

7장. 스프링부트 시큐리티



security



Spring Boot

▪ 인증과 인가

인증(Authentication)과 인가(Authorization)

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


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

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.



▪ BootSecurity 프로젝트 생성

New Spring Starter Project 

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

▪ 모듈 추가(시큐리티 스타터 제외)

Spring Boot Version:

Frequently Used:

- | | | |
|--|---|--|
| <input type="checkbox"/> H2 Database | <input checked="" type="checkbox"/> Lombok | <input checked="" type="checkbox"/> MariaDB Driver |
| <input checked="" type="checkbox"/> Spring Boot DevTools | <input checked="" type="checkbox"/> Spring Data JPA | <input type="checkbox"/> Spring Security |
| <input checked="" type="checkbox"/> Spring Web | <input type="checkbox"/> Spring Web Services | <input checked="" type="checkbox"/> Thymeleaf |



- application.properties

```
# WebApplication Type Setting
# spring.main.web-application-type=none
spring.main.web-application-type=servlet

# DataSource Setting
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://127.0.0.1:3306/bootboard
spring.datasource.username=root
spring.datasource.password=12345

# JPA Setting
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=false
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MariaDB103Dialect
spring.jpa.properties.hibernate.format_sql=true

# static resource
spring.mvc.static-path-pattern=/static/**

# Security log level Setting
logging.level.org.springframework.security=debug
```



스프링부트 시큐리티

- 시큐리티를 적용하지 않았을때

/static/hello.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
  <title>Hello~</title>
</head>
<body>
  <h2>Hello~ 스프링 부트!!</h2>
</body>
</html>
```

← → ↻ ⓘ localhost:8080/hello.html

Hello~ 스프링 부트!!



스프링부트 시큐리티

- 시큐리티 적용(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>
```

```
<dependency>  
  <groupId>org.thymeleaf.extras</groupId>  
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>  
</dependency>
```



스프링부트 시큐리티

- 시큐리티 적용

메인 앱을 실행하면 암호화된 비밀번호가 발급됨

```
2022-09-24 07:31:51.249 WARN 1424 --- [ restartedMain] ion$DefaultTemplateReso]
2022-09-24 07:31:51.331 WARN 1424 --- [ restartedMain] .s.s.UserDetailsService/
```

```
Using generated security password: e3eb1a8b-5d57-4f49-afba-49b18712a0ff
```

