

# 7장. 스프링부트 시큐리티

*authentication*



Spring Boot

## ■ 인증과 인가

인증은 통해 사용자를 식별하고, 인가를 통해 시스템 자원에 대한 접근을 통제한다.

### 인증(Authentication)

어떤 직원이 회사 건물에 들어가기 위해서는 사원증이나 RFID 카드를 이용해서 반드시 인증에 통과해야 한다. 인증에 실패한 사람이 회사 건물에 들어가려고 하면 당연히 보안 직원이 재제할 것이다.

### 인가(Authorization)

그리고 직급과 직무에 따라 부여된 권한이 다르기 때문에 회사 내에서 열람할 수 있는 문서의 종류도 제한되어 있다. 이렇게 직원이 특정 자원에 접근할 때 적절한 권한이 있는지 확인하는 과정을 인가라고 할 수 있다.



# 스프링부트 시큐리티

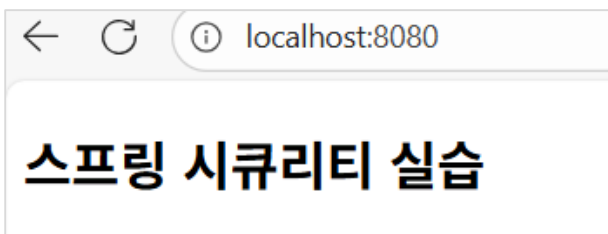
## ■ 프로젝트 계층 구조

```
✓ > Spring-Security [boot] [devtools] [javaweb main]
  ✓ > src/main/java
    ✓ > com.springboot
      ✓ > config
        > SecurityConfig.java
      ✓ > controller
        > AuthController.java
        > HomeController.java
        > SpringSecurityApplication.java
    ✓ > src/main/resources
      static
    ✓ > templates
      ✓ > authentication
        adminPage.html
        homePage.html
        userPage.html
        viewPage.html
      ✓ > member
        login.html
        home.html
      application.properties
```



# 스프링부트 시큐리티

- Home page 생성



```
@Controller
public class HomeController {

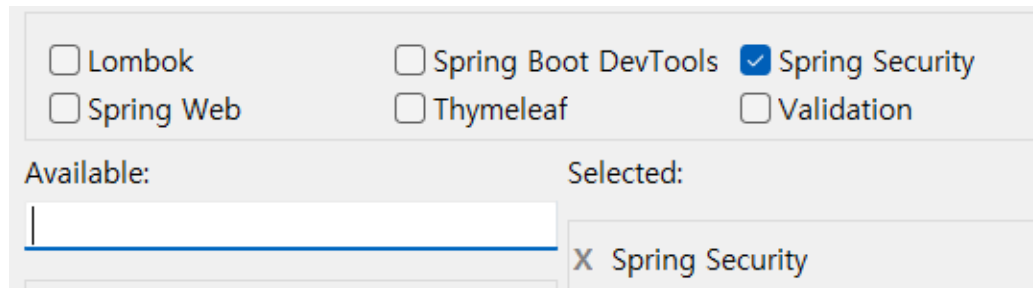
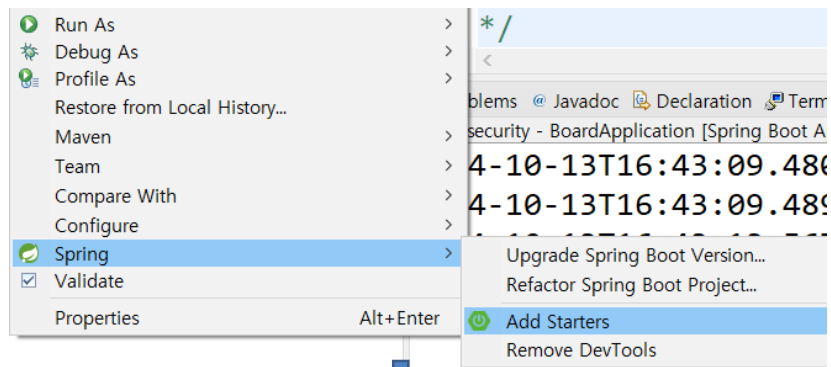
    @GetMapping
    public String home() {
        return "home";
    }
}
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome~</title>
</head>
<body>
    <h2>스프링 시큐리티 실습</h2>
</body>
</html>
```



