



추석맞이 스프링 시큐리티 연습해보기

작성자 : 아라한사

2015-09-29

페북 : <https://fb.com/me.adunhansa>

트위터 : <https://twitter.com/arahansa>

블로그 : <http://adunhansa.tistory.com/>

사이트 : <http://arahansa.com>





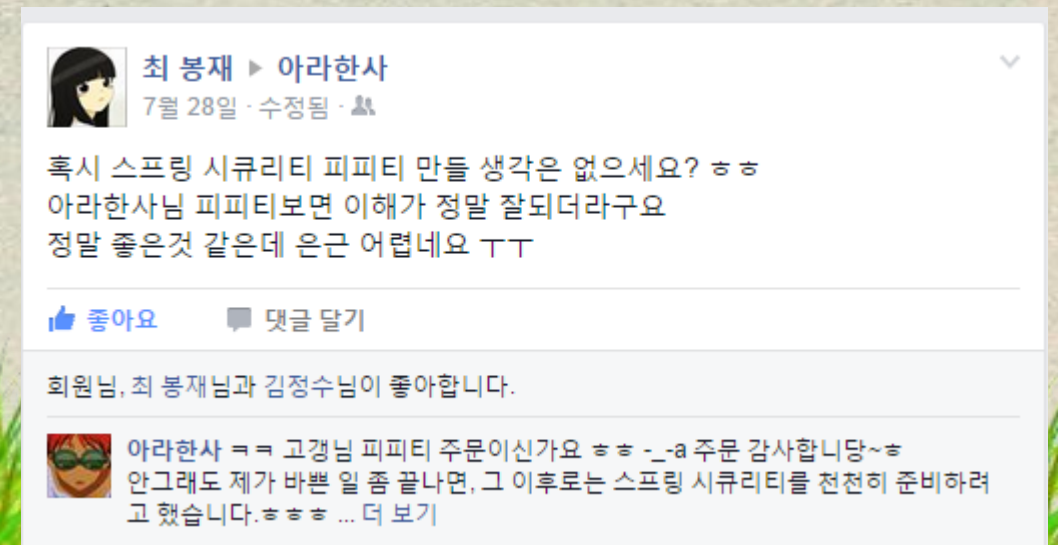
이름 : 아라한사

주력&관심 기술 : Java, Spring, ORM, Go

**평소 정리를 즐겨합니다. 한 때 동영상강좌도
만들다보니.. 비실명과 만화캐릭터를 쓰네요
양해 부탁드립니다 ㅎㅎ**

1. 작성자 소개

- 추석을 맞이해서 흠 뭘 할까 고민을 하다가 좀 필요한 일도 있기도해서 스프링 시큐리티를 좀 봐보기로 했습니다.
- 그전에는 돌아가는 소스를 복붙해서 그냥 시큐리티 붙여쓰고 그랬는데, 이참에 조~~금만더 알아보기로 하였습니다.
- 아아~;; 사실 마음은 예전부터; 있긴했어요 ^.^a
봉재님 감사해요 ;ㅁ;



**학습해보면서 만드는 슬라이드입니다.
잘못된 부분이 있어도 책임을 못 집니다.**

π.T

- 음 우선 하고 싶은 일을 정의해보겠습니다.

꼭 데이터베이스로 저장될 필요가 없는 경우도 있어서 **인메모리 로그인**을 하고 싶은 경우와~

로그인, 로그아웃을 해봐서 **권한에 따른 경로 설정**도 좀 하고 싶고,

데이터베이스에 사용자 정보도 넣어서 로그인 처리도 하고~

Remember Me 도 좀 해서 한번 로그인 하면 다음에는 로그인할 필요도 없게 하고 싶습니다.

아 **패스워드도 암호화** 해야겠죠?^^;



하고 싶은 일을 생각해보자 !

- 저도 그 전부터 사실 몇번 시큐리티 관련 블로깅들을 훑어보고 , 책에 있는 챕터를 훑어봤는데 어려워요 어려워요 ㅠ흠흠.
- 그냥 요구사항을 하나씩 구현해보면서 감을 잡아보려고 합니다~~

아.. 그냥 시나리오대로 따라가려고 했는데-_-뒤늦게 적고나서 목차를 좀 적어야겠습니다. 이 슬라이드는 5개의 프로젝트로 구성되어있습니다.

1. `inmemoryLogin` - 시큐리티의 기본적인 설정과 인증을 보게 됩니다. 로그인 폼 커스터마이징도 들어갑니다.
2. `jdbcLogin` - 여기서는 기본적으로 제공하는 JDBC테이블에 맞춰서 로그인을 해보고 jsp페이지에서 유저정보에 접근해봅니다.
3. `customizingLogin` - JPA로 사용자 엔티티를 만들어서 커스터마이징된 테이블에 정보를 담고 로그인해봅니다. 스프링 시큐리티의 내부 클래스를 조금 설명합니다.

4. **rememberMeLogin** - 스프링 시큐리티에서 리멤버미 토큰을 설정하고, 쿠키의 토큰방식과 DB로 저장하는 방식을 살펴보겠습니다.

5. **etcSecurity** 그밖에 정적자원관리, 비밀번호 암호화, @Preauthorize 스프링4에서 나온 기능에서 테스트와 Spring data 와의 통합에 관해 간략히 알아봅니다.

1. 인메모리 로그인 해보기 !

(깃헙의 inmoryLogin폴더입니다)

시작!

- STS에서 Spring boot 를 쉽게 만드는 Spring starter project로 만들어보겠습니다. 요거는 항상 보시던 화면이니, 이제 이 부분 설명은 패..패쓰;하겠습니다. ㅎㅎ;

New Spring Starter Project

Boot Version: 1.2.6

Dependencies:

Cloud	Core	Data	Database	I/O	Ops	Social	Template Engines	Web
<input type="checkbox"/> AWS	<input type="checkbox"/> AOP	<input type="checkbox"/> Elasticsearch	<input type="checkbox"/> Apache Derby	<input type="checkbox"/> AMQP	<input type="checkbox"/> Actuator	<input type="checkbox"/> Facebook	<input type="checkbox"/> Freemarker	<input type="checkbox"/> HATEOAS
<input type="checkbox"/> Cloud Connectors	<input type="checkbox"/> Lombok	<input type="checkbox"/> Redis	<input type="checkbox"/> H2	<input type="checkbox"/> Batch	<input type="checkbox"/> Actuator Docs	<input type="checkbox"/> LinkedIn	<input type="checkbox"/> Groovy Templates	<input type="checkbox"/> Jersey (JAX-RS)
<input type="checkbox"/> Cluster Zookeeper	<input checked="" type="checkbox"/> Security	<input type="checkbox"/> Solr	<input type="checkbox"/> HSQLDB	<input type="checkbox"/> Integration	<input type="checkbox"/> Remote Shell	<input type="checkbox"/> Twitter	<input type="checkbox"/> Mustache	<input type="checkbox"/> Mobile
<input type="checkbox"/> Consul Discovery	<input type="checkbox"/> Atomikos (JTA)	<input type="checkbox"/> JDBC	<input type="checkbox"/> MySQL	<input type="checkbox"/> JMS			<input type="checkbox"/> Thymeleaf	<input type="checkbox"/> REST Docs
<input type="checkbox"/> Hystrix Dashboard	<input type="checkbox"/> Bitronix (JTA)	<input checked="" type="checkbox"/> JPA	<input type="checkbox"/> PostgreSQL	<input type="checkbox"/> Mail			<input type="checkbox"/> Velocity	<input type="checkbox"/> Rest Repositories
<input type="checkbox"/> Stream Rabbit	<input type="checkbox"/> Cache	<input type="checkbox"/> MongoDB						<input type="checkbox"/> Rest Repositories HAL Browser
<input type="checkbox"/> Zookeeper Configuration	<input type="checkbox"/> DevTools							<input type="checkbox"/> Vaadin
<input type="checkbox"/> AWS JDBC								<input type="checkbox"/> WS
<input type="checkbox"/> AWS Messaging								<input type="checkbox"/> Web
<input type="checkbox"/> Cloud Bootstrap								<input type="checkbox"/> Websocket
<input type="checkbox"/> Cluster Hazelcast								
<input type="checkbox"/> Consul Bus								
<input type="checkbox"/> Feign								
<input type="checkbox"/> Sleuth								
<input type="checkbox"/> Turbine AMQP								
<input type="checkbox"/> Cloud Bus AMQP								
<input type="checkbox"/> Cluster Redis								
<input type="checkbox"/> Consul Configuration								
<input type="checkbox"/> Hystrix								
<input type="checkbox"/> Stream Kafka								
<input type="checkbox"/> Zipkin								

Navigation: ? < Back Next > Finish Cancel

스프링 시큐리티 레퍼런스와 스프링 부트 레퍼런스를 참조하겠습니다.

<http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#hello-web-security-java-configuration>

<http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-security>

를 참조해주고, 기초적으로 인메모리 로그인 환경을 해보겠습니다.

요구사항 : /admin 으로 들어갈 때는 admin/1234 를 쳐야 로그인이 되게 해보겠습니다.