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

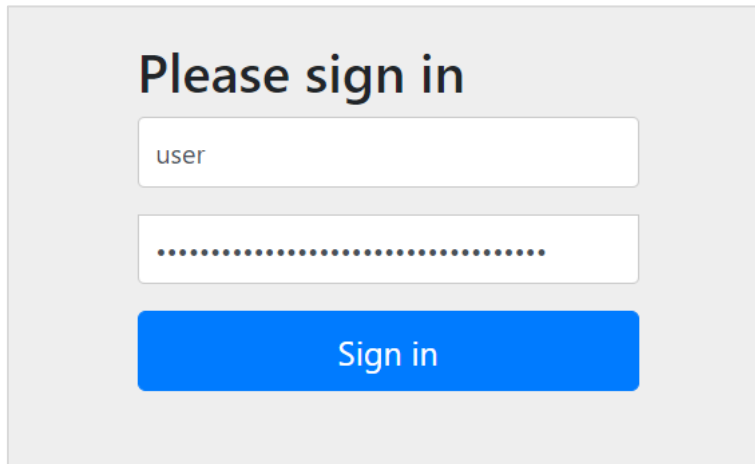


스프링부트 시큐리티

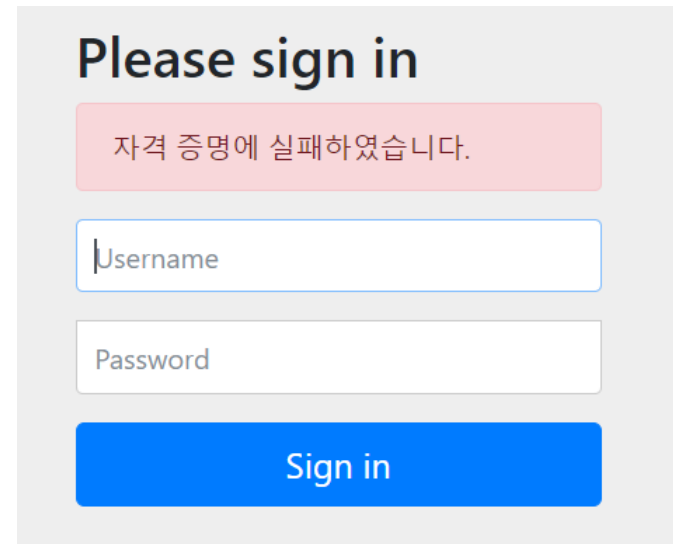
- 시큐리티 적용

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 page. It features a light gray background with the text "Please sign in" at the top. Below this, there are two input fields: the first contains the text "user", and the second is filled with dots to represent a password. At the bottom, there is a blue button with the text "Sign in".



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



스프링부트 시큐리티

- 시큐리티 커스터마이징

1. 시큐리티 설정파일

```
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

//시큐리티 설정 파일을 의미하며 시큐리티를 사용하는데 필요한 수많은 객체를 생성함
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Override
    protected void configure(HttpSecurity http) throws Exception {

    }

}
```

SecurityConfig 클래스가 빈으로 등록되기만 해도 애플리케이션에서는 더 이상 로그인을 강제하지 않는다.



스프링부트 시큐리티

- 시큐리티 커스터마이징

2. 시큐리티 화면 구성

요청 URL	의미
/	인증을 하지 않은 모든 사용자가 접근할 수 있다.
/member	인증을 통과한 사용자만 접근할 수 있다.
/manager	인증을 통과했고, MANAGER 권한을 가진 사용자만 접근할 수 있다.
/admin	인증을 통과했고, ADMIN 권한을 가진 사용자만 접근할 수 있다.



스프링부트 시큐리티

▪ SecurityController 클래스

```
@Slf4j
@Controller
public class SecurityController {

    @GetMapping("/")
    public String index() {
        log.info("index 요청입니다.");
        return "index";
    }

    @GetMapping("/member")
    public void forMember() {
        log.info("Member 요청입니다.");
    }
}
```

```
▼ BootSecurity [boot] [devtools]
  ▼ src/main/java
    > com.boot
  ▼ com.boot.config
    > SecurityConfig.java
  ▼ com.boot.controller
    > SecurityController.java
  ▼ src/main/resources
    ▼ static
      > css
        hello.html
    ▼ templates
      admin.html
      index.html
      manager.html
      member.html
    application.properties
```



스프링부트 시큐리티

- SecurityController 클래스

```
@GetMapping("/manager")  
public void forManager() {  
    Log.info("Manager 요청입니다.");  
}  
  
@GetMapping("/admin")  
public void forAdmin() {  
    Log.info("Admin 요청입니다.");  
}
```



2. 화면 작성

templates/index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
  <title>Hello~</title>
  <link rel="stylesheet" href="/static/css/style.css">
</head>
<body>
  <div id="container">
    <h2>인덱스 화면입니다.</h2>
  </div>
</body>
</html>
```



스프링부트 시큐리티

2. 화면 작성

templates/member.html

```
<div id="container">
  <h2>로그인 성공한 사용자만 접근할 수 있는 화면입니다.</h2>
  <a th:href="@{/loginSuccess}">뒤로 가기</a>
</div>
```

templates/manager.html

```
<div id="container">
  <h2>MANAGER 권한을 가진 사용자를 위한 화면입니다.</h2>
  <a th:href="@{/loginSuccess}">뒤로 가기</a>
  <p></p>
  <form th:action="@{/logout}" method="get">
    <input type="submit" value="로그아웃">
  </form>
</div>
```



스프링부트 시큐리티

2. 화면 작성

templates/admin.html

```
<div id="container">  
  <h2>ADMIN 권한을 가진 사용자를 위한 화면입니다.</h2>  
  <a th:href="@{/loginSuccess}">뒤로 가기</a>  
</div>
```



스프링부트 시큐리티

✓ HttpSecurity가 제공하는 주요 메소드

메소드	사용가능한 메소드	의미
authorizeRequests()		사용자 인증과 권한을 설정
	antMatchers("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
logout()		로그아웃 페이지 설정
	logoutUrl("/logout")	로그아웃을 처리하는 페이지 설정
csrf()		csrf는 크로스 사이트 위조 요청에 대한 설정
	disable()	RESTfull을 사용하기 위해서는 csrf 기능을 비활성화해야함



3. SecurityConfig 재정의

configure() 메소드를 수정하여 URL 경로와 권한을 설정함