# 스프링부트 시큐리티

- spring security 설정
  - 시큐리티 스타터 추가



# 스프링부트 시큐리티

- spring security 설정
  - pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```



## ▪ spring security 실행

비밀번호 자동 생성

```
Using generated security password: d93b057b-024f-47a1-9de5-cf318edb2ab4
```

```
This generated password is for development use only. Your security config
```

```
2025-10-28T19:19:05.452+09:00 INFO 16300 --- [Spring-Security] [ restart
2025-10-28T19:19:05.545+09:00 INFO 16300 --- [Spring-Security] [ restart
2025-10-28T19:19:05.575+09:00 INFO 16300 --- [Spring-Security] [ restart
2025-10-28T19:19:05.584+09:00 INFO 16300 --- [Spring-Security] [ restart
```

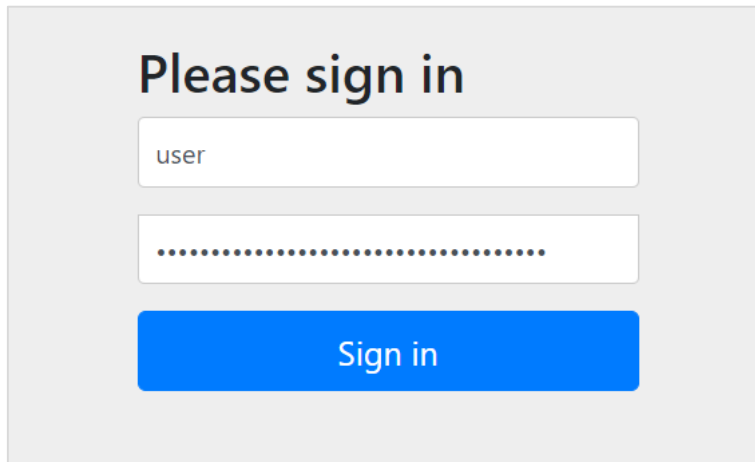


# 스프링부트 시큐리티

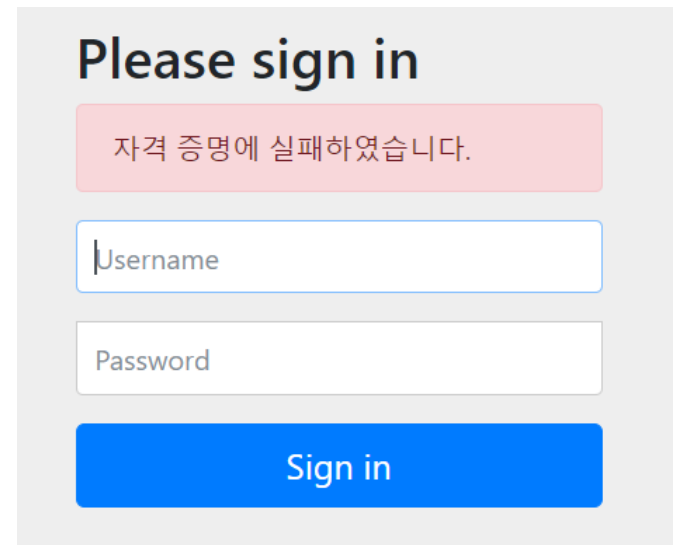
- 제공된 로그인 폼

메인 앱을 실행하고 브라우저를 새로 열어서 hello.html 요청하면 스프링 시큐리티가 제공하는 기본 로그인 화면이 제공됨

**Username에 'user', Password에 콘솔에서 복사한 암호를 입력함**



A screenshot of the default Spring Security login form. It has a light gray background. At the top, it says "Please sign in". Below that are two input fields: the first contains the text "user", and the second is filled with dots to represent a password. At the bottom is a blue button with the text "Sign in".



A screenshot of the Spring Security login form after an unsuccessful login attempt. It has a light gray background. At the top, it says "Please sign in". Below that is a pink error message box containing the text "자격 증명에 실패하였습니다." (Authentication failed). Underneath the error box are two input fields: the first is labeled "Username" and the second is labeled "Password". At the bottom is a blue button with the text "Sign in".





