Spring Boot Application

스프링 부트 응용

스프링 시큐리티 적용

한국기술교육대학교
온라인평생교육원

학습내용

- 스프링 시큐리티 적용
- 스프링 시큐리티 커스터마이징

학습목표

- 스프링 부트에서 시큐리티를 적용할 수 있다.
- 스프링 시큐리티를 커스터마이징할 수 있다.

스프링 시큐리티 적용

1 인증(Authentication)과 인가(Authorization)

〈1〉 인증과 인가 개념

- 대부분의 애플리케이션에서는 인증을 통해 사용자를 식별하고, 인가를 통해 시스템 자원을 통제함

1 인증(Authentication)과 인가(Authorization)

〈1〉 인증과 인가 개념

인증 (Authentication)	• 시스템 사용자를 식별하는 것 (로그인 처리)
인가 (Authorization)	• 인증된 사용자가 해당 기능을 실행할 권한이 있는지 확인하는 것 (권한체크)

스프링 시큐리티 적용

1 인증(Authentication)과 인가(Authorization)

2 일반적인 인증

- 웹 애플리케이션에서는 일반적으로 세션(HttpSession)을 기반으로 인증과 인가를 처리함

1 인증(Authentication)과 인가(Authorization)

2 일반적인 인증

- getBoardList 메소드에서 글목록을 요청한 브라우저의 인증 상태를 확인하는 코드

```
@RequestMapping("/board/getBoardList")
public String getBoardList(Model model, HttpSession session) {
    if(session.getAttribute("member") == null) return "redirect:/";
    model.addAttribute("boardList", boardService.getBoardList());
    return "board/getBoardList";
}
```

모든 메소드에서 반복

스프링 시큐리티 적용

1 인증(Authentication)과 인가(Authorization)

2 일반적인 인증

- 글삭제 요청한 사용자의 권한이 'ADMIN'인지 체크하는 코드

```
<p th:align="center">
<a th:href="@{insertBoard}">글등록</a>   
<a th:if="${session['member'].role == 'ADMIN'}"
   th:href="@{deleteBoard(seq=${board.seq})}">글삭제</a>   
<a th:href="@{getBoardList}">글목록</a>
</p>
```

1 인증(Authentication)과 인가(Authorization)

2 일반적인 인증

- 인증과 인가에 관련된 코드가 흩어져 있으면
  유지보수가 어려울 뿐만 아니라
  정책을 변경하기도 어려움

스프링 시큐리티 적용

2 스프링 시큐리티 적용

〈1〉 시큐리티 스타터 추가

- 스프링 부트 프로젝트에 시큐리티를 적용하기 위해 pom.xml 편집 화면에서 시큐리티 스타터를 추가함


```
    <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>
</dependencies>
```

2 스프링 시큐리티 적용

〈1〉 자동 생성 계정

1. 시큐리티 스타터 추가
2. 애플리케이션 다시 실행
3. 시큐리티 관련 자동 설정 클래스 동작
4. 시큐리티와 관련된 다양한 객체 생성

스프링 시큐리티 적용

2 스프링 시큐리티 적용

〈2〉 시큐리티 스타터 추가

- application.yml 파일을 통해 스프링 시큐리티 관련 로깅 레벨을 변경함
- debug로 해야 상세한 정보를 확인할 수 있음

```
# Logging Level 설정
logging:
  level:
    org:
      hibernate: info
# 동일한 설정 : '[org.hibernate]': debug
      springframework:
        security: debug
```

2 스프링 시큐리티 적용

〈3〉 자동 생성 계정

```
2021-08-16 11:38:52.446  INFO 5232 --- [  restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 071166b7-547f-4ac6-a136-2d6a067becb8

2021-08-16 11:38:52.566  INFO 5232 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     :
2021-08-16 11:38:52.613  INFO 5232 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       :
2021-08-16 11:38:52.683  INFO 5232 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  :
2021-08-16 11:38:52.692  INFO 5232 --- [  restartedMain] com.mycompany.BoardWebApplication        :
```