```
//시큐리티 설정 파일을 의미하며 시큐리티를 사용하는데 필요한 수많은 객체를 생성함
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

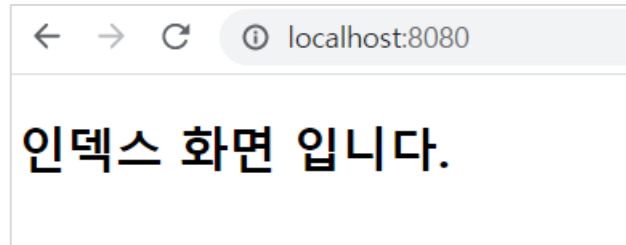
    @Override
    protected void configure(HttpSecurity security) throws Exception {
        /*security.authorizeRequests().antMatchers("/").permitAll();
        security.authorizeRequests().antMatchers("/member/**").authenticated();*/

        //AuthorizedUrl은 빌더 패턴을 사용
        security.authorizeRequests()
            .antMatchers("/").permitAll()
            .antMatchers("/member/**").authenticated()
            .antMatchers("/manager/**").hasRole("MANAGER")
            .antMatchers("/admin/**").hasRole("ADMIN");
    }
}
```

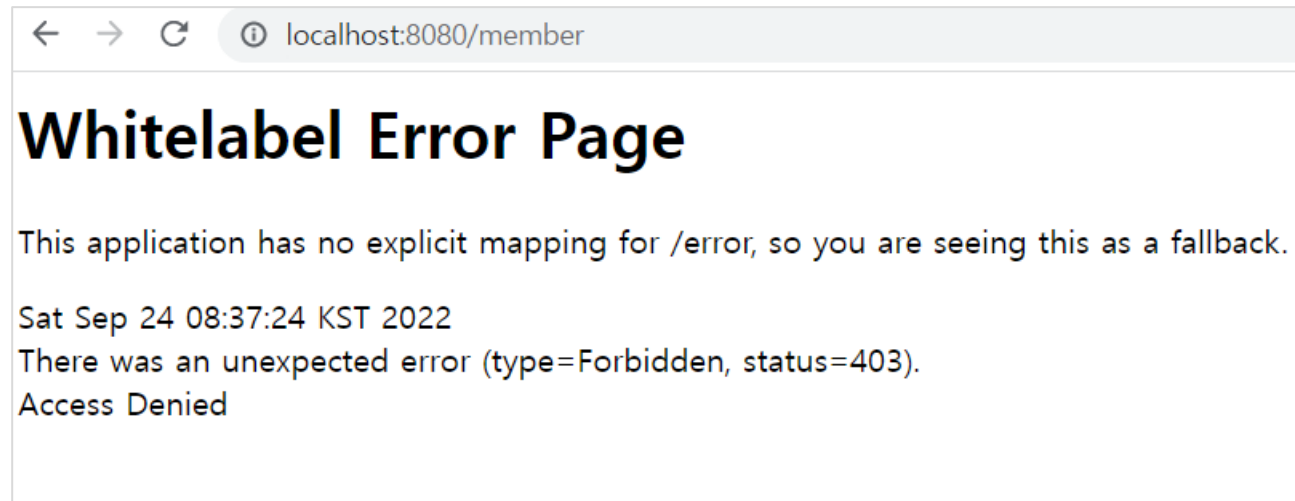


스프링부트 시큐리티

새로운 브라우저를 실행하고 '/' 요청



로그인 인증을 거쳐야만 하는 '/member', '/admin', '/manager'로 요청 -> **에러 발생**



스프링부트 시큐리티

■ 사용자 인증하기

인증된 회원, 즉 로그인에 성공한 사용자에게만 특정 페이지를 보여주기 위해서는 로그인 화면을 제공해야 함