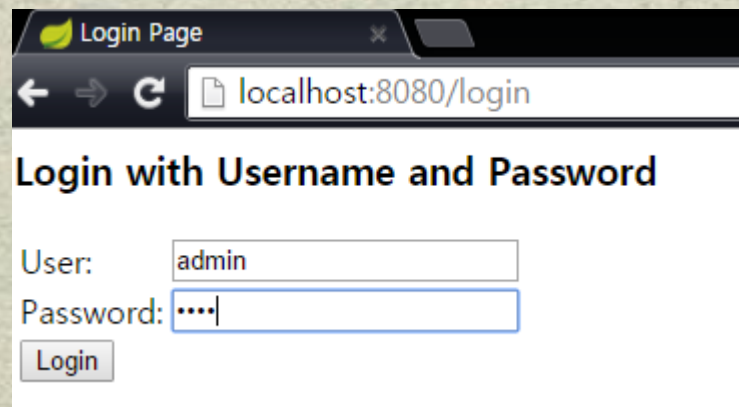
(기초적인 설정은 생략합니다. 스프링 부트에서 컨트롤러로 헬로월드까지 띄웠다는 가정하에 진행합니다.)

윗 링크를 참조해서 클래스를 만들어주고, 다음과 같이 적어주겠습니다.

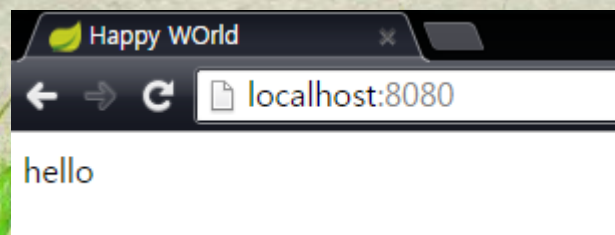
```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    // 01. 인메모리 설정
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
            .withUser("admin").password("1234").roles("ADMIN");
    }
}
```

1.1. 인메모리 로그인 설정

- 더 많은 설정을 하기엔 참을 수가 없습니다.
바로 시작을 해줘보도록 하겠습니다.



바로 스프링 부트를 시작해주면
다음과 같은 화면이 나오게 됩니다.
시큐리티에서 기본으로 제공으로 하는
화면같군요. Admin 과 1234를 쳐보고
로그인해보면!



/ 로 매핑해둔 헬로페이지가 뜨는군요 !

이대로 바로 스프링 부트 실행!

- 저의 추측에 의하면 여기서는 Boot가 자동으로 Configuration을 등록해준 것 같습니다만,
- 레퍼런스의 다음 단락을 보시면 필터를 등록해야 한다는 말이 나옵니다. **springSecurityFilterChain** 을 등록해줘야합니다.
<http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#abstractsecuritywebapplicationinitializer-without-existing-spring>
- 위 링크를 참조해주셔서 필요할 때는 등록해줍니다.
(하단 참고.. 음 클래스 내부 돌아댁기다가.. 제가 아는 그분이 맞는 건지..흠흠;;)
아무튼 다음 단계로 가보겠습니다.

```
*  
* @author Rob Winch  
* @author Keesun Baik  
*/  
public abstract class AbstractSecurityWebApplicationInitializer implements WebApplicationInitializer {  
    private static final String SERVLET_CONTEXT_PREFIX = "org.springframework.web.servlet.FrameworkServlet.  
    public static final String DEFAULT_FILTER_NAME = "springSecurityFilterChain";
```

잠시 생략하는 부분을 설명하며..

- 참고 링크 : <http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#jc-httpsecurity>
- 음 그러면 이제 설정을 조금 더 짚보도록 하겠습니다.
첫 페이지부터 로그인 창이 뜨니 불만스럽습니다.
- **/admin** 에 들어갈 때 해당 로그인 창이 뜨고 싶습니다.
- 설정을 조금 해짚보도록 하겠습니다.
- 레퍼런스 순서와는 약간 순서가 다르지만,
<http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#authorize-requests>
- 다음의 링크를 참조해주고 권한 설정을 해주겠습니다.

1.2. Security 설정

- 레퍼런스와는 조금 다르게 설정해줘보도록 하겠습니다.
- /admin 부분은 ADMIN 권한이 있게 설정해주고,
그 외에 나머지는 모두 허용해보도록 하겠습니다.
그리고 formLogin()을 넣어서 기본로그인 폼이 들어가게
하겠습니다.

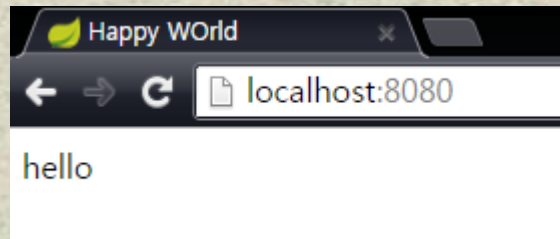
```
// 02. http 설정
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/**").permitAll()
            //.anyRequest().authenticated()
            .and()
            .formLogin();
}
```

```
@RequestMapping("/admin")
public void admin() {}
```

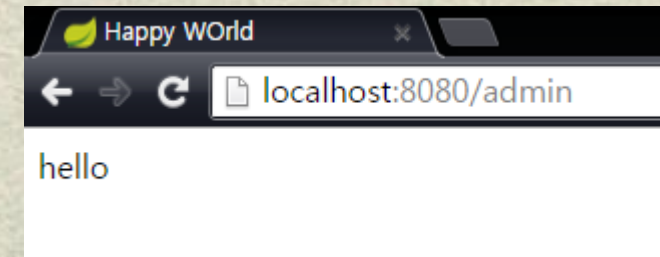
아참, admin.jsp 페이지를 만들어주시고
컨트롤러에도 등록을 해주셔야 합니다^^;

1.2. Security 설정

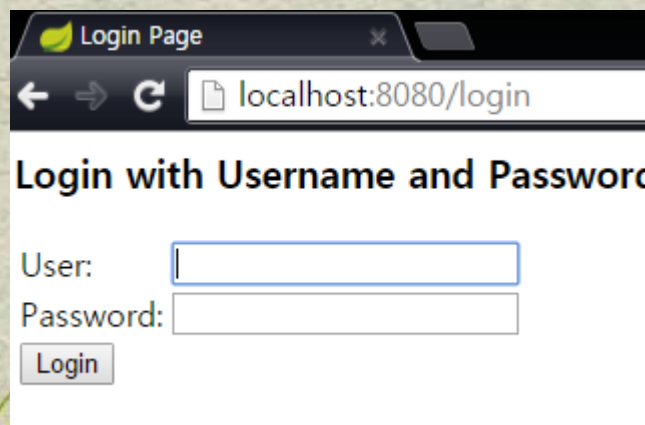
- 다시 그림 시작을 해보도록 하겠습니다.



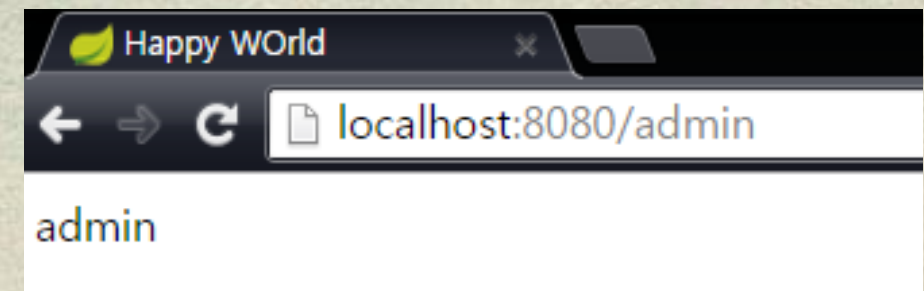
주소입력



로그인 폼 이동



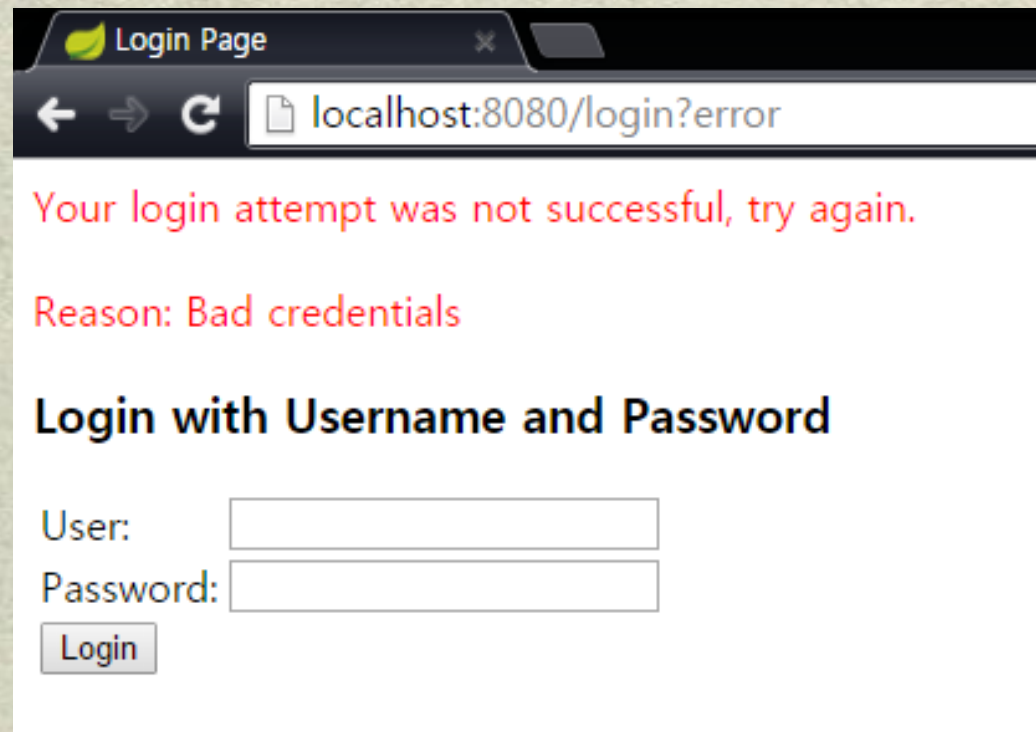
로그인 성공



다시 그림 시작!