스프링 시큐리티 적용

2 스프링 시큐리티 적용

⟨4⟩ 기본 로그인 인증

- 이제부터 브라우저의 모든 요청은 스프링 시큐리티가 제공하는 기본 로그인 화면으로 연결됨

2 스프링 시큐리티 적용

⟨4⟩ 기본 로그인 인증

스프링 시큐리티가 제공하는 로그인 화면에서 로그인 인증

- 아이디 : 'user'
- 비밀번호 : 서버 구동 시에 출력된 비밀번호

스프링 시큐리티 적용

2 스프링 시큐리티 적용

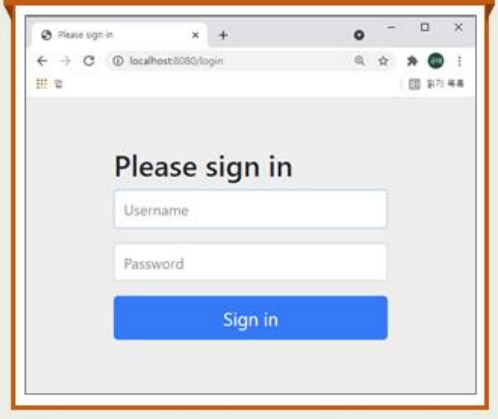
④ 기본 로그인 인증

- 인증에 실패하면 간단한 메시지와 함께 다시 인증 화면으로 되돌아 옴

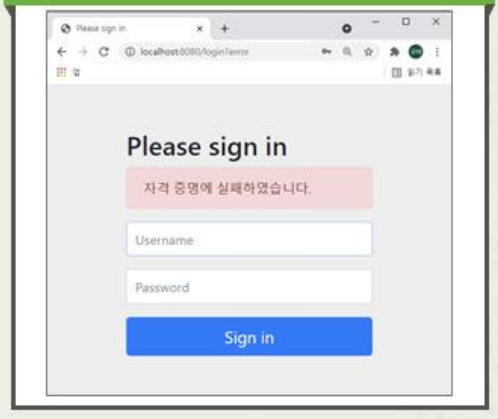
2 스프링 시큐리티 적용

④ 기본 로그인 인증

로그인 화면



로그인 실패 화면



스프링 시큐리티 적용

3 스프링 시큐리티 동작 원리

〈1〉 시큐리티 필터

스프링 시큐리티

- 서블릿의 필터(javax.servlet.Filter) 기술을 기반으로 개발됨

3 스프링 시큐리티 동작 원리

〈1〉 시큐리티 필터

서블릿 필터

- 클라이언트의 서블릿이 수행되기 전에 서블릿 요청을 가로채서 전처리와 후처리를 담당함

스프링 시큐리티 적용

3 스프링 시큐리티 동작 원리

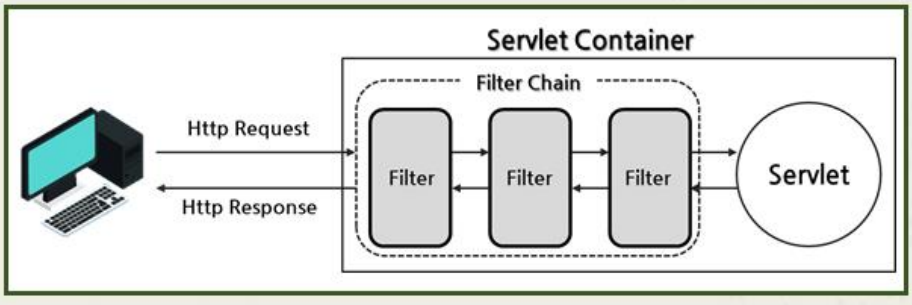
〈1〉 시큐리티 필터

- 일반적으로 필터 한 개당 하나의 기능을 처리하기 때문에 여러 개의 필터를 만들어 필터 체인을 구성함

3 스프링 시큐리티 동작 원리

〈1〉 시큐리티 필터

- 스프링 시큐리티는 시큐리티 처리와 관련된 다양한 기능을 필터 체인을 통해 제공함



스프링 시큐리티 적용

3 스프링 시큐리티 동작 원리

② 시큐리티 필터 체인

- 스프링 시큐리티가 제공하는 필터 중에서 중요한 필터들

필터	기능
SecurityContextPersistenceFilter	SecurityContextRepository에서 SecurityContext 객체를 로딩하여 SecurityContextHolder에 저장하고 요청 처리가 끝나면 제거함
LogoutFilter	로그아웃을 처리하고 지정한 페이지로 이동함
UsernamePasswordAuthenticationFilter	• 아이디/비밀번호 기반의 인증을 처리함 • 인증에 성공하면 지정한 페이지로 이동하고, 실패하면 다시 로그인 화면을 제공함