```
//AuthorizedUrl은 빌더 패턴을 사용
security.authorizeRequests()
    .antMatchers("/").permitAll()
    .antMatchers("/member/**").authenticated()
    .antMatchers("/manager/**").hasRole("MANAGER")
    .antMatchers("/admin/**").hasRole("ADMIN");
```

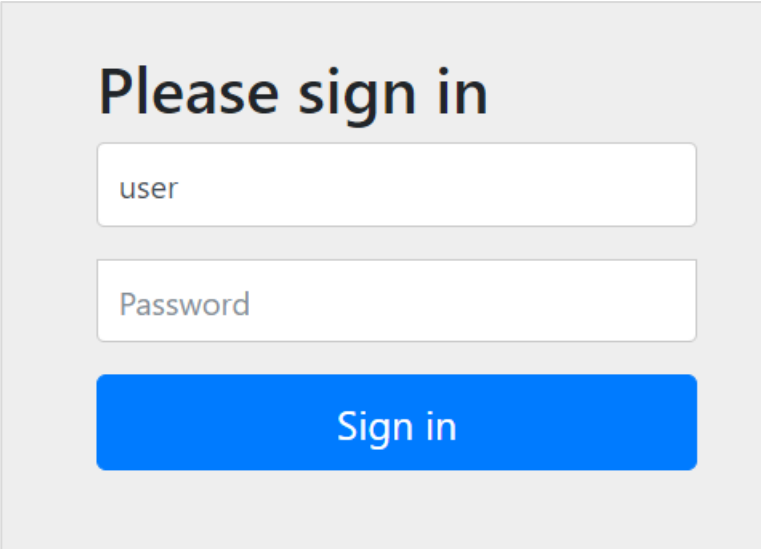
```
security.formLogin(); //시큐리티 기본 로그인 폼 사용
```



스프링부트 시큐리티

- 사용자 인증하기

로그인 인증을 거쳐야만 하는 '/member', '/admin', '/manager'로 요청



Please sign in

user

Password

Sign in



스프링부트 시큐리티

- 사용자가 직접 작성한 로그인 폼으로 인증하기

```
//security.formLogin(); //시큐리티 기본 로그인 폼 사용

security.csrf().disable(); //csrf() 비활성화 함

//사용자가 만든 로그인 폼이 실행되고, 결과 페이지로 이동
security.formLogin()
    .loginPage("/login")
    .defaultSuccessUrl("/loginSuccess", true);
```



- SecurityController 작성

```
@Slf4j
@Controller
public class SecurityController {

    @GetMapping("/login")
    public void login() {
        Log.info("login 요청입니다.");
    }

    @GetMapping("/loginSuccess")
    public void loginSuccess() {
        Log.info("login 성공입니다.");
    }
}
```



스프링부트 시큐리티

- 사용자가 직접 작성한 로그인 폼으로 인증하기

templates/login.html