- 참조 메뉴얼

Spring.io > Learn > Guides > sec로 검색 > Securing a Web Application

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests((requests) -> requests
                .requestMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin((form) -> form
                .loginPage("/login")
                .permitAll()
            )
            .logout((logout) -> logout.permitAll());

        return http.build();
    }
}
```



# 스프링부트 시큐리티

- 시큐리티 설정파일

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig{

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        return http.build();
    }
}
```

**SecurityFilterChain** 클래스가 빈으로 등록되면 애플리케이션에서는 더 이상 로그인을 요구하지 않는다.



# 스프링부트 시큐리티

## ▪ 시큐리티 전반 동작 개요

구 분	기 능
SecurityConfig	스프링 시큐리티 전반 설정 (formLogin, logout, 권한별 접근제한)
CustomUserDetails	DB 회원정보(Member)를 UserDetails 객체로 감싸 시큐리티에서 인증 처리 가능하게 함
Thymeleaf 보안	sec:authorize, \${#authentication.principal.name}을 통해 로그인 상태 표시



## ■ HttpSecurity가 제공하는 주요 메소드

메서드	사용가능한 메서드	의미
authorizeHttpRequests()		사용자 인증과 권한을 설정
	requestMatchers("url패턴")	매칭되는 url 패턴들에 대한 접근 허용 permitAll()은 모든 사용자에게 접근 허용 hasRole("권한")은 특정 권한을 가진 사용자만 접근 허용
formLogin()		로그인 페이지 설정
	loginPage("/login")	로그인이 필요한 url로 접근하면 '/login' 화면으로 이동
	loginProcessUrl("/perform_login")	/perform_login을 action 경로로 로그인 처리됨
logout()		로그아웃 페이지 설정
	logoutUrl("/logout")	로그아웃을 처리하는 페이지 설정



# 스프링부트 시큐리티

- **sec:authorize – 보안 조건부 렌더링**

```
<th:block sec:authorize="조건">  
    <!-- 조건이 true일 때만 보이는 내용 -->  
</th:block>
```

조건	의미
isAuthenticated()	로그인한 사용자일 경우
isAnonymous()	로그인하지 않은(익명) 사용자
hasRole('ADMIN')	ADMIN 권한이 있을 때
hasAnyRole('USER','ADMIN')	여러 권한 중 하나라도 있으면



# 스프링부트 시큐리티

## ■ 시큐리티 설정

```
@EnableWebSecurity
@Configuration
public class SecurityConfig {

    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http
            .authorizeHttpRequests(auth -> auth
                //해당 경로만 허용
                .requestMatchers("/", "/auth", "/home/**").permitAll()
                .anyRequest().authenticated() //나머지는 인증 필요
            )
            .formLogin(form -> form
                .loginPage("/login") //사용자 로그인 페이지
                .permitAll() //누구나 접근 가능
            )
    }
}
```



# 스프링부트 시큐리티

## ■ 시큐리티 설정

```
.logout(logout -> logout //로그 아웃
    .logoutRequestMatcher(new AntPathRequestMatcher("/logout", "GET"))
    .logoutSuccessUrl("/auth")
    .invalidateHttpSession(true)
    .deleteCookies("JSESSIONID")
);

return http.build();
}
```



# 스프링부트 시큐리티

## ■ 시큐리티 설정

구성 요소	역할
@Configuration	설정 클래스로 등록 (스프링이 인식)
@EnableWebSecurity	Spring Security 웹 보안 기능 활성화
@RequiredArgsConstructor	final 필드(userDetailsService) 자동 주입
DaoAuthenticationProvider	DB 기반 사용자 인증 처리기 등록
SecurityFilterChain	HTTP 요청별 보안 규칙 정의 (핵심 설정 부분)
PasswordEncoder	비밀번호 암호화 방식 지정 (BCrypt)





## ▪ AuthController.java

```
@Controller
public class AuthController {

    @GetMapping("/auth")
    public String method1() {
        return "authentication/viewPage";
    }

    @GetMapping("/home/main")
    public String method2(Model model) {
        model.addAttribute("data", "homePage.html");
        return "authentication/homePage";
    }

    @GetMapping("/user/main")
    public String method3(Model model) {
        model.addAttribute("data", "userPage.html");
        return "authentication/userPage";
    }
}
```