3 스프링 시큐리티 동작 원리

② 시큐리티 필터 체인

- 스프링 시큐리티가 제공하는 필터 중에서 중요한 필터들

필터	기능
DefaultLoginPageGeneratingFilter	기본 로그인 페이지를 제공함
AnonymousAuthenticationFilter	• 이 필터가 실행되는 시점까지 사용자가 인증을 받지 못했다면 Authentication 객체를 생성하여 SecurityContext에 설정함 • 생성된 Authentication의 아이디는 “anonymouseUser”, 권한은 “ROLE_ANONYMOUS”이며, 인증되지 않은 상태의 값을 가짐

스프링 시큐리티 적용

3 스프링 시큐리티 동작 원리

〈2〉 시큐리티 필터 체인

- 스프링 시큐리티가 제공하는 필터 중에서 중요한 필터들

필터	기능
FilterSecurityInterceptor	• 현재 사용자가 지정한 경로에 접근할 수 있는지 여부를 검사함 • 권한이 있으면 보안 필터를 통과시켜 자원에 접근할 수 있게 하고, 권한이 없으면 예외를 발생시킴
ExceptionTranslationFilter	FilterSecurityInterceptor에서 발생한 예외를 웹에 맞는 응답으로 변환함

3 스프링 시큐리티 동작 원리

〈3〉 스프링 시큐리티 동작 원리

- UsernamePasswordAuthenticationFilter

  • 사용자가 입력한 아이디와 비밀번호를 이용해서 사용자를 인증하는 필터
  • 시큐리티에서 가장 중요함

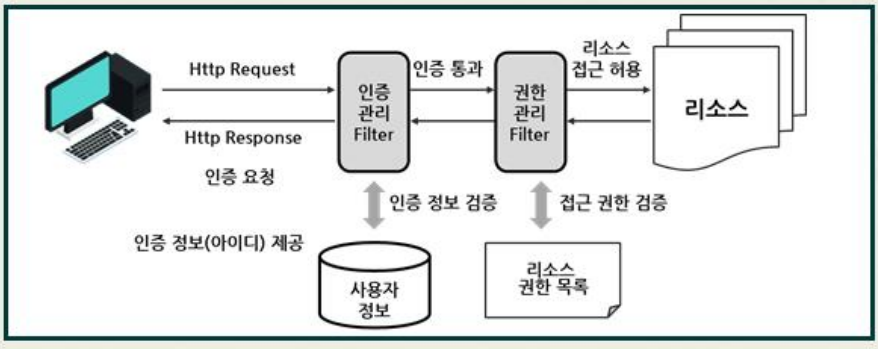
스프링 시큐리티 적용

3 스프링 시큐리티 동작 원리

〈3〉 스프링 시큐리티 동작 원리

- FilterSecurityInterceptor

> • 인증에 성공한 사용자가 해당 리소스에 접근할 권한이 있는지 검증함



스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

⟨1⟩ 시큐리티 테스트를 위한 사전작업

- 스프링 시큐리티를 커스터마이징하기 위해서 시큐리티 설정 클래스를 작성함

```
package com.mycompany.security.config;

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;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity security) throws Exception {

    }
}
```