```
<div id="container">
  <h2>로그인</h2>
  <form method="post">
    <table class="tbl_login">
      <tr>
        <td>아이디</td>
        <td><input type="text" name="username"></td>
      </tr>
      <tr>
        <td>비밀번호</td>
        <td><input type="password" name="password"></td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <button type="submit">로그인</button>
        </td>
      </tr>
    </table>
  </form>
</div>
```



스프링부트 시큐리티

- 사용자가 직접 작성한 로그인 폼으로 인증하기

templates/loginSuccess.html

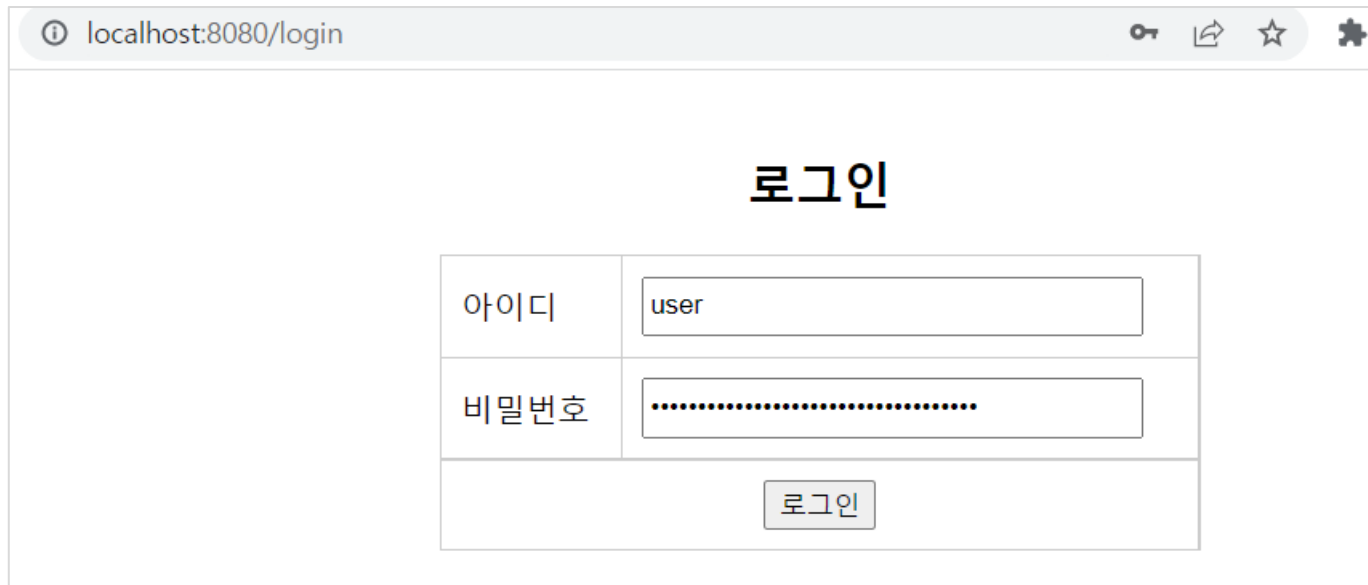
```
<div id="container">
  <h3>로그인 인증을 성공했습니다.</h3>
  <h4><a th:href="@{/}">INDEX 페이지로 이동</a></h4>
  <h4><a th:href="@{/member}">MEMBER 페이지로 이동</a></h4>
  <h4><a th:href="@{/manager}">MANAGER 페이지로 이동</a></h4>
  <h4><a th:href="@{/admin}">ADMIN 페이지로 이동</a></h4>
</div>
```



스프링부트 시큐리티

- 사용자 인증하기

'/member', '/admin', '/manager'로 요청



localhost:8080/login

로그인

아이디	<input type="text" value="user"/>
비밀번호	<input type="password" value="....."/>
<input type="button" value="로그인"/>	



- 사용자 인증하기

인증 성공 화면으로 이동

로그인 인증을 성공했습니다.

[INDEX 페이지로 이동](#)

[MEMBER 페이지로 이동](#)

[MANAGER 페이지로 이동](#)

[ADMIN 페이지로 이동](#)



스프링부트 시큐리티

■ 메모리 사용자 인증하기

```
//시큐리티 설정 파일을 의미하며 시큐리티를 사용하는데 필요한 수많은 객체를 생성함
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    //~ 생략 ~

    @Autowired
    public void authenticate(AuthenticationManagerBuilder auth) throws Exception
        /*메모리 사용자 인증하기
        AuthenticationManagerBuilder 객체를 의존성 주입받은
        authenticate() 메소드에서는 인증에 필요한 사용자 정보를 생성한다.*/
        auth.inMemoryAuthentication()
            .withUser("manager")           //사용자 아이디 설정
            .password("{noop}manager123")  //비밀번호에 대한 암호화 하지 않음
            .roles("MANAGER");             //권한 설정