- AuthController.java

```
@GetMapping("/admin/main")
public String method4(Model model) {
    model.addAttribute("data", "adminPage.html");
    return "authentication/adminPage";
}

@GetMapping("/login")
public String login() {
    return "member/login";
}
}
```



# 스프링부트 시큐리티

## ▪ viewPage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>스프링 시큐리티</title>
</head>
<body>
    <h3>스프링 시큐리티</h3>
    <ul>
        <li><a th:href="@{/home/main}">/home/main</a> (누구나 접근 가능)</li>
        <li><a th:href="@{/member/main}">/member/main</a> (USER 전용)</li>
        <li><a th:href="@{/admin/main}">/admin/main</a> (ADMIN 전용)</li>
    </ul>
</body>
</html>
```



## ▪ homePage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>homePage</title>
</head>
<body>
    <h3>홈 페이지</h3>
    <p>뷰 페이지는 <span th:text="${data}"></span> 입니다.</p>
    <p>
        <a th:href="@{/auth}">[메인으로]</a>
    </p>
</body>
</html>
```



# 스프링부트 시큐리티

- **userPage.html**

```
<h2>회원(USER) 전용 페이지</h2>
<p>뷰 페이지는 <span th:text="${data}">입니다.</span>

<p>
  <a th:href="@{/auth}">[메인으로]</a>
  <a th:href="@{/logout}">[로그아웃]</a>
</p>
```

- **adminPage.html**

```
<h2>관리자(ADMIN) 전용 페이지</h2>
<p>뷰 페이지는 <span th:text="${data}">입니다.</span>

<p>
  <a th:href="@{/auth}">[메인으로]</a>
  <a th:href="@{/logout}">[로그아웃]</a>
</p>
```



# 스프링부트 시큐리티

## ▪ login.html

name 속성(필수) -> 아이디: username, 비밀번호: password

```
<h3>로그인</h3>
<form th:action="@{/login}" method="post">
  <p>
    <label>아이디: <input type="text" name="username"></label>
  </p>
  <p>
    <label>비밀번호: <input type="password" name="password"></label>
  </p>
  <p>
    <button type="submit">로그인</button>
  </p>
</form>
<p><a th:href="@{/auth}">[홈으로]</a></p>
```



# 스프링부트 시큐리티

## ▪ 사용자 계정 설정

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

//사용자 계정 설정 - In memory
@Bean
UserDetailsService userDetailsService() {
    UserDetails user = User.builder()
        .username("guest")
        .password(passwordEncoder().encode("g1234"))
        .roles("USER")
        .build();

    UserDetails admin = User.builder()
        .username("admin")
        .password(passwordEncoder().encode("a1234"))
        .roles("ADMIN")
        .build();

    return new InMemoryUserDetailsManager(user, admin);
}
```



# 스프링부트 시큐리티

- 사용자 계정 설정

<< guest로 로그인 >>

**로그인**

아이디:

비밀번호:

[\[홈으로\]](#)



## 회원(USER) 전용 페이지

뷰 페이지는 userPage.html

[\[메인으로\]](#) [\[로그아웃\]](#)





# 스프링부트 시큐리티

- 사용자 계정 설정

<< admin으로 로그인 >>

**로그인**

아이디:

비밀번호:

[\[홈으로\]](#)



## 관리자(ADMIN) 전용 페이지

뷰 페이지는 adminPage.html

[\[메인으로\]](#) [\[로그아웃\]](#)



# DB 회원 정보

## ■ 프로젝트 계층 구조

```

└─ > Members [boot] [devtools] [javaweb main]
    └─ > src/main/java
        └─ > com.springboot
            ├── > config
            ├── > controller
            ├── > dto
            ├── > entity
            ├── > repository
            ├── > service
            └─ > MembersApplication.java
        └─ > src/main/resources
            ├── > static
            │   ├── > css
            │   └─ > images
            └─ > templates
                ├── > board
                ├── > error
                └─ > member
                    └─ > index.html
            └─ > application.properties

```



## Good Releation - 좋은 사이

---



[로그인](#) | [회원 가입](#) | [회원 목록](#) |

# DB 기반 사용자 인증

## ▪ index.html

