
    );  
}
```

### 홈 컨트롤러에서 로그인 처리

컨트롤러에 커스텀 로그인 페이지를 반환하는 메소드를 추가한다.

#### HomeController

```
@GetMapping("/customlogin")  
public String customLogin() {  
    return "/home/login";  
}
```

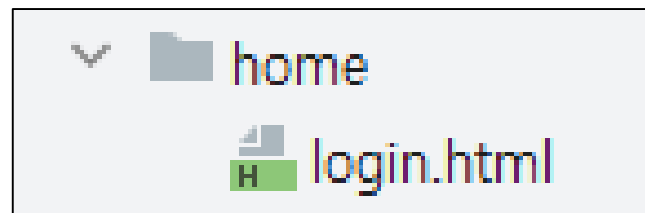
### 정적 페이지 활용하기

해당 프로젝트는 미리 준비된 정적 페이지를 사용합니다.

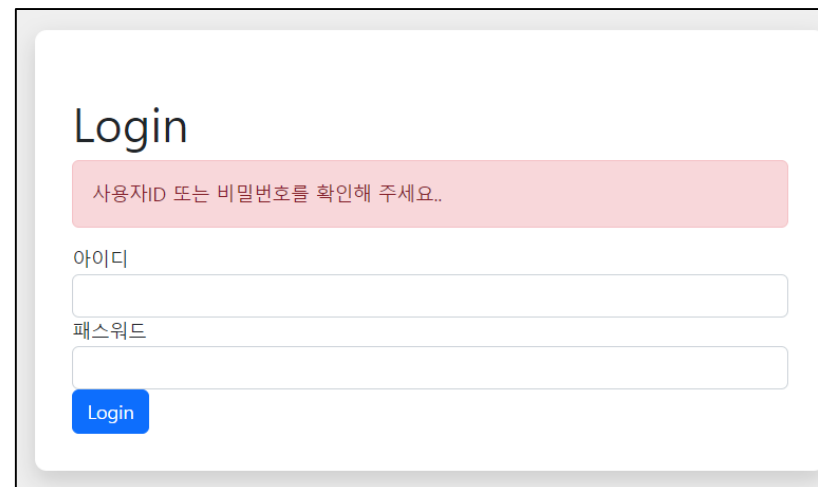
spring study의 Project10에 있는 HTML 파일을 복사하여 본인의 프로젝트에 저장하세요.

(/resources/static/static\_html/아래파일 → 복사 → /resource/templates/home/)

템플릿 폴더



결과 화면



로그인 폼에서 로그인 처리 URL을 설정한다.

action 속성에 로그인 처리 URL을 설정하면, 로그인 데이터가 Spring Security로 전송된다.

입력필드의 name 속성을 Spring Security에서 식별할 수 있는 이름으로 지정해야 한다.

로그인을 실패하면 URL 주소에 error 파라미터가 추가된다.

조건문을 사용하여 error 파라미터가 전달된 경우, 에러메세지를 출력한다.