        auth.inMemoryAuthentication()
            .withUser("admin")
            .password("{noop}admin123")
            .roles("ADMIN");
    }
```



스프링부트 시큐리티

■ 메모리 사용자 인증하기

새로운 브라우저를 열어서 “/member” 로 요청
manager/manager123 -> 로그인

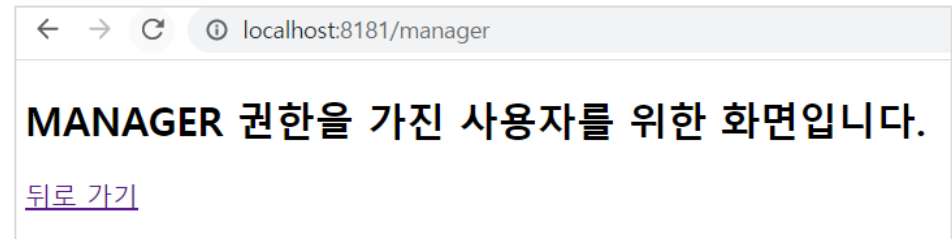
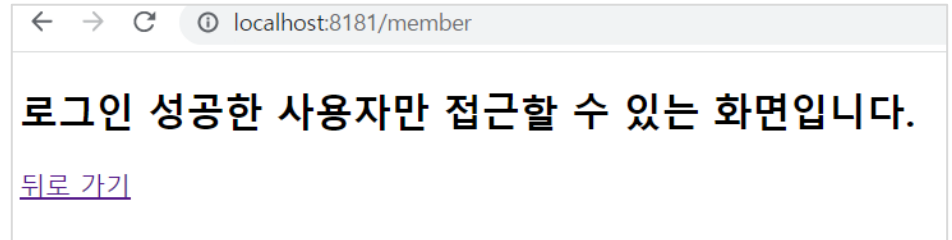
로그인 인증을 성공했습니다.

[INDEX 페이지로 이동](#)

[MEMBER 페이지로 이동](#)

[MANAGER 페이지로 이동](#)

[ADMIN 페이지로 이동](#)



스프링부트 시큐리티

- 접근 권한 없음 페이지 처리

SecurityConfig.java

```
//접근 권한 없음 페이지 처리  
security.exceptionHandling().accessDeniedPage("/accessDenied");
```

SecurityController.java

```
//접근 권한 없음 페이지  
@GetMapping("/accessDenied")  
public void accessDenied() {  
    Log.info("accessDenied 접근 거부");  
}
```



스프링부트 시큐리티

- 접근 권한 없음 페이지 처리

accessDenied.html

```
<link rel="stylesheet" href="/static/css/style.css">
</head>
<body>
  <div id="container">
    <h3>페이지에 대한 접근 권한이 없습니다.</h3>
    <h4>다시 로그인을 원하시면 <a th:href="@{/login}">여기(클릭)</a></h4>
  </div>
</body>
```

localhost:8181/admin

페이지에 대한 접근 권한이 없습니다.

다시 로그인을 원하시면 [여기\(클릭\)](#)



스프링부트 시큐리티

- 로그아웃 처리하기

SecurityConfig.java

```
//로그아웃 후 로그인 페이지로 이동  
security.logout().invalidateHttpSession(true).logoutSuccessUrl("/login");
```

manager.html

```
<h2>MANAGER 권한을 가진 사용자를 위한 화면입니다.</h2>  
<a th:href="@{/loginSuccess}">뒤로 가기</a>  
<p></p>  
<form th:action="@{/logout}" method="get">  
    <input type="submit" value="로그아웃">  
</form>
```



스프링부트 시큐리티

- 로그아웃 처리하기

MANAGER 권한을 가진 사용자를 위한 화면입니다.

[뒤로 가기](#)

로그아웃

로그인

아이디

비밀번호

로그인



스프링부트 시큐리티

▪ Mariadb 연동하기

member 테이블 생성 - 기존 테이블은 삭제함

기본 옵션 인덱스 (1) 외래 키 (0) 제약 조건 확인 (0) 분할 CREATE 코드 ALTER 코드

이름:

member

코멘트:

열:

+ 추가 × 제거 ▲ 위로 ▼ 아래로

#	이름	데이터 유형	길이/설정	부호 ...	NULL 허용	0으로 채움	기본값
1	id	VARCHAR	10	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	기본값 없음
2	password	VARCHAR	100	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	기본값 없음
3	name	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	기본값 없음
4	role	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	기본값 없음
5	enabled	TINYINT	1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL

- password의 길이는 암호화를 위해서 크게 지정함
- enabled 칼럼은 해당 계정이 사용 가능한지 여부를 저장함.



스프링부트 시큐리티

- 데이터베이스 연동하기
member 데이터 입력