```
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">

  <section id="container">
    <div id="main">
      <h2>Good Releation - 좋은 사이</h2>
      <div class="main-pic">
        
      </div>
      <div class="menu">
        <th:block sec:authorize="isAnonymous()">
          <a th:href="@{/members/login}" th:text='로그인'></a> |
        </th:block>
        <th:block sec:authorize="isAuthenticated()">
          [<span sec:authentication="principal.name"></span>]님
          <a th:href="@{/logout}" th:text='로그아웃'></a> |
        </th:block>
        <a th:href="@{/members/join}">회원 가입</a> |
        <a th:href="@{/members}">회원 목록</a> |
      </div>
    </div>
  </section>
```



# DB 기반 사용자 인증

## ▪ style.css

```
/* 공통 스타일 */
#container{
    width: 80%;
    margin: 0 auto;
}
h2{
    border-bottom: 2px solid #ccc;
    padding: 16px;
    text-align: center;
}
a{text-decoration: none;}
ul li{
    list-style: none;
    margin: 16px;
}
table thead{background-color: #eee;}
.msg{color: #00f; margin: 10px;}
.error{color: #f00; margin: 10px;}
```

```
/* index 스타일 */
#main{text-align: center;}
.main-pic img{
    width: 40%;
    border-radius: 8px;
}
.menu{margin: 10px;}

/* join 스타일 */
.joinForm, .LoginForm{
    width: 400px;
    margin: 0 auto;
}
.joinForm label, .LoginForm label{
    width: 80px;
    float: left;
}
```



# DB 기반 사용자 인증

## ▪ style.css

```
/* memberList 스타일 */
.memberList, .boardList{
    width: 600px;
    margin: 20px auto;
    border: 1px solid #ccc;
    border-collapse: collapse;
}

.memberList th, td, .boardList th, td{
    border: 1px solid #ccc;
    padding: 5px 10px;
}

/* memberInfo 스타일 */
.memberInfo{
    width: 400px;
    margin: 0 auto;
}

.memberInfo label{
    width: 80px;
    float: left;
}

.btnList{margin: 16px;}
.btnList, button{padding: 5px 10px;}
```



# DB 기반 사용자 인증

- application.property

```
# DataSource 설정 - mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/bootdb?serverTime=Asia/Seoul
spring.datasource.username=bootuser
spring.datasource.hikari.password=pwboot

# JPA 설정
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```



# DB 기반 사용자 인증

- 시큐리티 설정파일

```
@EnableWebSecurity
@Configuration
public class SecurityConfig {

    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/", "/css/**", "/images/**",
                    "/members/join", "/members/login").permitAll()
                //USER, ADMIN 권한 모두 허용
                .requestMatchers("/members/**").hasAnyRole("USER", "ADMIN")
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated() //나머지는 인증 필요
            )
    }
}
```





# DB 기반 사용자 인증

## ■ 시큐리티 설정파일

```
.formLogin(form -> form
    .loginPage("/members/login") //사용자 로그인 페이지
    .permitAll()                //누구나 접근 가능
)
.logout(logout -> logout //로그 아웃
    .logoutRequestMatcher(new AntPathRequestMatcher("/logout", "GET"))
    .logoutSuccessUrl("/")
    .invalidateHttpSession(true)
    .deleteCookies("JSESSIONID")
);

return http.build();
}
```

```
//비밀번호 암호화
@Bean
PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```



# DB 기반 사용자 인증

## ▪ Member 엔티티

```
@Data
@Entity
public class Member {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    @Column(unique=true)
    private String email;

    @Column(nullable=false)
    private String passwd;

    @Column(length=30, nullable=false)
    private String name;

    @Column(length=10)
    private String gender;
```



# DB 기반 사용자 인증

## ▪ Member 엔티티

```
@Column(nullable=false)
private String role;

@CreationTimestamp
private Timestamp joinDate;

//DTO를 Entity로 변환
public static Member toSaveEntity(MemberDTO dto,
    PasswordEncoder pwEncoder) {
    Member member = new Member();
    member.setEmail(dto.getEmail());
    member.setPasswd(pwEncoder.encode(dto.getPasswd()));
    member.setName(dto.getName());
    member.setGender(dto.getGender());
    member.setRole("ROLE_USER");
    return member;
}
```

관리자는  
ROLE\_ADMIN으로 변경



# DB 기반 사용자 인증

## ▪ Member Repository