- 지금까지 간단히 경로별 설정과 인메모리 설정을 보았습니다. 아직 만족할 수가 없습니다. 이런 기초 화면으로는 서비스를 할 수가 없으니까요..
- 로그인 화면과 로그아웃을 조금 더 커스터마이징 할 필요가 느껴집니다.



- 다음 미션으로 가볼까요...

로그인이 실패할 때는?

2. 로그인 로그아웃 커스터마이징

(깃헙의 inmoryLogin폴더입니다)

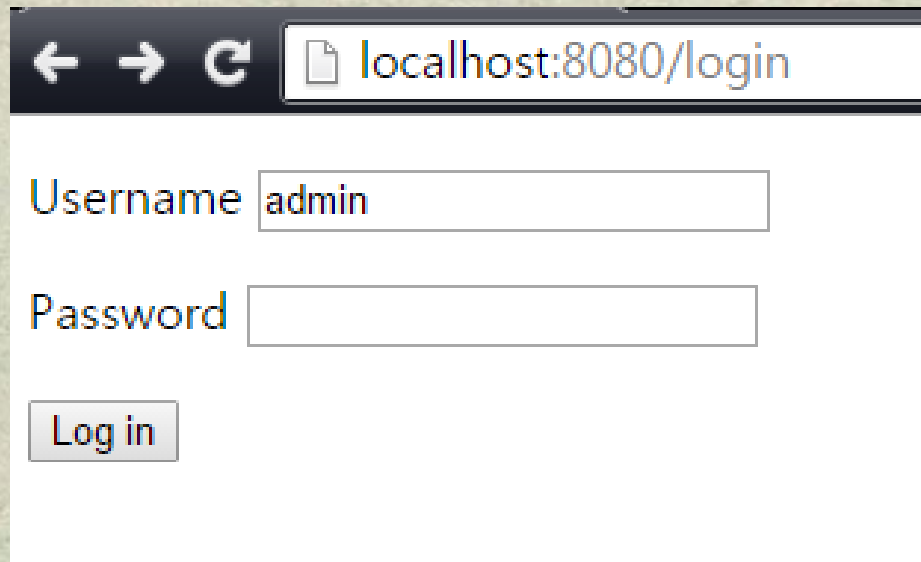
- 자 그러면 이제 로그인, 로그아웃 화면을 조금 더 이쁘게 꾸며보겠습니다!
- <http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#jc-form> 를 참조해주겠습니다.
- 여기에 JSP 샘플 폼도 나오니, 여길 참고주시고 login.jsp 페이지에 (으 만들고 보니 이름이 마음에 안드는;;) 내용을 적어주시고, 시큐리티 설정도 다음과 같이 추가해주겠습니다.

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

```
// 1.2. http 설정  
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
            .antMatchers("/admin/**").hasRole("ADMIN")  
            .antMatchers("/**").permitAll()  
            //.anyRequest().authenticated()  
            .and()  
        .formLogin()  
            .loginPage("/login") // 2.2 로그인 페이지로 이동시킴  
            .permitAll();  
}
```

2.1 로그인 설정

- 다시 시작해보면 다음과 같이 로그인 창이 변경된 것을 보실 수가 있습니다 .

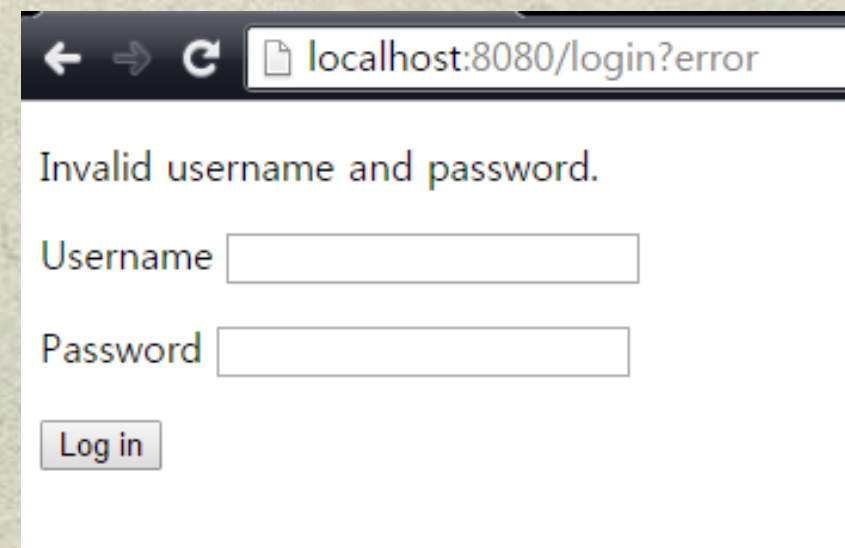


← → ↻ localhost:8080/login

Username

Password

로그인 화면



← → ↻ localhost:8080/login?error

Invalid username and password.

Username

Password

로그인 실패 화면

2.1 로그인 설정 : 재시작



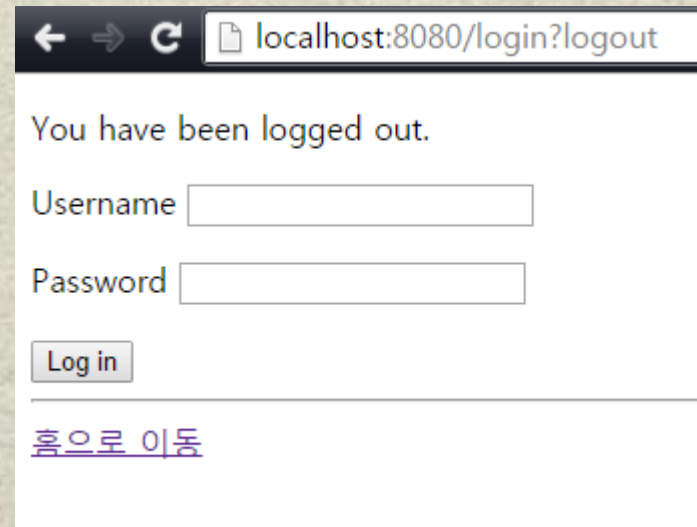
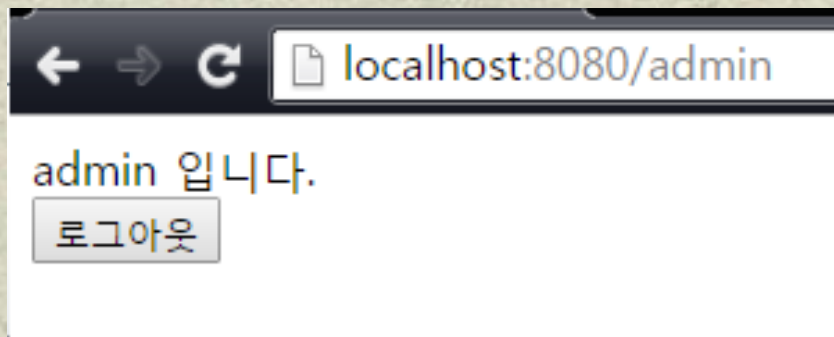
- 예.. 그럼 로그아웃을 한번 해보겠습니다 ‘ㄹ’..
- 시큐리티 설정에 로그아웃을 추가해주겠습니다.
사실 기본적으로 설정이 되어있습니다.
- <http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#jc-logout>
- 음.. Admin.jsp 페이지에 기본값으로 된 주소들을 넣어줘서 로그아웃을 해보겠습니다.
- admin.jsp 페이지에 다음 내용을 넣어주세요.

```
<body>
admin 입니다.

<form action="/logout" method="post">
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
  <input type="submit" value="로그아웃">
</form>
```

2.2 로그아웃

- 다음과 같이 로그아웃 시에 디폴트로 설정된 `login?logout` 주소로 이동하게 됩니다.