#### login.html

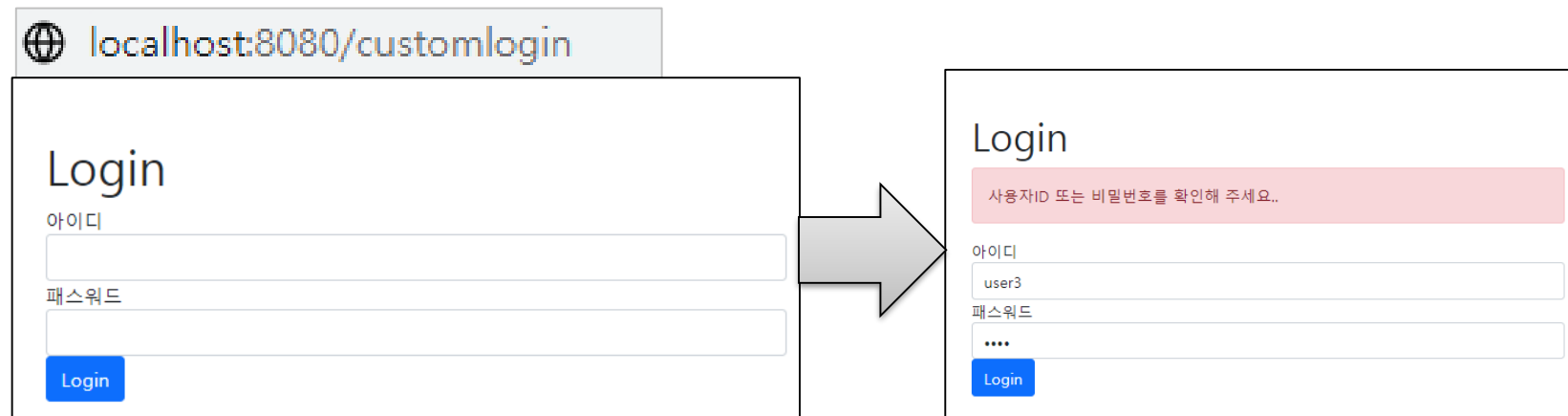
```
<form action="/login" method="post">
<div th:if="${param.error != null}">
    사용자ID 또는 비밀번호를 확인해 주세요..
</div>
<label >아이디</label><input type="text" name="username">
<label >패스워드</label><input type="password" name="password">
```

 localhost:8080/customlogin?error

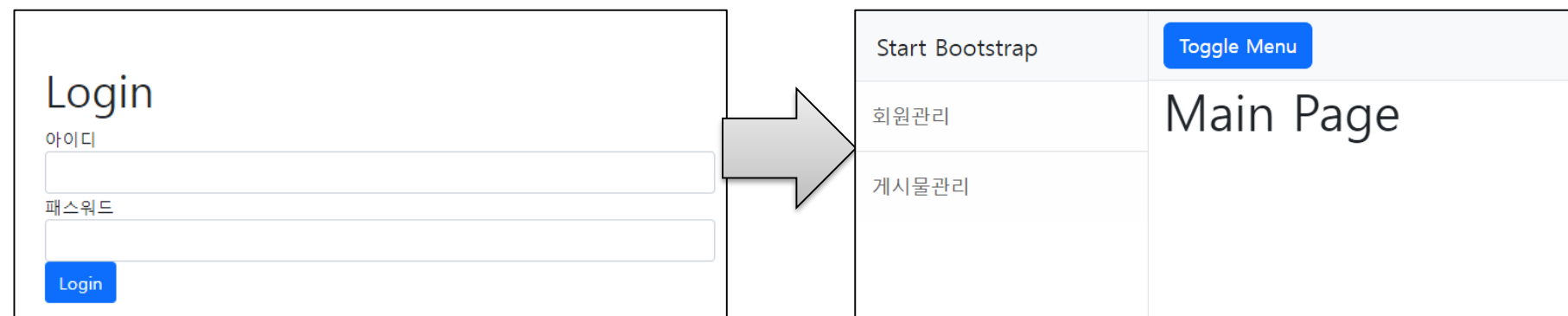
### 테스트

1. localhost:8080 주소로 접속한다.
2. localhost:8080/customlogin 으로 이동하는지 확인한다.
3. 커스텀 로그인 페이지가 나타나는지 확인한다.
4. 존재하지 않는 아이디 또는 잘못된 비밀번호로 로그인하면 에러메세지가 표시되는지 확인한다.

### 비정상적인 로그인



### 정상적인 로그인



메인 화면에 로그인한 사용자의 정보를 출력한다.

sec태그를 이용하면 HTML에서 Spring Security 기능을 사용할 수 있다.

authentication은 현재 로그인된 사용자의 정보를 포함하는 객체로, 사용자의 아이디와 등급을 가지고 있다.

#### main.html