```
public interface MemberRepository extends JpaRepository<Member, Long>{  
  
    //이메일을 찾아서 Member를 반환하는 메서드  
    Optional<Member> findByEmail(String email);  
}
```



# DB 기반 사용자 인증

## ▪ Member Service

```
@RequiredArgsConstructor
@Service
public class MemberService {

    private final MemberRepository repository;
    private final PasswordEncoder pwEncoder;

    //회원 추가
    public void save(MemberDTO dto) {
        if (repository.findByEmail(dto.getEmail()).isPresent()) {
            throw new IllegalArgumentException("이미 존재하는 이메일입니다.");
        }
        //DTO를 Entity로 변환 메서드 호출
        Member member = Member.toSaveEntity(dto, pwEncoder);
        repository.save(member);
    }
}
```



# DB 기반 사용자 인증

## ▪ Member Service

```
//회원 목록
public List<Member> findAll() {
    return repository.findAll();
}

//회원 정보
public Member findById(Long id) {
    return repository.findById(id)
        .orElseThrow(() ->
            new IllegalArgumentException("회원이 존재하지 않습니다."));
}
```



# DB 기반 사용자 인증

## ▪ Member Controller

```
@RequiredArgsConstructor
@RequestMapping("/members")
@Controller
public class MemberController {

    private final MemberService service;

    @GetMapping("/join") //회원 가입 페이지
    public String joinForm() {
        return "member/join";
    }

    @PostMapping("/join") //회원 가입 처리
    public String join(@ModelAttribute MemberDTO dto,
        RedirectAttributes ra) {
        try {
            service.save(dto);
            return "redirect:/members/login";
        } catch (Exception e) {
            ra.addFlashAttribute("error", e.getMessage());
            return "redirect:/members/join";
        }
    }
}
```



# DB 기반 사용자 인증

## 회원 가입

이메일

비밀번호

이름

성별 ☒ 남 ☐ 여

가입

취소





# DB 기반 사용자 인증

## ▪ join.html

```
<section id="container">
  <h2>회원 가입</h2>
  <form th:action="@{/members/join}" method="post"
        class="joinForm">
    <div th:if="${error}" class="error">
      [[${error}]]
    </div>
    <fieldset>
      <ul>
        <li>
          <label for="email">이메일</label>
          <input type="text" name="email" required>
        </li>
        <li>
          <label for="mid">비밀번호</label>
          <input type="password" name="passwd">
        </li>
      </ul>
    </fieldset>
  </form>
</section>
```



# DB 기반 사용자 인증

## ▪ join.html

```
<li>
  <label for="mid">이름</label>
  <input type="text" name="name">
</li>
<li>
  <label for="mid">성별</label>
  <input type="radio" name="gender" value="남자" checked>남
  <input type="radio" name="gender" value="여자">여
</li>
<li>
  <input type="submit" value="가입">
  <input type="reset" value="취소">
</li>
</ul>
</fieldset>
</form>
```



# DB 기반 사용자 인증

## ● 사용자 정의 UserDetailsService 구현하기

인증 관리자는 UserDetailsService 객체를 통해 UserDetails 객체를 획득하고, UserDetails 객체에서 인증과 인가에 필요한 정보들을 추출하여 사용한다.

스프링부트는 UserDetailsService를 구현한 클래스를 기본으로 제공한다. 그리고 이클립스가 제공하는 UserDetails 객체는 아이디가 'user' 이고 비밀번호는 암호화되어 콘솔에 출력되는 긴 문자열이다.

스프링부트가 제공하는 UserDetailsService를 커스터 마이징하고 싶으면 UserDetailsService 인터페이스를 구현한 클래스를 직접 작성하여 등록하면 된다.



# DB 기반 사용자 인증

## ▪ CustomUserDetails

```
//UserDetails을 구현한 CustomUserDetails
public class CustomUserDetails implements UserDetails{

    private static final long serialVersionUID = 1L;

    private Member member;

    public CustomUserDetails(Member member) {
        this.member = member;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        //권한 반환
        return Collections.singletonList(
            new SimpleGrantedAuthority(member.getRole())
        );
    }
}
```



# DB 기반 사용자 인증

## ▪ CustomUserDetails