여기서 궁금증이 왜 로그아웃 할 때 csrf 를 넣어주는 지 궁금증이 듭니다. 가장 만만하게 위키네요.
다음 위키를 봅시다.

https://ko.wikipedia.org/wiki/사이트_간_요청_위조

2.2 로그아웃 : 화면보기

- 기본적으로 로그아웃 설정 커스터마이징은 다음과 같은 설정을 통해서 가능합니다. (자세한 설명은 생략... ^^;) 참고 : JSP 태그 라이브러리를 쓰면 csrf를 다음과같이 써줘도 됩니다.

```
<sec:csrfInput />
```

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .logout()  
            .logoutUrl("/my/logout")  
            .logoutSuccessUrl("/my/index")  
            .logoutSuccessHandler(logoutSuccessHandler)  
            .invalidateHttpSession(true)  
            .addLogoutHandler(logoutHandler)  
            .deleteCookies(cookieNamesToClear)  
            .and()  
        ...  
}
```

- ① Provides logout support. This is automatically applied when using `WebSecurityConfigurerAdapter`.
- ② The URL that triggers log out to occur (default is `/logout`). If CSRF protection is enabled (default), then the request must also be a POST. For for information, please consult the [JavaDoc](#).
- ③ The URL to redirect to after logout has occurred. The default is `/login?logout`. For for information, please consult the [JavaDoc](#).
- ④ Let's you specify a custom `LogoutSuccessHandler`. If this is specified, `logoutSuccessUrl()` is ignored. For for information, please consult the [JavaDoc](#).
- ⑤ Specify whether to invalidate the `HttpSession` at the time of logout. This is `true` by default. Configures the `SecurityContextLogoutHandler` under the covers. For for information, please consult the [JavaDoc](#).
- ⑥ Adds a `LogoutHandler`. `SecurityContextLogoutHandler` is added as the last `LogoutHandler` by default.
- ⑦ Allows specifying the names of cookies to be removed on logout success. This is a shortcut for adding a `CookieClearingLogoutHandler` explicitly.

2.2 로그아웃 참고로 넣는 레퍼런스

우선 여기까지 진행해보고, 언제까지 인메모리로
로그인과 로그아웃을 할 수는 없으니, 이제
데이터베이스로 로그인/로그아웃을 해보겠습니다.

3. 데이터 베이스에 사용자 정보 저장하기

(여기서부터는 깃헙-jdbcLogin 폴더)

- 언제까지 메모리에서 시큐리티 정보를 사용할 수는 없습니다. 이제 기본적인 로그인 로그아웃 처리를 데이터베이스를 통해 해보겠습니다.
- 먼저 가장 기본적으로 어떤 형태로 일어나는 지 보도록 하겠습니다. 먼저 Datasource가 필요합니다.
- Boot의 application.properties 파일을 다음과 같이 수정해주겠습니다.

```
4|
5 spring.datasource.url=jdbc:mysql://localhost/learn_security
6 spring.datasource.username=arahansa
7 spring.datasource.password=
8 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
9
```

3.1 JDBC 로그인

- <http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#user-schema>
- 를 참고해줘서 데이터베이스 테이블을 만들어주겠습니다.

```

1 create table users(
2     username varchar(50) not null primary key,
3     password varchar(50) not null,
4     enabled boolean not null
5 );
6
7 create table authorities (
8     username varchar(50) not null,
9     authority varchar(50) not null,
10    constraint fk_authorities_users foreign key(username) references users(username)
11 );
12 create unique index ix_auth_username on authorities (username,authority);

```

- 테이블을 정의해준 뒤에 값을 넣어주겠습니다.

learn_security.authorities: 1 행 (총) (대략적)

username	authority
admin	ROLE_ADMIN

learn_security.users: 1 행 (총) (대략적)

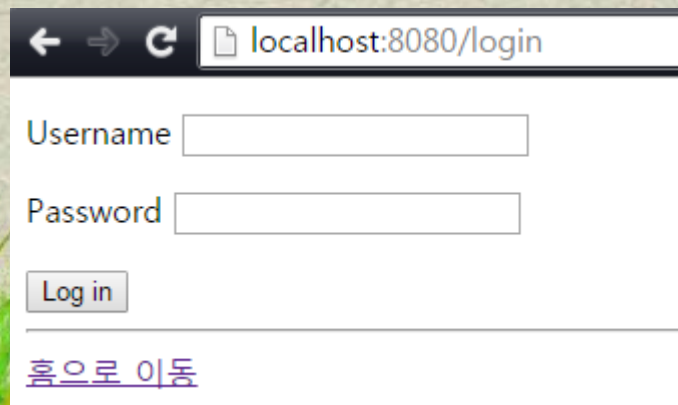
username	password	enabled
admin	1234	1


```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired DataSource datsSource;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication().dataSource(datsSource);
    }
}
```

기본 jdbcAuthentication을 이용해서
설정해보겠습니다. 잘 동작함을 알
수가 있습니다.



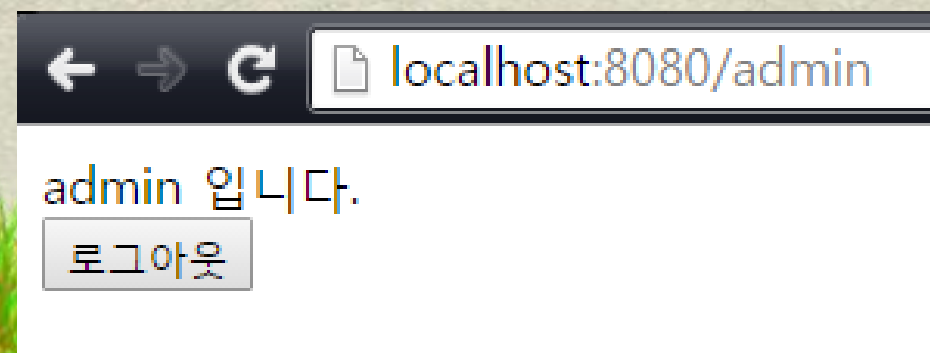
← → ↻ localhost:8080/login

Username

Password

Log in

[홈으로 이동](#)



← → ↻ localhost:8080/admin

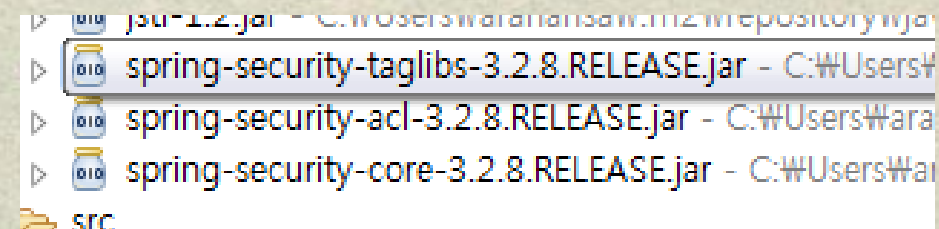
admin 입니다.

로그아웃

3.1 JDBC 로그인 설정 후 실행

- JDBC로 로그인만 보고 넘어가기에 이번 단락은 너무 심심하네요.. Jsp 페이지에서 시큐리티 정보를 얻는 방법을 보겠습니다.
- Pom.xml 에 다음의 태그 라이브러리를 넣어주겠습니다. (우측확인)

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
</dependency>
```



- 그리고 JSP페이지에 다음의 태그 라이브러리를 넣어주겠습니다.

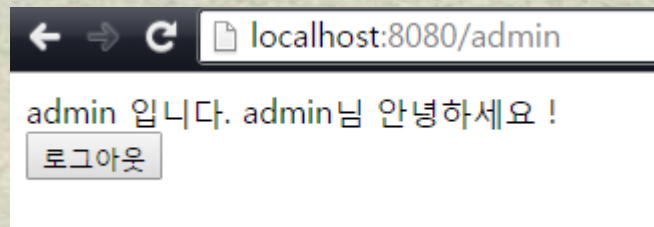
```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
```

- 그리고 다음의 인사말을 넣어주도록 할까요?! (여기서 principal이란 것을 당장 설명하지 않습니다. 지금은 흐름을 적고, 어떤 클래스가 있다는 지의 개념은 다음 장에서 좀 더 필요할 때 적겠습니다.)

```
<sec:authentication property="principal.username"/>님 안녕하세요 !
```

3.2 JSP에 기타 정보 더하기

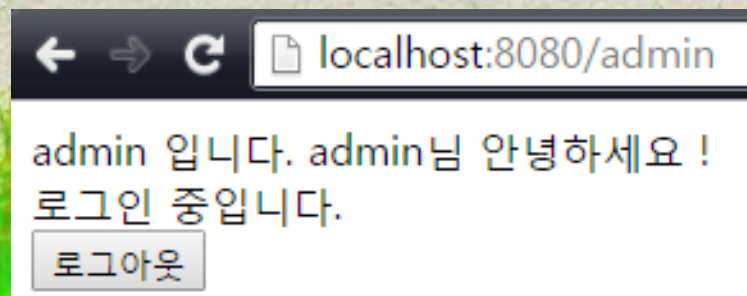
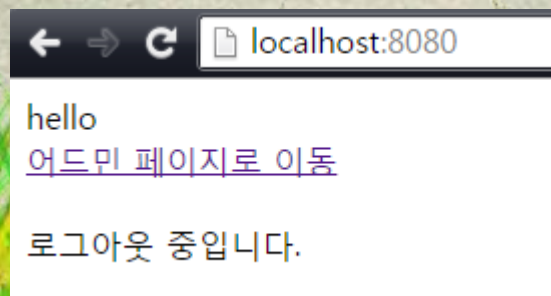
- 그리고 이제 실행해보겠습니다!
다음과 같은 메시지를 보실 수가 있을 것입니다.



- 나중에는 메뉴에 - 네비게이션도 넣어서, 로그인/로그아웃 상태별로 다른 상황을 넣어줘야 합니다.

```
<sec:authorize access="isAnonymous()">
    <br> 로그아웃 중입니다.
</sec:authorize>
<sec:authorize access="isAuthenticated()">
    <br> 로그인 중입니다.
</sec:authorize>
```

- 로그인여부에 따라 다르게 처리해주는 다음의 코드를 admin.jsp 와 index.jsp 에 넣어주면 다음과 같이 나옵니다.



3.2 JSP에 기타 정보 더하기

4. 좀 더 커스터마이징..

(여기서부터는 깃헙-customizingLogin 폴더)

- customLogin 프로젝트에서는 데이터에 좀 더 값을 넣어줬고, user 권한으로 접근할 수 있는 경로도 좀 넣어줬습니다. 그냥 좀 재미로요~ㅎ..지금까지 한 내용으로 충분히 할 수 있는 내용이니 넘어갑니다.

learn_security.users: 2 행 (총) (대략적)

username	password	enabled
admin	1234	1
arahansa	1234	1

learn_security.authorities: 2 행 (총) (대략적)

username	authority
admin	ROLE_ADMIN
arahansa	ROLE_USER

```
@RequestMapping("/user")  
public void user() {}
```

hello
[어드민 페이지로 이동](#)
[유저 페이지로 이동](#)
로그인 중입니다.