```
-- member 테이블  
INSERT INTO member VALUES ('member', 'member12', '회원', 'ROLE_MEMBER', TRUE);  
INSERT INTO member VALUES ('manager', 'manager12', '매니저', 'ROLE_MANAGER', TRUE);  
INSERT INTO member VALUES ('admin', 'admin12', '어드민', 'ROLE_ADMIN', TRUE);
```

member (3r × 5c)					
id	password	name	role	enabled	
admin	admin12	어드민	ROLE_ADMIN	1	
manager	manager12	매니저	ROLE_MANAGER	1	
member	member12	회원	ROLE_MEMBER	1	



스프링부트 시큐리티

- 시큐리티 설정 수정 – jdbcAuthentication()으로 변경

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private DataSource dataSource;
```

```
@Autowired
public void authenticate(AuthenticationManagerBuilder auth) throws Exception {
    //데이터베이스에 저장된 사용자로 인증 처리
    //query1 : 사용자가 입력한 아이디로 사용자 정보 조회
    //아이디는 username에 비밀번호는 password 변수에 각각 저장함(Alias[별칭]를 적용)
    String query1 = "select id username, concat('{noop}', password) "
        + "password, true enabled from member where id=?";
    //query2 : 사용자가 입력한 아이디로 권한 정보 조회
    String query2 = "select id, role from member where id=?";

    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .usersByUsernameQuery(query1)
        .authoritiesByUsernameQuery(query2);
}
```



스프링부트 시큐리티

■ 시큐리티 테스트

localhost:8181/manager로 요청
manager/manager12 -> 로그인

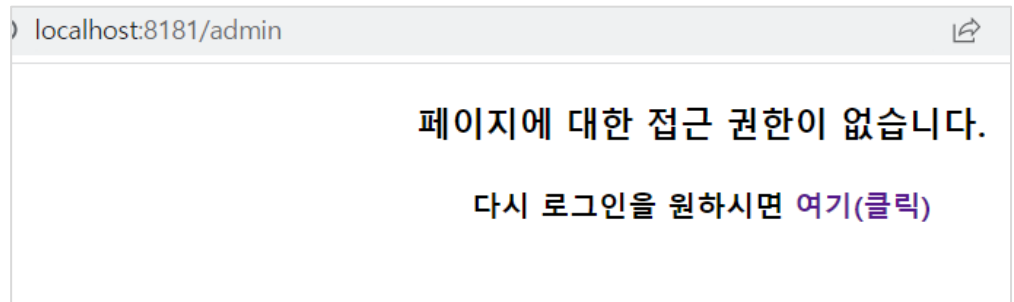
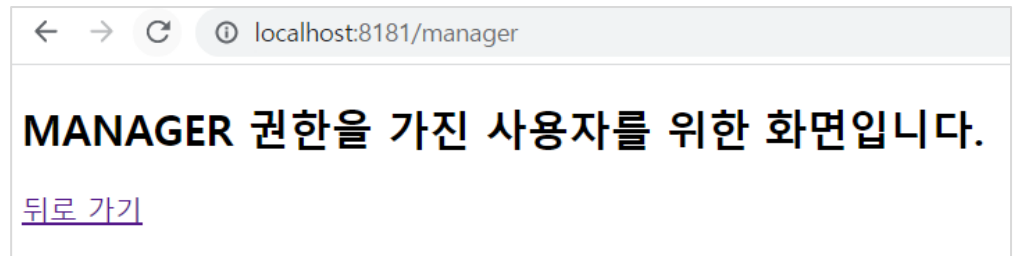
로그인 인증을 성공했습니다.

[INDEX 페이지로 이동](#)

[MEMBER 페이지로 이동](#)

[MANAGER 페이지로 이동](#)

[ADMIN 페이지로 이동](#)



스프링부트 시큐리티

- JPA 연동하기
 - 엔티티 클래스

```
public enum Role {  
    ROLE_ADMIN, ROLE_MANAGER, ROLE_MEMBER  
}
```

```
@Data  
@Entity  
public class Member {  
    @Id  
    private String id;  
    private String password;  
    private String name;  
  
    @Enumerated(EnumType.STRING)  
    private Role role;  
  
    private boolean enabled;  
}
```



스프링부트 시큐리티

- JPA 연동하기
 - 리포지토리

```
package com.boot.persistence;

import org.springframework.data.jpa.repository.JpaRepository;

public interface MemberRepository extends JpaRepository<Member, String>{

}
```



- 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 이클래스가 제공하는 UserDetails 객체는 아이디가 'user' 이고 비밀번호는 암호화 되어 콘솔에 출력되는 긴 문자열이다.