```
@Override
public String getPassword() {
    return member.getPassword();
}

@Override
public String getUsername() {
    return member.getEmail();
}

//Member 접근용 getter
public Long getId() {
    return member.getId();
}

public String getName() {
    return member.getName();
}
}
```



# DB 기반 사용자 인증

## ▪ Service - CustomUserDetailsService

```
@RequiredArgsConstructor
@Service
public class CustomUserDetailsService implements UserDetailsService{

    private final MemberRepository repository;

    @Override
    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException {
        Member member = repository.findByEmail(username)
            .orElseThrow(() ->
                new UsernameNotFoundException("사용자를 찾을 수 없습니다."));
        return new CustomUserDetails(member);
    }
}
```



# DB 기반 사용자 인증

## 회원 목록

번호	이메일	이름	성별	권한	가입일
1	<a href="mailto:today@space.com">today@space.com</a>	김기용	남자	ROLE_USER	2025-10-29 19:08
2	<a href="mailto:coli@robot.com">coli@robot.com</a>	박미량	여자	ROLE_USER	2025-10-29 19:16
4	<a href="mailto:admin@korea.kr">admin@korea.kr</a>	관리자	여자	ROLE_ADMIN	2025-10-29 23:14



# DB 기반 사용자 인증

## ▪ list.html

```
<h2>회원 목록</h2>
<table class="memberList">
  <thead>
    <tr>
      <th>번호</th><th>이메일</th><th>이름</th><th>성별</th><th>권한</th><th>가입일</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="m: ${memberList}">
      <td th:text="${m.id}"></td>
      <td>
        <a th:href="@{/members/{id}(id=${m.id})}" th:text="${m.email}"></a>
      </td>
      <td th:text="${m.name}"></td>
      <td th:text="${m.gender}"></td>
      <td th:text="${m.role}"></td>
      <td th:text="${#dates.format(m.joinDate, 'yyyy-MM-dd HH:mm')}"></td>
    </tr>
  </tbody>
</table>
```





# DB 기반 사용자 인증

## ▪ login.html

```
<h2>로그인</h2>
<!-- 로그인 실패 시 에러 메시지 출력 -->
<form th:action="@{/Login}" method="post" class="loginForm">
    <div th:if="${param.error}" class="error">
        아이디 또는 비밀번호가 일치하지 않습니다.
    </div>
    <fieldset>
        <ul>
            <li>
                <label for="email">이메일</label>
                <input type="text" name="username">
            </li>
            <li>
                <label for="mid">비밀번호</label>
                <input type="password" name="password">
            </li>
            <li>
                <input type="submit" value="로그인">
            </li>
        </ul>
    </fieldset>
```



# DB 기반 사용자 인증

## ▪ `${#authentication}` — 인증 객체 접근

`#authentication`은 Spring Security가 제공하는 현재 인증 정보(Authentication 객체)를 참조할 수 있는 Thymeleaf 내장 객체입니다.

즉, Java 코드에서 `SecurityContextHolder.getContext().getAuthentication()`을 꺼내는 것과 동일하다.

속성	의미	예시
<code>\${#authentication.name}</code>	로그인한 사용자 이름 (username)	이메일이나 ID
<code>\${#authentication.principal}</code>	사용자 정보 (UserDetails 객체)	CustomUserDetails
<code>\${#authentication.authorities}</code>	사용자 권한 목록	[ROLE_USER, ROLE_ADMIN]



# DB 기반 사용자 인증

- DaoAuthenticationProvider (인증 공급자)

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

    private final CustomUserDetailsService userDetailsService;

    @Bean //sec:authorize를 작동함
    public DaoAuthenticationProvider
        authenticationProvider(PasswordEncoder encoder) {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(encoder);
        return authProvider;
    }
}
```



# DB 기반 사용자 인증

## ▪ DaoAuthenticationProvider (인증 공급자)

- 역할:  
Spring Security가 로그인 시 입력한 ID/PW를 DB 정보와 비교할 때 이 DaoAuthenticationProvider를 사용합니다.
- 내부적으로 UserDetailsService를 호출해 UserDetails 객체(CustomUserDetails)를 가져옵니다.
- 이후 PasswordEncoder로 비밀번호를 검증합니다.