// 1.2. http 설정

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
            .antMatchers("/admin/**").hasRole("ADMIN")  
            .antMatchers("/user/**").hasAnyRole("USER", "ADMIN")  
            .antMatchers("/**").permitAll()  
}
```

그 전에 잠시.. 재미로..

- 여기까지 했는데, 또 불만 사항이 있습니다.. 만약 jdbcLogin 말고 다른 처리를 하고 싶다면, username 말고도 principal 에 다른 정보를 넣어서 로그인 시마다 가지고 다니게 하고 싶을 때는 어떻게 할까요.. 이럴 때 어떻게 할지 한번 생각해봅시다.
- 음.. 다른 예제도 있긴 하지만, 백기선님의 아무고나 방송시리즈를 조금 참조하겠습니다. 여기서는 spring data jpa를 이용해서 데이터베이스 조회를 하고 권한설정을 하도록 하겠습니다.

```
@Entity
public class Account {
    @Id
    @GeneratedValue
    private Long id;

    private String userid;

    private String password;

    // 이 부분은 나중에 enum 과 일대다로 빼든 지 하는 작업이 필요
    // 시큐리티 튜토리얼이므로 간략하게 적습니다.
    private String role;

    private String nick;
```


- 아참 application.properties 에는 다음의 jpa설정을 추가를 좀 해줘야겠네요

```
10 #jpa
11 spring.jpa.database=mysql
12 spring.jpa.hibernate.ddl-auto=update
13 spring.jpa.show-sql=true
```

- 여기서부터는 시큐리티의 어떤 특정 클래스들이 나오게 되는데, 먼저 코딩부터 하고 돌려보고, 어떤 클래스가 있는지 보겠습니다.
security.core.userdetails.User를 상속하는 다음 클래스를 만들어주겠습니다.

```
// 기선님의 아무고나 소스 참조함.
public class UserDetailsImpl extends User {

    public UserDetailsImpl(Account account) {
        super(account.getUserid(), account.getPassword(), authorities(account));
    }

    private static Collection<? extends GrantedAuthority> authorities(Account account) {
        List<GrantedAuthority> authorities = new ArrayList<>();
        // 원래 일대다, 이놈을 하는 식으로 해서 처리를 해야 할 것같지만
        // 그 부분의 매핑은 책을 사서 보시기를 바랍니다.
        // 절대 제가 귀찮아서ㅋㅋㅋ이러는게 아닙니다.
        // 최근에 배운것을 막 바로 적어 블로그하기도 뭐해서요 ㅎㅎ
        authorities.add(new SimpleGrantedAuthority(account.getRole()));
        return authorities;
    }
}
```

4.1 커스터마이징 로그인 - 준비작업

- Repository 영역에는 다음의 JpaRepository를 만들어주겠습니다.

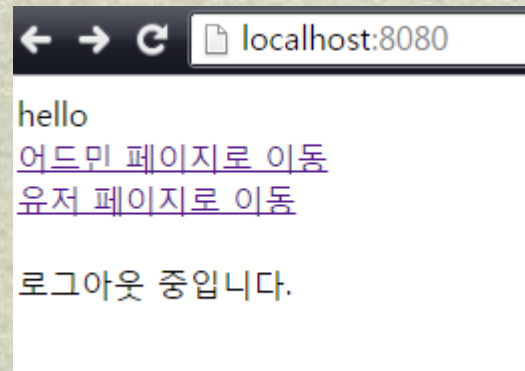
```
public interface AccountRepository extends JpaRepository<Account, Long>{  
    public Account findByUserId(String userid);  
}
```

- 그리고 security.core.userdetails.UserDetailsService를 구현하는 다음 클래스를 만들어주겠습니다.

```
@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
    @Autowired  
    AccountRepository accountRepository;  
  
    @Override  
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
        Account account = accountRepository.findByUsername(username);  
        if (account == null) {  
            throw new UsernameNotFoundException(username);  
        }  
        return new UserDetailsImpl(account);  
    }  
}
```


- 자 그럼, 프로그램을 다시 시작해보도록 하겠습니다. 여기서부터는 기존의 JDBC를 쓰는 것이 아니라 JPA가 생성한 테이블을 쓰므로, 테이블 생성을 확인한 뒤에.. 회원가입하는 부분을 안 만들어줬으니, 직접 테이블에 데이터를 넣어주고 로그인 해보겠습니다.

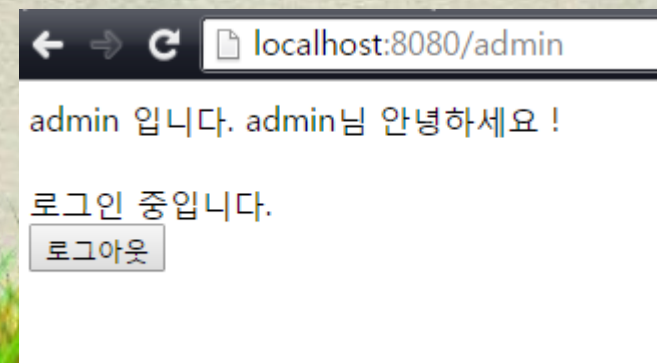
- 페이지로 이동해서..



- 다음과 같이 데이터를 넣어주고 로그인하면 짜잔!.

learn_security.account: 1 행 (총) (대략적)

id	nick	password	role	userid
1	arahansa	1234	ROLE_ADMIN	admin



- 자 여기서 몇가지 스프링 시큐리티에서 제공하는 **User** 라던가, **UserDetailsService** 라는 것들을 상속 혹은 구현해 썼는지 이제 좀 알아보도록 하겠습니다.
- (미리 말씀드리자면, 전 항상 이론과 설명에 약합니다 ㅠ 그냥 API랑 레퍼런스를 보고 따라치는 것을 즐겨서요. 흠..그래도 적어보겠습니다. ㅠ)
- 먼저 스프링 시큐리티에서는 **Principal** 이라는 개념을 알아야 합니다. 구글에서 what is principal in spring security 를 검색해보면 스프링 시큐리티 API에 Authentication(인증) 인터페이스가 나오고 이것이 Principal 을 상속한다고 나옵니다

4.1 커스터마이징 로그인 = 설명

- 이 Authentication은 Principal을 상속하며 인증된 요청혹은 증명된 principal을 위한 토큰을 나타냅니다. 인증이 되면 쓰레드 로컬의 SecurityContext 에 Authentication은 저장됩니다.

```
public interface Authentication  
extends Principal, Serializable
```

Represents the token for an authentication request or for an authenticated principal once the request has been processed by the AuthenticationManager.authenticate() method.

Once the request has been authenticated, the Authentication will usually be stored in a thread-local SecurityContext managed by the SecurityContextHolder by the Authentication instance and using the code:

```
SecurityContextHolder.getContext().setAuthentication(anAuthentication);
```

- 자바문서에서는 Principal은 로그인아이디같은 엔티티를 나타내는 데 사용될 수 있는 추상화된 개념입니다.

```
public interface Principal
```

This interface represents the abstract notion of a principal, which can be used to represent any entity, such as an individual, a corporation, and a login id.

- 아...저는 아무튼 결론은 로그인한 정보를 principal이라는 이름으로 들고있다고 생각하고 있습니다-_-a
- 이 principal에 들어갈 수 있는 클래스로는 **UserDeatils** 인터페이스를 구현하거나 **User**를 상속해야 합니다. (User는 Userdetails를 implement 합니다)
- 여기서는 간단하게 User를 상속해서 super를 통해 아이디, 패스워드, 권한을 설정하였습니다.. 더 이상의 자세한 설명은 백기선님의 아무거나 방송에 시큐리티 편을 참고하겠습니다. 절대 제가 귀찮아서 그런게 아닙니다. ㄸ

```
public UserDetailsImpl(Account account) {  
    super(account.getUserid(), account.getPassword(), authorities(account));  
}
```



4.1 커스터마이징 로그인 = 설명

- 또 로그인을 사용하게 하려면 `security.core.userdetails.UserDetailsService` 의 구현체를 만들어서 사용자를 찾는 `loadUserByUsername` 메서드를 구현해야 합니다. 반환값은 `UserDetails` 인터페이스를 구현체를 받습니다.
- 이렇게 알맞은 `UserDetails` 구현체를 찾아서 돌려주면 로그인은 알아서 시켜줍니다. 역시 더 이상의 자세한 설명은 생략합니다;;

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    AccountRepository accountRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Account account = accountRepository.findByUsername(username);
        if (account == null) {
            throw new UsernameNotFoundException(username);
        }
        return new UserDetailsImpl(account);
    }
}
```



- 자 그러면, 여기서 principal의 username 뿐만 아니라 principal의 닉네임도 넣고 싶다면.. 어떻게 할까요?! 간단합니다. UserDetailsImpl에 필드 좀 더 넣어주도록 하겠습니다. 다음과 같이 넣어주시고 로그인 해보시면! 다음과 같이 잘 뜨겠군요 !

```
// 기선님의 아무고나 소스 참조함.  
public class UserDetailsImpl extends User {  
  
    private String nick;  
    public String getNick() { return nick;}  
    public void setNick(String nick) {this.nick = nick; }  
  
    public UserDetailsImpl(Account account) {  
        super(account.getUserid(), account.getPassword(), auth  
        this.nick = account.getNick();  
    }  
}
```

```
<sec:authentication property="principal.username"/>님 안녕하세요 !<br>  
<sec:authentication property="principal.nick"/>님 안녕하세요 !<br>
```

admin 입니다. admin님 안녕하세요 !
arahansa님 안녕하세요 !