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



스프링부트 시큐리티

1. UserDetailsService 구현 클래스 작성

```
@Service
public class BoardUserDetailsService implements UserDetailsService{

    @Autowired
    private MemberRepository memberRepo;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //MemberRepository로 회원 정보를 조회하여
        //UserDetails 타입의 객체로 리턴한다.
        Optional<Member> optional = memberRepo.findById(username);
        if(!optional.isPresent()) {
            throw new UsernameNotFoundException(username + "사용자 없음");
        }else {
            Member member = optional.get();
            return new SecurityUser(member);
        }
    }
}
```



스프링부트 시큐리티

▪ SecurityUser 클래스

User 클래스를 상속했으며, User 클래스의 생성자를 호출할때 검색 결과로 얻은 Member 객체를 전달한다.

```
public class SecurityUser extends User{

    private static final long serialVersionUID = 1L;

    public SecurityUser(Member member) {
        super(member.getId(), "{noop}" + member.getPassword(),
            AuthorityUtils.createAuthorityList(member.getRole().toString()));
    }
}
```



스프링부트 시큐리티

- 사용자 정의 UserDetailsService 적용하기

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private BoardUserDetailsService boardUserDetailsService;

    @Override
    protected void configure(HttpSecurity security) throws Exception {
```

```
        //스프링 시큐리티가 제공하는 UserDetailsService가 아닌 사용자 정의 UserDetailsService 사용
        security.userDetailsService(boardUserDetailsService);
    }
```



- 사용자 정의 UserDetailsService 적용하기

localhost:8080/manager로 요청
manager/manager12 -> 로그인

로그인 인증을 성공했습니다.

[INDEX 페이지로 이동](#)

[MEMBER 페이지로 이동](#)

[MANAGER 페이지로 이동](#)

[ADMIN 페이지로 이동](#)



스프링부트 시큐리티

- PasswordEncoder 사용하기
 - 비밀번호 암호화

```
package com.boot.config;

import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;

import com.boot.domain.Member;

public class SecurityUser extends User{

    private static final long serialVersionUID = 1L;

    public SecurityUser(Member member) {
        super(member.getId(), member.getPassword(),
            AuthorityUtils.createAuthorityList(member.getRole().toString()));
    }
}
```



스프링부트 시큐리티

- PasswordEncoder 사용하기
 - 비밀번호 암호화 적용하기

SecurityConfig.java

```
//암호화 - PasswordEncoder 객체를 리턴하는 passwordEncoder() 추가
@Bean
public PasswordEncoder passwordEncoder() {
    return PasswordEncoderFactories.createDelegatingPasswordEncoder();
}
```



▪ PasswordEncoder 테스트

```
@SpringBootTest
public class PasswordEncoderTest {

    @Autowired
    private MemberRepository memberRepo;

    @Autowired
    private PasswordEncoder encoder;

    @Test
    public void testInsert() {
        Member member = new Member();
        member.setId("manager2");
        member.setPassword(encoder.encode("manager222"));
        member.setName("매니저2");
        member.setRole(Role.ROLE_MANAGER);
        member.setEnabled(true);

        memberRepo.save(member);
    }
}
```



스프링부트 시큐리티

▪ PasswordEncoder 테스트

id	password	name	role	enabled
admin	admin12	어드민	ROLE_ADMIN	1
manager	manager12	매니저	ROLE_MANAGER	1
manager2	{bcrypt}\$2a\$10\$luJuD9O/BqeljA8qFAoc3uyj.44E4....	매니저2	ROLE_MANAGER	1
member	member12	회원	ROLE_MEMBER	1

{noop} 제거

```
public SecurityUser(Member member) {  
    //password를 암호화하여 저장한 경우는 {noop} 제거함  
    super(member.getId(), member.getPassword(),  
        AuthorityUtils.createAuthorityList(member.getRole().toString()));  
  
    /*super(member.getId(), "{noop}" + member.getPassword(),  
        AuthorityUtils.createAuthorityList(member.getRole().toString()));*/  
}
```



스프링부트 시큐리티

▪ PasswordEncoder 테스트

localhost:8181/manager로 요청
manager2/manager222 -> 로그인

로그인 인증을 성공했습니다.

[INDEX 페이지로 이동](#)

[MEMBER 페이지로 이동](#)

[MANAGER 페이지로 이동](#)

[ADMIN 페이지로 이동](#)