# DB 기반 사용자 인증

## ▪ 로그인 처리 흐름

- /login 요청 발생
- DaoAuthenticationProvider가 userDetailsService.loadUserByUsername() 호출
- DB의 비밀번호와 입력된 비밀번호를 BCryptPasswordEncoder.matches()로 비교
- 성공 시 Authentication 객체 생성 후 세션 저장



# DB 기반 사용자 인증

## Good Releation - 좋은 사이

---



[김기용]님 로그아웃 | 회원 가입 | 회원 목록 |



# DB 기반 사용자 인증

## 회원 정보

이메일	<input type="text" value="today@space.com"/>
비밀번호	<input type="password" value="....."/>
이름	<input type="text" value="김기용"/>
성별	<input checked="" type="radio"/> 남 <input type="radio"/> 여

수정

탈퇴

목록



# DB 기반 사용자 인증

## ▪ info.html

```
<h2>회원 정보</h2>
<div class="memberInfo">
  <fieldset>
    <ul>
      <li>
        <label for="email">이메일</label>
        <input type="text" name="email"
          th:value="${member.email}" readonly>
      </li>
      <li>
        <label for="passwd">비밀번호</label>
        <input type="password" name="passwd"
          th:value="${member.passwd}" readonly>
      </li>
      <li>
        <label for="name">이름</label>
        <input type="text" name="name"
          th:value="${member.name}" readonly>
      </li>
    </ul>
  </fieldset>
</div>
```





# DB 기반 사용자 인증

## ▪ info.html

```
<li>
  <label for="gender">성별</label>
  <input type="radio" name="gender"
    value="남자" th:checked="${member.gender == '남자'}">남
  <input type="radio" name="gender"
    value="여자" th:checked="${member.gender == '여자'}">여
</li>
</ul>
</fieldset>
<div sec:authorize="isAuthenticated()" class="btnList">
  <th:block th:if="${#authentication.principal.name == member.name}">
    <a th:href="@{/members/edit/${member.id}|"><button>수정</button></a>
    <a th:href="@{/members/delete/${member.id}|"
      onclick="return confirm('정말 탈퇴하시겠습니까?')"><button>탈퇴</button></a>
  </th:block>
  <a th:href="@{/members}"><button>목록</button></a>
</div>
</div>
```



## Good Releation - 좋은 사이

---



[로그인](#) | [회원 가입](#) | [회원 목록](#) | [관리자](#)

## 관리자 대시보드

뷰 페이지는 adminPage.html

[\[메인으로\]](#) [\[로그아웃\]](#)



- AdminController

```
@Controller
public class AdminController {

    @GetMapping("/admin/dashboard")
    public String adminDashboard(Model model) {
        model.addAttribute("data", "adminPage.html");
        return "admin/adminPage";
    }
}
```



- adminPage.html

```
<h2>관리자(ADMIN) 전용 페이지</h2>
<p>뷰 페이지는 <span th:text="${data}">입니다.</span>

<p>
  <a th:href="@{/auth}">[메인으로]</a>
  <a th:href="@{/logout}">[로그아웃]</a>
</p>
```



- ROLE\_USER 권한으로 관리자 메뉴에 접근하면 권한 에러 페이지로 이동

## 403 접근 권한 에러

이 페이지에 접근할 권한이 없습니다.

[\[메인으로\]](#)



## ▪ SecurityConfig

```
        .logout(logout -> logout //로그 아웃
            .logoutRequestMatcher(new AntPathRequestMatcher(
            .logoutSuccessUrl("/")
            .invalidateHttpSession(true)
            .deleteCookies("JSESSIONID")
        )
        .exceptionHandling(ex -> ex
            .accessDeniedPage("/access-denied"));
    return http.build();
}
```



- AccessErrorController

```
@Controller
public class AccessErrorController {

    @GetMapping("/access-denied")
    public String accessDenied() {
        return "error/accessDenied";
    }
}
```





- accessDenied.html

```
<section id="container">
  <h2>403 접근 권한 에러</h2>
  <h3>이 페이지에 접근할 권한이 없습니다.</h3>

  <p>
    <a th:href="@{/}">[메인으로]</a>
  </p>
</section>
```