로그인 중입니다.

로그아웃

4.1 커스터마이징 - 추가 정보

- 자 그럼, 여기서 질문이.. 로그인은 `loadByUsername`으로 한다고 치는데..
- 회원가입은 어떻게 할까요? 회원가입하고 나서.. 또 로그인 하라고 할까요? 보통 자동으로 로그인이 되는 걸로 알고 있습니다.
- 그러면, 이쯤에서 로그인도 하면서 회원가입도 해보도록 하겠습니다.

4.1 커스터마이징 회원가입?

- Index.jsp 에는 회원가입 페이지로 이동하는 링크를 넣어주시고

```
<a href="/registerForm">회원가입 페이지로 이동</a><br>
```

- 컨트롤러에 매핑도 만들어주시고~

```
@RequestMapping("/registerForm")  
public void registerForm() {}
```

- 그러면 이름에 맞게 registerForm도 만들어줍니다.

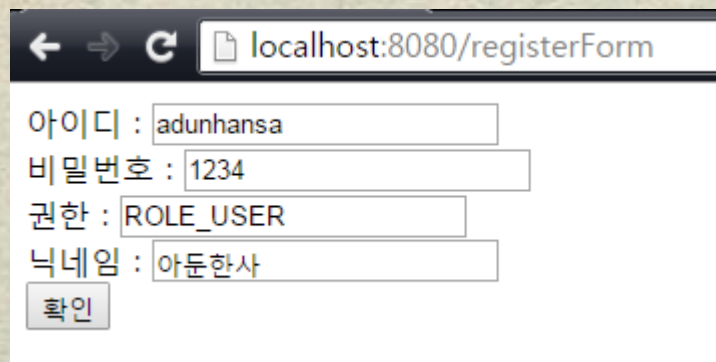
```
8  
9 <form method="post" action="/register">  
10 아이디 : <input type="text" name="userid" /><br>  
11 <!-- Toy 이라 비밀번호도 그냥 확인하고 싶어서 text로 걸었습니다 -->  
12 비밀번호 : <input type="text" name="password" /><br>  
13 권한 : <input type="text" name="role" value="ROLE_USER" /><br>  
14 닉네임 : <input type="text" name="nick" value="" /><br>  
15 <input type="submit" value="확인">  
16 <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>  
17 </form>  
18
```


- 그 이후는 간단합니다. POST를 받는 컨트롤러를 작성해주고, 이전의 API설명에서 나온대로 SecurityContext 에 Authentication을 직접 set해주겠습니다.

```
// 필요한 부분은 Service계층으로 옮겨줘야 합니다.  
@Autowired AccountRepository accountRepository;  
  
@RequestMapping(value="/register", method=RequestMethod.POST)  
public String register(Account account){  
    // 회원정보 데이터베이스에 저장  
    accountRepository.save(account);  
  
    // SecurityContextHolder에서 Context를 받아 인증 설정  
    UserDetailsImpl userDetails = new UserDetailsImpl(account);  
    Authentication authentication =  
        new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());  
    SecurityContextHolder.getContext().setAuthentication(authentication);  
    return "redirect:/";  
}
```

- 아 이부분은 구글에서 How to set manually Login 을 쳐도 나오긴 하지만 <http://stackoverflow.com/questions/4664893/how-to-manually-set-an-authenticated-user-in-spring-security-springmvc> 예전에 봤던 소스중에 더 간단하게 한 것같은 부분이 있어서 저 부분으로 처리를 해봤습니다.

- 이제 회원가입을 해볼까요? 다음과 같이 회원가입 후에 로그인이 바로 됩니다 ^.^;



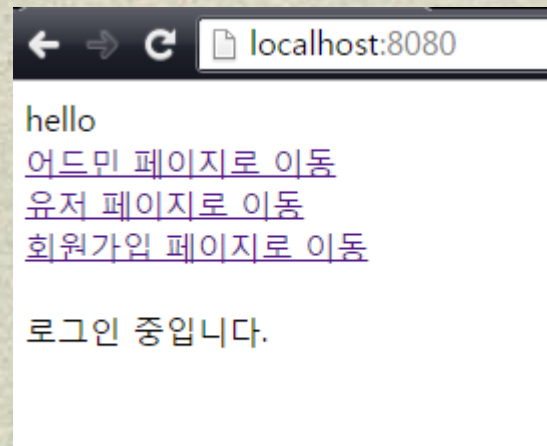
localhost:8080/registerForm

아이디 :

비밀번호 :

권한 :

닉네임 :



localhost:8080

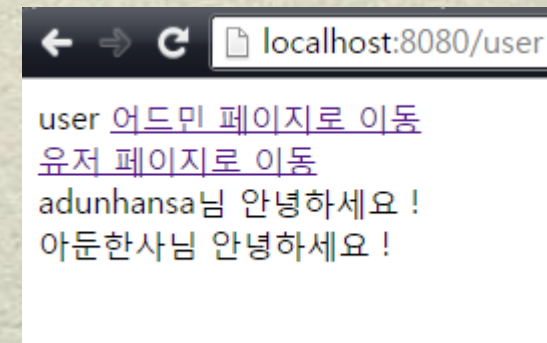
hello

[어드민 페이지로 이동](#)

[유저 페이지로 이동](#)

[회원가입 페이지로 이동](#)

로그인 중입니다.



localhost:8080/user

user [어드민 페이지로 이동](#)

[유저 페이지로 이동](#)

adunhansa님 안녕하세요 !

아둔한사님 안녕하세요 !

- 자 이제 여기까지 해봤으니, 뭔가 앞으로 잘 할 수 있을 것같은 기분이 들지만, 여기서 만족할 수가 없습니다. 요즘에는 아이디 저장같은 기능이 있어야합니다. (아이디, 비밀번호 저장 클릭도 귀찮습니다...;;)

- 여기서부터는 인텔리제이로 하겠습니다.

5.Rembmer Me..

(여기서부터는 깃헙-rememberMeLogin 폴더)



5. Remember ME를 구현해보자. < 46 >

- Remember Me는 토큰기반의 쿠키저장방식과 Persistence 한 방법 두가지가 있다고 합니다.

15. Remember-Me Authentication

15.1. Overview

15.2. Simple Hash-Based Token Approach

15.3. Persistent Token Approach

15.4. Remember-Me Interfaces and Implementations

15.4.1. TokenBasedRememberMeServices

15.4.2. PersistentTokenBasedRememberMeServices

- 두가지 모두를 해보도록 하겠습니다. 우선 간단한 Remember-ME부터 해보도록 할까요 SecurityConfig 파일의 http 변수 에는 다음과 같이 remember Me 설정을, login.jsp 에는 다음과 같이 체크박스를 넣어주겠습니다.

```
http.rememberMe().key("arahansaKey");
```

```
<p>  
Remember Me : <input type="checkbox" name="_spring_security_remember_me" value="true"/>  
</p>
```

5. 나를 기억해 주겠니..Remember Me..

- 로그인이 되었습니다. 이제 크롬창을 닫고 다시 같은 페이지에 접근해보겠습니다.

Username

Password

Remember Me : ☒

[홈으로 이동](#)

user [어드민 페이지로 이동](#)

[유저 페이지로 이동](#)

arahansa님 안녕하세요 !

아라한사님 안녕하세요 !

- 크롬창을 닫고 다시 접근해보면? 다시.. 로그인중이라는 메시지가 계속 뜨게 됩니다 .

hello
[어드민 페이지로 이동](#)
[유저 페이지로 이동](#)
[회원가입 페이지로 이동](#)
로그인 중입니다.

- 쿠키방식과 데이터베이스 방식을 좀 더 알아보겠습니다.

```
public class TokenBasedRememberMeServices  
extends AbstractRememberMeServices
```

Identifies previously remembered users by a Base-64 encoded cookie.

This implementation does not rely on an external database, so is attractive for simple applications.

- 구글링해보니 쿠키방식에는 TokenBasedRememberMeServices가 있습니다. Base-64 인코드된 쿠키 방식으로 외부 DB에 의존하지 않습니다.

```
username + ":" + expiryTime + ":"  
+ Md5Hex(username + ":" + expiryTime + ":" + password + ":" + key)
```


- 자, 그럼 API 생성자에 맞춰서 토큰을 넣어주도록 하겠습니다. 그러면 다음과 같이 로그인이 쿠키토큰 방식으로 놓여지게 됩니다. 크롬에서 F12를 누르시고 Resources 탭을 보시면 다음과 같이 정해진 이름대로 쿠키가 나오는 것을 보실 수가 있습니다.

