Spring Security

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Spring5

Spring Security

- Web 보안 개요
  - All-in-One Java 애플리케이션 개발, 전병선, 2014 참조
  - Web Application을 작성하면서 보안과 관련해서 고려할 사항은 다음과 같다.
  - 인증(***Authentication***)
    - 사용자가 자신임을 증명하는 속성
    - Principal*의 신원(identity)을 증명하는 과정
    - 자신의 신원 증명 정보(Credential)을 제시
    - 만일 Principal이 User일 경우 Credential은 대체로 Password 이다.
  - 권한(***Authorization***) 부여
    - 사용자가 어떤 Resource에 접근하게 할지를 결정하는 속성
    - 인증을 마친 User에게 권한을 부여하여 대상 Application의 특정 Resource에 접근할 수 있게 허가하는 과정.
    - 반드시 인증 과정 이후 수행돼야 하며, 권한은 Role 형태로 부여하는게 일반적.

*Principal은 User, Device, System 등이 될 수 있지만, 보통 User를 의미한다.

Spring Security (Cont.)

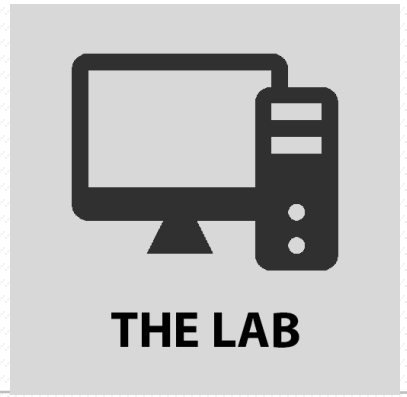
- Web 보안 개요
  - 접근 통제(Access Control)
    - Application Resource에 접근하는 행위를 제어하는 일.
    - 어떤 User가 어떤 Resource에 접근하도록 허락할지를 결정하는 행위, 즉 접근 통제 결정(Access Control Decision)이 뒤따르게 된다.
    - Resource의 접근 속성과 User에게 부여된 권한 또는 다른 속성들을 비교하여 결정한다.
  - 기밀성(***Privacy*** or ***Confidentiality***)
    - 권한이 없는 사용자에게 접근을 허용하지 않게 하는 속성
  - 무결성(***Integrity***)
    - 권한이 없는 사용자가 데이터를 변경하지 못하게 하는 속성
  - 부인방지(***Repudiation***)
    - 사용자가 자신이 했다는 것을 부인하지 못하게 하는 속성

Spring Security 4

- Spring Security는 이들 보안 요소 중에서 인증(***Authentication***)과 권한(***Authorization***) Service를 제공한다.
- ***Authentication***은 사용자가 자신임을 증명하는 것이고, ***Authorization***은 사용자가 어떤 Resource에 접근할 수 있는지를 결정하는 것이다.
- 우리가 Web Application의 Resource에 접근할 수 있는지 여부는 권한과 관련되어 있다.
- 사용자가 권한이 있는지를 알기 위해서는 권한을 가진 사용자인지 인증되어야 한다.
- 그러니까 먼저 사용자가 인증되어야 하고, 인증된 사용자에 부여된 권한으로 Resource에 접근할 수 있는지를 Check한다.

Spring Security 4 (Cont.)

- 사용자를 인증하는 다양한 방법이 있지만 가장 보편적으로 사용하는 방법은 Form 인증 방식이다.
- 사용자가 인증된 후에는 Resource에 접근할 수 있는 권한을 가졌는지를 Check해야 한다.
- 사용자의 권한을 수립하는 첫 번째 단계는 ***Principle*** 형식으로 사용자를 표현하는 것이다.
- ***Principle***은 Resource에 접근하기 위한 보안 식별자가 할당된 계정 보유자로서, 사용자, group, service, computer 등이 ***Principle***이 될 수 있다.
- 개별 사용자에 대해 일일이 권한을 가졌는지 검사하는 것은 비효율적일 뿐만 아니라 경우에 따라서는 불가능하기도 하다.
- 따라서, 사용자에게 권한을 부여할 때 가장 많이 사용하는 일반적인 방법은 사용자를 역할(role)에 할당하는 역할 기반 방식이다.
- `ROLE_ADMIN`, `ROLE_USER`, `ROLE_MNAGER` 등의 역할을 정의하고, 개별 사용자에게 역할을 할당한다.