1 스프링 시큐리티 커스터마이징

⟨1⟩ 시큐리티 테스트를 위한 사전작업

1 @EnableWebSecurity

- 시큐리티 클래스를 자동설정 클래스로 선언하는 어노테이션

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

〈1〉 시큐리티 테스트를 위한 사전작업

2 WebSecurityConfigurerAdapter 클래스

- 스프링 시큐리티가 제공하는 기본 설정을 커스터마이징할 수 있는 configure 메소드를 제공함

1 스프링 시큐리티 커스터마이징

〈1〉 시큐리티 테스트를 위한 사전작업

2 WebSecurityConfigurerAdapter 클래스

- configure 메소드 매개변수로 받은 HttpSecurity 객체를 통해 인증과 인가에 필요한 다양한 설정을 재정의할 수 있음

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

〈2〉 메모리 사용자 등록

- AuthenticationManagerBuilder를 이용하여 시큐리티를 테스트하기 위한 사용자 계정을 생성함

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    public void authenticate(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication().withUser("test").password("{noop}test123").roles("MANAGER");
    }

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

1 스프링 시큐리티 커스터마이징

〈2〉 메모리 사용자 등록

- AuthenticationManagerBuilder.inMemoryAuthentication 메소드를 이용하면 간단하게 메모리에 테스트용의 사용자를 생성할 수 있음

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

〈2〉 메모리 사용자 등록

withUser	• 아이디 설정
password	• 비밀번호 설정({noop}은 비밀번호에 대한 암호화 처리 생략)
roles	• 권한 설정 (배열로 여러 권한을 설정할 수 있음)

1 스프링 시큐리티 커스터마이징

〈3〉 접근 제어

- configure 메소드에서는 권한 체크를 통해 특정 경로에 대한 사용자 접근을 통제할 수 있음

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

3 접근 제어

```
@Override
protected void configure(HttpSecurity security) throws Exception {
    // 접근 제어
    security.authorizeRequests().antMatchers("/").permitAll();
    security.authorizeRequests().antMatchers("/board/**").authenticated();
    security.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");
}
```

1

1 로그인 성공 여부와 상관 없이 모든 사용자의 '/' 접근을 허용

1 스프링 시큐리티 커스터마이징

3 접근 제어

```
@Override
protected void configure(HttpSecurity security) throws Exception {
    // 접근 제어
    security.authorizeRequests().antMatchers("/").permitAll();
    security.authorizeRequests().antMatchers("/board/**").authenticated();
    security.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");
}
```

2

1 로그인 성공 여부와 상관 없이 모든 사용자의 '/' 접근을 허용

2 로그인 성공한 사용자에 대해서만 '/board/**' 경로에 대한 접근을 허용

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

③ 접근 제어

```
@Override
protected void configure(HttpSecurity security) throws Exception {
    // 접근 제어
    security.authorizeRequests().antMatchers("/").permitAll();
    security.authorizeRequests().antMatchers("/board/**").authenticated();
    security.authorizeRequests().antMatchers("/manager/**").hasRole("MANAGER");
}
```

3

1 로그인 성공 여부와 상관 없이 모든 사용자의 '/' 접근을 허용

2 로그인 성공한 사용자에 대해서만 '/board/**' 경로에 대한 접근을 허용

3 로그인 성공했으면서 'MANAGER' 권한을 가진 사용자에 대해서만 '/manager/**' 경로에 대한 접근을 허용

1 스프링 시큐리티 커스터마이징

④ CSRF 인증 비활성화

1 CSRF(Cross Site Request Forgery)란?

- CSRF는 사이트간 요청 위조를 의미함
- 웹 애플리케이션 사용자의 의지와 상관 없이 공격자가 의도한 행동을 하도록 하여 강제로 데이터를 수정 또는 삭제하도록 만드는 공격방법

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

⟨4⟩ CSRF 인증 비활성화

2 CSRF 인증 비활성화 설정

- 스프링 시큐리티는 CSRF 공격을 방어하기 위해 기본적으로 CSRF 토큰을 사용함

1 스프링 시큐리티 커스터마이징

⟨4⟩ CSRF 인증 비활성화

2 CSRF 인증 비활성화 설정

- 테스트 과정에서 CSRF 인증을 사용하지 않을 경우

- 간단한 테스트를 위해 configure 메소드에서 csrf().disable() 메소드를 통해 CSRF 인증을 비활성화 함

```
@Override
protected void configure(HttpSecurity security) throws Exception {
    // CSRF 인증 비활성화
    security.csrf().disable();
}
```