Constructor Summary

Constructors

Constructor and Description

TokenBasedRememberMeServices(String key, UserDetailsService userDetailsService)

user [어드민 페이지로 이동](#)
[유저 페이지로 이동](#)

 arahansa님 안녕하세요!
 아라한사님 안녕하세요!

```
http.rememberMe().key("arahansaKey").rememberMeServices(tokenBasedRememberMeServices());
}
```

```
@Bean
public TokenBasedRememberMeServices tokenBasedRememberMeServices(){
    TokenBasedRememberMeServices tokenBasedRememberMeServices =
        new TokenBasedRememberMeServices("arahansaKey", userDetailsService);
    tokenBasedRememberMeServices.setCookieName("arahansaCookie");
    return tokenBasedRememberMeServices;
}
```

Name	Value
JSESSIONID	529E5653E608CC83EB39
arahansaCookie	YXJhaGFuc2E6MTQ0ND

- 음.. 더 상세한 내용이 토큰기반 RememberMe는 API를 보시도록 하겠습니다. 기본적으로는 14일 동안 유효한 것 같고, 음.. API를 보니 좀 더 안전한 접근을 원하면, 데이터베이스 기반 구현체를 추천한다고 하는군요..

This is a basic remember-me implementation which is suitable for many applications. However, we recommend a database-based implementation if you require a more secure remember-me approach (see PersistentTokenBasedRememberMeServices).

By default the tokens will be valid for 14 days from the last successful authentication attempt. This can be changed using AbstractRememberMeServices.setTokenValiditySeconds(int). If this value is less than zero, the expiryTime will remain at 14 days, but the negative value will be used for the maxAge property of the cookie, meaning that it will not be stored when the browser is closed.

- API 주소 <https://docs.spring.io/spring-security/site/docs/current/apidocs/org/springframework/security/web/authentication/rememberme/TokenBasedRememberMeServices.html>

- 이번엔 Persistent 기반의 RememberMe를 구현해보겠습니다. PersistentTokenBasedRememberMeService(헥헥)의 일반 생성자는 Deprecated되었고 생성자로 세 개의 파라미터를 필요로 합니다. Key, userDetailsService, PersistentTokenRepository 이며, 여기서 PersistentTokenRepository 인터페이스는 두가지의 구현클래스를 제공하는데, JDBCToken~을 써주겠습니다. 다음과 같이 적어봅시다.

```
http.rememberMe().key("arahansaKey");
http.rememberMe().key(REMEMBER_ME_KEY).rememberMeServices(persistentTokenBasedRememberMeServices());
}

@Bean
public PersistentTokenBasedRememberMeServices persistentTokenBasedRememberMeServices(){
    PersistentTokenBasedRememberMeServices persistentTokenBasedRememberMeServices
        = new PersistentTokenBasedRememberMeServices(REMEMBER_ME_KEY, userDetailsService, jdbcTokenRepository());
    return persistentTokenBasedRememberMeServices;
}

@Bean
public JdbcTokenRepositoryImpl jdbcTokenRepository(){
    JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
    jdbcTokenRepository.setCreateTableOnStartup(false);
    jdbcTokenRepository.setDataSource(dataSource);
    return jdbcTokenRepository;
}
```



- 아참 그 전에 테이블을 다음과 같이 만들어주시고, 로그인을 해주고 데이터베이스를 확인해주면 다음과 같이 데이터가 들어온 것을 볼 수가 있습니다.

The database should contain a `persistent_logins` table, created using the following SQL (or equivalent):

```
create table persistent_logins (username varchar(64) not null,
                                series varchar(64) primary key,
                                token varchar(64) not null,
                                last_used timestamp not null)
```

user [어드민 페이지로 이동](#)
 유저 [페이지로 이동](#)
[로그아웃](#)
 arahansa님 안녕하세요 !
 아라한사님 안녕하세요 !

learn_security.persistent_logins: 1 행 (총) (대략적)

username	 series	token	last_used
arahansa	KZnJr8tBuhDsMJbxbciBnw==	2AQXvwfqUXf34dd7B7h6nQ==	2015-09-28 21:44:54

5. Remember Me - Persistent based

6. 기타

@PreAuthorize

패스워드 암호화

403페이지(권한없음) 커스터마이징
시큐리티 4버전에 추가된 테스트해보기

(여기서부터는 깃헙-etcSecurity 폴더)

- @PreAuthorize는 레퍼런스의 23장에 해당하는 부분입니다.

23. Expression-Based Access Control

23.1. Overview

23.1.1. Common Built-In Expressions

23.2. Web Security Expressions

23.3. Method Security Expressions

23.3.1. @Pre and @Post Annotations

Access Control using @PreAuthorize and @PostAuthorize

Filtering using @PreFilter and @PostFilter

23.3.2. Built-In Expressions

The PermissionEvaluator interface

- 사실 이 부분에 대해서는 <http://whiteship.me/?tag=스프링-시큐리티-3-0> 에 잘 나타나 있습니다.

6.1. @PreAuthorize

- @PreAuthorize 를 해주기 위해선 설정파일에서 이 옵션을 켜주셔야 합니다.

```
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

- 그리고 이 부분을 컨트롤러에 추가해보겠습니다. 설정은 보시면 이해가 가실 것입니다. 설정에서 /admin/** 뭐 이런 것들 설정잡아주던 것을 메소드레벨에서도 설정을 하였습니다. 첫번째 메소드 같은 경우는 파라미터로 오는 account가 로그인된 username과 같거나 관리자인 경우만 볼 수 있게 하였습니다.

```
@RequestMapping("/getPrivateMessage")
@PreAuthorize("(#account.userid == principal.Username) or hasRole('ROLE_ADMIN')")
public String authstring(Account account, Model model) {
    model.addAttribute("msg", "당신은 관리자거나 요청 파라미터랑 아이디가 같습니다?");
    return "authorizedMessage";
}

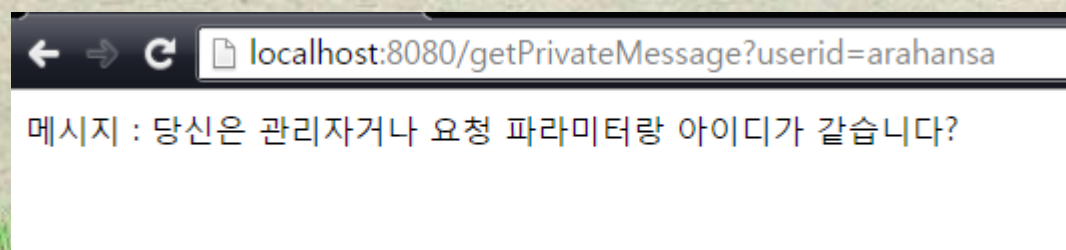
@RequestMapping("/getUserMessage")
@PreAuthorize("hasRole('ROLE_USER')")
public String userMesasge(Account account, Model model){
    model.addAttribute("msg", "당신은 한낱 유저입니다. ㅠ");
    return "authorizedMessage";
}
```

6.1. @PreAuthorize

- 일반유저로 로그인하고 접근하니 안된다고 나옵니다.



- 하지만 userid를 현재 로그인 사용자로 주니, 다음과 같이 권한이 있어서 페이지를 볼 수가 있습니다.



6.1. @PreAuthorize 실행

- 지금까지 그냥 일반 패스워드를 썼지만 패스워드는 안전한 방식으로 암호화되어야 합니다. ‘ㄹ’ 최근에..뽕X사이트에서 개인정보가 털려서-_-.. (저는 이미 포기..)흠흠..
- 아무튼 <http://d2.naver.com/helloworld/318732> 네이버의 암호화 문서를 한번 봐주고~ 여기서는 Bcrypt 로 암호화를 해주겠습니다.
- 패스워드 인코더를 설정해야 합니다. 시큐리티 설정 클래스에 다음의 빈을 설정해주겠습니다.

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

6.2 패스워드 암호화

- 그리고 auth 설정에도 해줘야겠군요.. 계속 설정파일입니다.

```
@Autowired PasswordEncoder passwordEncoder;

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
}
```

- 그리고 스프링 부트 실행파일에 더미데이터를 넣어주는 부분을 initializingBean으로 실행시키는 부분이 있는데 그 부분도 이제 암호화를 해주겠습니다. 아 회원가입받을 때도 해주겠습니다.

```
@Autowired
PasswordEncoder passwordEncoder;

@Bean
InitializingBean insertFixtureUsers(){
    return ()->{
        Account admin = new Account();
        admin.setUserId("admin");
        admin.setPassword(passwordEncoder.encode("1234"));
        admin.setRole("ROLE_ADMIN");
    };
}
```

```
@RequestMapping(value="/register", method=RequestMethod.POST)
public String register(Account account){
    // 회원정보 데이터베이스에 저장
    account.setPassword(passwordEncoder.encode(account.getPassword()));
    accountRepository.save(account);
}
```

6.2 패스워드 암호화

- 패스워드 암호화 후 실행해보면 다음과 같이 암호화된 비밀번호가 들어감을 볼 수가 있습니다.

learn_security.account: 2 행 (총) (대략적)

id	nick	password	role	userid
1	관리자한사	\$2a\$10\$SL.8oO.vqMiHxwipadiue4QQH1YTf7...	ROLE_ADMIN	admin
2	마라한사	\$2a\$10\$8d.FyPQWqFfqfSOdRzVMketS4AFpLp...	ROLE_USER	arahansa

- 실행을 해보니.. 회원가입을 새로 하건, 로그인을 하건, 뭐 잘 되는군요! ‘ㄹ’..

user [어드민 페이지로 이동](#)
 유저 [페이지로 이동](#)

로그아웃

adunhanasa님 안녕하세요!
 아둔한사님 안녕하세요!

6.2 패스워드 암호화

- 지금까지 접근이 제한된 페이지를 보시면 다음과 같은 에러를 종종 보셨을 것입니다. 이번엔 403페이지를 커스터마이징해서 보여드리겠습니다.

Whitelabel Error Page

This application has no explicit mapping for /error, so y

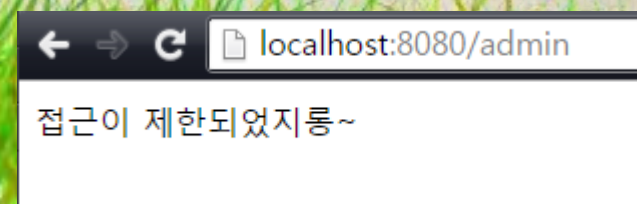
Tue Sep 29 07:37:03 KST 2015

There was an unexpected error (type=Forbidden, status
Access is denied

- 사실 간단합니다. 설정파일의 http에서 403을 지정해주고 컨트롤러랑 jsp페이지 만들어주면 되겠습니다~ 유저권한으로 어드민 페이지 접근하면 다음과 같이 나옵니다~