```
<!--인증 객체 정보-->
<b>Authenticated DTO:</b>
<div sec:authentication="principal"></div>

<b>Authenticated username:</b>
<div sec:authentication="name"></div>

<b>Authenticated user role:</b>
<div sec:authentication="principal.authorities"></div>
```

PPT에서 코드를 복사하여  
프로젝트에 붙여 넣으세요.



### 테스트

1. localhost:8080 에 접속한다.
2. 메인 화면에 사용자 정보가 출력되는지 확인한다.

**Authenticated DTO:**

com.example.demo.dto.CustomUser [Username=cc, Password=[PROTECTED], Enabled=true, AccountNonExpired=true, [ROLE\_USER]]

**Authenticated username:**

cc

**Authenticated user role:**

[ROLE\_USER]

메뉴바에서 로그인된 사용자의 권한에 따라 메뉴를 표시하거나 숨긴다.

sec:authorize 속성은 인증과 관련된 정보를 가져오거나 조작하는 기능이다.

hasRole메소드를 사용하여 사용자가 관리자 권한을 가지고 있으면, '회원관리' 메뉴가 나타나도록 설정한다.

hasAnyRole메소드를 사용하여 관리자 또는 사용자 권한을 가지고 있으며, '게시물관리' 메뉴가 나타나도록 설정한다.

basic.html

```
<a sec:authorize="hasRole('ROLE_ADMIN')" th:href="@{/member/list}">회원관리</a>
```

```
<a sec:authorize="hasAnyRole('ROLE_ADMIN','ROLE_USER')" th:href="@{/board/list}">게시물관리</a>
```

### 테스트

1. 관리자로 로그인하여 사이드바에 회원관리와 게시물관리 메뉴가 보이는지 확인한다.
2. 일반 사용자로 로그인하여 사이드바에 게시물관리 메뉴가 보이는지 확인한다.

Start Bootstrap	Toggle Menu
회원관리	<b>Authenticated DTO:</b> com.example.demo.dto.CustomUser [Username: cc, Password: 1234, Role: ROLE_ADMIN] <b>Authenticated username:</b> zz <b>Authenticated user role:</b> [ROLE_ADMIN]
게시물관리	

관리자로 로그인

Start Bootstrap	Toggle Menu
게시물관리	<b>Authenticated DTO:</b> com.example.demo.dto.CustomUser [Username: cc, Password: 1234, Role: ROLE_USER] <b>Authenticated username:</b> cc <b>Authenticated user role:</b> [ROLE_USER]

일반사용자로 로그인

### 게시물 컨트롤러에서 작성자 처리

게시물 등록 시, 현재 로그인한 사용자의 아이디가 작성자로 입력되도록 메소드를 수정한다.

Principal 객체에서 로그인한 사용자의 아이디를 꺼내서, 게시물의 작성자 필드에 저장한다.

Principal: 로그인한 사용자 정보를 가지고 있는 객체

#### BoardController

```
@PostMapping("/register")
public String registerPost(BoardDTO dto, RedirectAttributes redirectAttributes, Principal principal) {
    String id = principal.getName();
    dto.setWriter(id);
    int no = service.register(dto);
    ...
}
```

### 게시물 등록 화면에서 작성자 처리

이제 게시물 등록할 때 사용자가 작성자를 입력할 필요가 없으므로, 회원가입 페이지에서 작성자 입력 필드를 삭제한다.

register.html

```
<div class="form-group">
  <label >작성자</label>
  <input type="text" class="form-control" name="writer">
</div>
```

삭제

### 게시물 등록 테스트

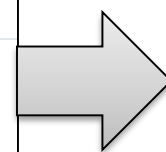
1. 게시물 등록 화면에 접속한다.
2. 새로운 게시물을 등록한다.
3. 작성자가 자동으로 저장됐는지 확인한다.

Board Register Page

제목

내용

등록



Board List Page			게시물 등록
#	제목	작성자	
<a href="#">5</a>	안녕하세요~	user1	

### 댓글 컨트롤러에서 작성자 처리

이전에는 댓글을 등록할 때, 임시 아이디를 사용했다.

이제 로그인한 사용자의 아이디를 사용하도록 댓글 등록 메소드를 수정한다.

인증객체에서 로그인한 사용자의 아이디를 꺼내서, 작성자 필드에 저장한다.

#### CommentController

```
@PostMapping("/register")
public HashMap<String, Boolean> register(CommentDTO dto, Principal principal) {
    String id = principal.getName();
    dto.setWriter(id);
    service.register(dto);
}
```

# 사용자 정보 출력

## 댓글 등록에서 작성자 처리

### 댓글 등록 테스트

1. 게시물 상세 화면에 접속한다.
2. 새로운 댓글을 등록한다.
3. 작성자가 자동으로 저장됐는지 확인한다.