스프링 시큐리티 커스터마이징

1 스프링 시큐리티 커스터마이징

⑤ 로그인 인증

- 사용자에게 스프링 시큐리티가 제공하는 기본 로그인 화면을 제공하기 위해서 formLogin 메소드를 설정함

1 스프링 시큐리티 커스터마이징

⑤ 로그인 인증

- 이제 메모리 사용자(test/test123)를 이용하여 시스템에 로그인할 수 있음

```
@Override
protected void configure(HttpSecurity security) throws Exception {
    // CSRF 인증 비활성화
    security.csrf().disable();
    // 로그인 화면 제공
    security.formLogin();
}
```

스프링 시큐리티 커스터마이징

2 사용자 정의 로그인 화면

〈1〉 로그인 화면 수정

- 스프링 시큐리티가 제공하는 로그인 화면이 아닌 사용자가 직접 정의한 로그인 화면을 사용할 수 있음

2 사용자 정의 로그인 화면

〈1〉 로그인 화면 수정

- templates/board/login.html 파일에서 아이디에 해당하는 〈input〉 태그의 name 속성 값을 "id"에서 "username"으로 변경함

```
<form th:action="login" method="post">
<table th:align="center" border="1" th:cellpadding="0" th:cellspacing="0">
<tr>
<td bgcolor="orange" th:text="아이디"></td>
<td><input name="username" type="text" size="10"></td>
</tr>
<tr>
<td bgcolor="orange" th:text="비밀번호"></td>
<td><input name="password" type="password" size="30"></td>
</tr>
<tr><td colspan="2" align="center"><input type="submit" value="로그인"></td></tr>
</table>
</form>
```

스프링 시큐리티 커스터마이징

2 사용자 정의 로그인 화면

〈2〉 화면 이동 컨트롤러 작성

- 로그인 화면으로 이동하는 요청(board/login)이 들어왔을 때 로그인 화면으로 이동하는 컨트롤러를 작성함

```
package com.mycompany.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class SecurityController {

    @RequestMapping("board/login")
    public void loginView() {
    }
}
```