```
// 403 페이지 핸들링  
http.exceptionHandling().accessDeniedPage("/403");
```

```
@RequestMapping("/403")  
public void accessDeniedPage() {}
```



6.3. 403 페이지 만들기

- 좀 더 해봅시다. 스프링 시큐리티 4.0에서 새로운 부분이 있다고 하네요!~

2. What's new in Spring Security 4.0

There are 175+ tickets resolved with the Spring Security 4.0 release.

2.1 Features

Below are the highlights of the new features found in Spring Security 4.0.

- Web Socket Support
- Test Support
- Spring Data Integration
- CSRF Token Argument Resolver
- More Secure Defaults
- Methods with role in them do not require ROLE_ For example, previously the

```
<intercept-url pattern="/" access="hasRole('ROLE_USER')"/>
```

- 백기선님의 아무고나방송에서 나온대로 부트에서 스프링 버전과 시큐리티 버전을 좀 더 올려주겠습니다~

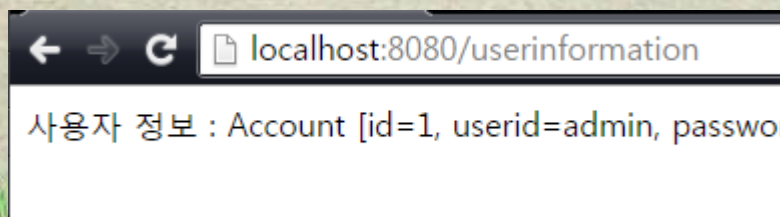
```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <java.version>1.8</java.version>
  <spring.version>4.2.1.RELEASE</spring.version>
  <spring-security.version>4.0.2.RELEASE</spring-security.version>
</properties>
```


- **앗참, 그리고 이 의존성도 필요합니다..;ㅁ;.**

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-data</artifactId>
</dependency>
```

- **다음과 같이 principal을 통해서 검색이 가능해집니다. 상당히 편해보일 것같군요... 얼핏 기억나기에 시큐리티 콘텍스트에서 정보를 얻어서 검색을 했었던 것같기도..**

```
@Query("select a from Account a where a.userid = ?#{ principal.Username }")
Account findMe();
```



- 앞으로는 **ROLE_** 을 생략을 해도 된다고 합니다.

Now you can optionally omit the `ROLE_` prefix. We do this to remove duplication. Specifically, since the automatically adds the prefix if it is not there. For example, the following is the same as the previous code:

```
<intercept-url pattern="/**" access="hasRole('USER')"/>
```

Similarly, the following configuration:

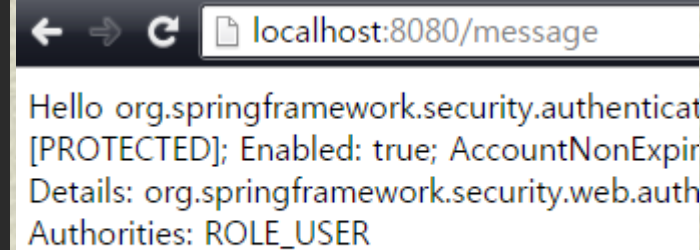
```
@PreAuthorize("hasRole('ROLE_USER')")
```

is the same as this more concise configuration:

```
@PreAuthorize("hasRole('USER')")
```


- 스프링 시큐리티 레퍼런스에 다음과 같이 인증이 된 후에 정보를 가져오는 서비스 메서드가 있습니다. 이 부분을 테스트해보겠습니다. 아참 spring-security-test로 불러와줍시다. 메시지를 받아오는 부분은 모두 테스트가 되는군요. 그러면 이제 시큐리티 웹 테스트를 해보겠습니다.

```
@Service
public class HelloMessageService {
    @PreAuthorize("authenticated")
    public String getMessage() {
        Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();
        return "Hello " + authentication;
    }
}
```



localhost:8080/message

Hello org.springframework.security.authentication.[PROTECTED]; Enabled: true; AccountNonExpired: true; CredentialsNonExpired: true; Details: org.springframework.security.web.authentication.UsernamePasswordAuthenticationToken; Authorities: ROLE_USER

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

```
@Test(expected = AuthenticationCredentialsNotFoundException.class)
public void 인증되지않은사용자는_AuthenticationCredentialsNotFoundException를_발생시킨다() {
    messageService.getMessage();
}

@Test
@WithMockUser
public void getMessageWithMockUser() {
    String message = messageService.getMessage();
    System.out.println("받은 메시지 : "+message);
    assertNotNull(message);
}
```

6.4. 스프링시큐리티4 - TEST

- 시큐리티 웹 테스트를 하려면 다음 부분을 적용해주면서 import해줘야 합니다~~ 그리고 스프링 테스트는 4.1.3 이상이 필요합니다. (아마 스프링부트에서 버전올리면서 테스트도 버전이 올라갔을 겁니다^^)

```
import static org.springframework.security.test.web.servlet.setup.SecurityMockMvcConfigurers.*;
```

```
@Before
public void setup() {
    mvc = MockMvcBuilders
        .webAppContextSetup(context)
        .apply(springSecurity())
        .build();
}
```



Spring Security's testing support requires spring-test-4.1.3.RELEASE or greater.

- user()로 유저를 주는 부분은 다음 부분을 static import 해주었습니다.

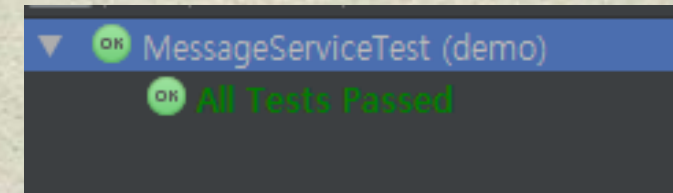
```
import static org.springframework.security.test.web.servlet.request.SecurityMockMvcRequestPostProcessors.*;
```

6.4. 스프링시큐리티4 - TEST

- 모든 테스트가 잘 통과하네요^^
- Admin권한이 필요한 컨트롤러 부분을 admin유저로 테스트를 해보았습니다.

```
@Test
@WithMockUser(username="admin", authorities = {"ADMIN", "USER"})
public void getMessageWithMockUserCustomAuthorities() {
    String message = messageService.getMessage();
    System.out.println("어드민으로 받은 메시지?: "+message);
    assertNotNull(message);
}

@Test
public void mvcAdminTest() throws Exception {
    ResultActions result = mvc.perform(
        get("/admin")
            .with(user("admin").password("pass").roles("USER", "ADMIN")));
    result.andDo(print());
    result.andExpect(status().isOk());
}
```



```
MockHttpServletResponse:
    Status = 200
    Error message = null
    Headers = {X-Content-Type-Options=[n
    Content type = null
    Body =
    Forwarded URL = /WEB-INF/views/admin.jsp
    Redirected URL = null
    Cookies = []
```


- 앗차, 정적자원은 뭐.. 이런 식으로 ignore시킨다고 합니다.^^

```
@Override  
public void configure(WebSecurity web) throws Exception {  
    web.ignoring().antMatchers("/resources/**");  
}
```

- 시큐리티의 더 많은 옵션들이 있지만.. 이제 그만 정리를 하려합니다. 여기까지 봐주신 분들 감사합니다. 혹시 개선해야 되거나, 잘못된 부분 있으시면 메시지주세요~

- 최범균님의 스프링 시큐리티 구조 이해 슬라이드 : <http://www.slideshare.net/madvirus/ss-36809454>
- Spring MVC님의 시큐리티 정리 : <http://egloos.zum.com/springmvc/v/504862>
- 스프링 시큐리티 배포본 : http://javakorean.com/wp2/wp-content/uploads/2014/05/%EC%98%A4%ED%94%88%EC%BB%A4%EB%AE%A4%EB%8B%88%ED%8B%B0_20%EC%B0%A8_Spring+Security+%EB%94%B0%EB%9D%BC%ED%95%98%EA%B8%B0%EB%B0%B0%ED%8F%AC%EB%B3%B8.pdf
- 스프링 시큐리티 레퍼런스 : <http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/>
- 스프링 시큐리티 강좌 : <http://zgundam.tistory.com/category/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D/Spring%20Security>
- Mkyong 스프링 시큐리티 강좌 : <http://www.mkyong.com/tutorials/spring-security-tutorials/>
- 황인서님의 리멤버미설정 참고할 부분 : <http://brantiffy.axisj.com/archives/384>

THANK YOU !

즐거운 개발됩니다.
아라한사 올림



arahansa

페북 : <https://fb.com/me.adunhansa>
트위터 : <https://twitter.com/arahansa>
블로그 : <http://adunhansa.tistory.com/>
사이트 : <http://arahansa.com>