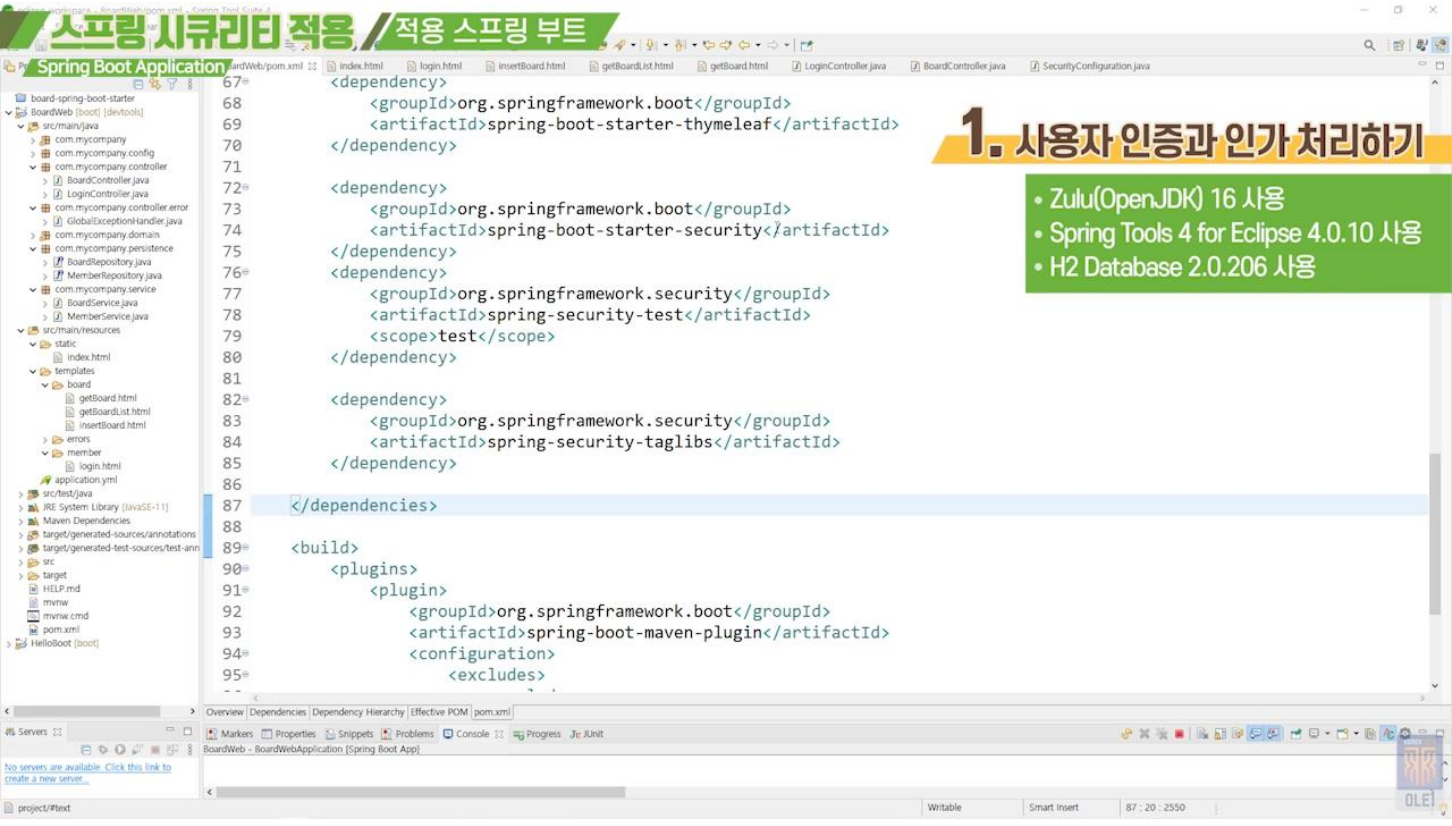
2 사용자 정의 로그인 화면

〈3〉 환경 설정 수정

- SecurityConfig.configure 메소드에 시큐리티가 기본으로 제공하는 화면이 아닌 사용자가 정의한 화면으로 이동할 수 있도록 loginPage 메소드를 추가로 설정함

```
// 로그인 화면 제공
security.formLogin().loginPage("/board/login");
```

적용 스프링 부트



실습단계
스프링 시큐리티를 이용하여 사용자 인증과 인가 커스터마이징
pom.xml → 시큐리티 스타터 추가
테스트 관련 라이브러리 추가 다운로드
mvnrepository.com 사이트를 통해 검색해서 추가해야 함
Custom 태그를 사용할 수 있게 라이브러리 추가
회원, 로그인 관련 요청과 게시글 관련 요청 구분을 위해 링크 수정
로그인 관련 member, 게시글 관련 board를 경로 상 추가
index.html 수정
경로 앞에 member 추가
스프링 시큐리티에서는 로그인 인증 시 사용자가 입력한 ID를 username 파라미터로 처리함
경로 앞에 board 추가
중요! 상세화면이나 새글 등록 화면으로 이동할 때 앞에 board 경로 추가
게시글 상세는 수정 버튼을 눌렀을 때 update 요청

적용 스프링 부트

실습단계
경로 상에 board 추가
중요! LoginController.java 수정
HTML 파일 위치가 바뀌어야 함
templates → board → board 관련 HTML, member 관련 HTML
BoardController.java에도 board 관련 경로 추가
글목록 이동에도 경로 앞에 board 추가
시큐리티 환경 설정 클래스 작성
com.mycompany.config 패키지 추가
SecurityConfiguration.java 환경 설정 클래스 작성
상속받는 클래스 WebSecurityConfigurerAdapter
오버라이딩 했기 때문에 몇몇 자동설정 클래스가 적용되지 않음
무조건 화면으로부터 인증할 때 받게 되어 있음
사용자의 요청을 핸들링할 수 있음
board 경로로 시작하는 모든 요청은 인증에 성공했을 때만 접근할 수 있음
로그인 화면 이동 테스트
그러나 board로 시작하는 접근은 허용되지 않음!
시큐리티 설정 클래스에 코드 추가
중요! 사실상 LoginController가 필요가 없는 설정
비밀번호 복사
중요! 아이디는 user
LoginController가 동작하지 않음
스프링 시큐리티가 제공하는 계정에는 이름이 없음