Task 1. 간단한 인증처리하기

Spring Security Library

- **spring-security-core**
  - Spring Security Framework의 핵심 Class와 Interface를 정의한다.
  - 이 Module은 Spring Security를 사용하는 모든 Application에 필수 항목이다.
- **spring-security-web**
  - Web Application의 보안을 위한 지원을 제공한다.
- **spring-security-config**
  - Spring Security는 Spring **tx** 및 **mvc** Schema와 비슷하게 Spring Security의 기능을 간편하게 구성하기 위한 **security** Schema를 제공한다.
  - **security** Namespace의 요소를 구문분석한다.
- **spring-security-taglibs**
  - 보안 정보에 접근하고 JSP Page에 표시된 내용을 보호하는 Tag를 정의한다.
- **spring-security-acl**
  - ACL(접근 제어 목록)을 사용해 Application에서 Domain 객체의 Instance를 보호할 수 있게 한다.

Web 요청 보안 구성

- Application에서 Web 요청을 보호하는 방법은 다음과 같다.
  - web.xml에서 Spring의 `DelegatingFilterProxy` Filter를 구성한다.
  - Spring Security Framework에서 제공하는 요청 보안을 활성화한다.

Web 요청 보안 구성 (Cont.)

- **DelegatingFilterProxy** Filter 구성
  - spring-web-5.x.x.RELEASE.jar file에 들어있는 Spring Framework의 Web Module Servlet API의 Filter Interface를 구현하는 **DelegatingFilterProxy**를 정의한다.
  - In web.xml 에 code 추가
  - 반드시 Filter의 이름은 **springSecurityFilterChain** 이어야 한다.

```
<filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy
        </filter-class>
</filter>
<filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Web 요청 보안 구성 (Cont.)

- **DelegatingFilterProxy** Filter 구성
  - 앞의 Code에서 **<filter-mapping>**요소에는 **DelegatingFilterProxy** Filter를 들어오는 모든 web 요청과 연결하도록 지정했다.
  - **<filter-name>** 요소에는 **DelegatingFilterProxy** Filter가 요청을 처리하도록 위임할 Spring bean의 이름을 지정한다.
  - 이 Code에서는 **DelegatingFilterProxy** Filter가 수신하는 Web 요청은 Root Application Context의 **springSecurityFilterChain**이라는 Spring bean에 위임된다.

Web 요청 보안 구성 (Cont.)

- **securityContext.xml**

```
<security:http auto-config="true">
        <security:intercept-url pattern="/index.html" access="permitAll" />
        <security:intercept-url pattern="/login.html" access="hasRole('ROLE_USER')" />
        <security:intercept-url pattern="/welcome.html" access="hasRole('ROLE_ADMIN')" />
</security:http>
```

- auto-config='true'를 설정한 것만으로 기본 login page/HTTP 기본인증 /logout 기능 등을 제공.
- 만일 여기서 use-expressions='true'라고 하면 SpEL을 사용한다는 의미이다.
- use-expressions은 기본값이 false이다.
- 이럴 때에는 SpEL을 사용하지 않는다는 의미이다.

Web 요청 보안 구성 (Cont.)

- securityContext.xml

```
<intercept-url pattern="..." access="ROLE_ANONYMOUS" />
<intercept-url pattern="..." access="IS_AUTHENTICATED_ANONYMOUSLY" />
<intercept-url pattern="..." access="permitAll" />
<intercept-url pattern="..." access="ROLE_USER" />
<intercept-url pattern="..." access="ROLE_ADMIN" />
```

  - <intercept-url> 요소의 access 속성은 bool 값으로 평가되는 SpEL 식이다.
  - 해당 URL 에 접근하기위한 권한을 설정한다.
  - 접근가능한 IP 등을 설정할 수도 있다.
  - 그리고 권한은 위쪽이 우선시된다.
  - 만일 SpEL식이 true를 반환하는 경우 사용자는 pattern과 일치하는 URL에 접근할 수 있으며 false를 반환하는 경우 pattern 속성과 일치하는 URL에 대한 접근이 거부된다.

Web 요청 보안 구성 (Cont.)

- **securityContext.xml**
  - Spring Security Framework에는 **hasRole**, **hasAnyRole**, **isAnonymous** 등의 몇 가지 기본식을 제공한다.
  - 만일 **hasAnyRole('ROLE_CUSTOMER', 'ROLE_ADMIN')**식은 사용자에게 **ROLE_CUSTOMER** 또는 **ROLE_ADMIN** 역할이 있는 경우 **true**를 반환한다.
  - 또한 **pattern**이 만일 **/****이면 모든 URL과 일치한다는 뜻이다.

```
<intercept-url pattern="/login" access="permitAll" />
```

  - /login 으로는 모두 허용해준다. /login 을 막아놓으면 안되니까.

```
<intercept-url pattern="/resources/**" access="permitAll" />
```

  - Resource도 허용

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

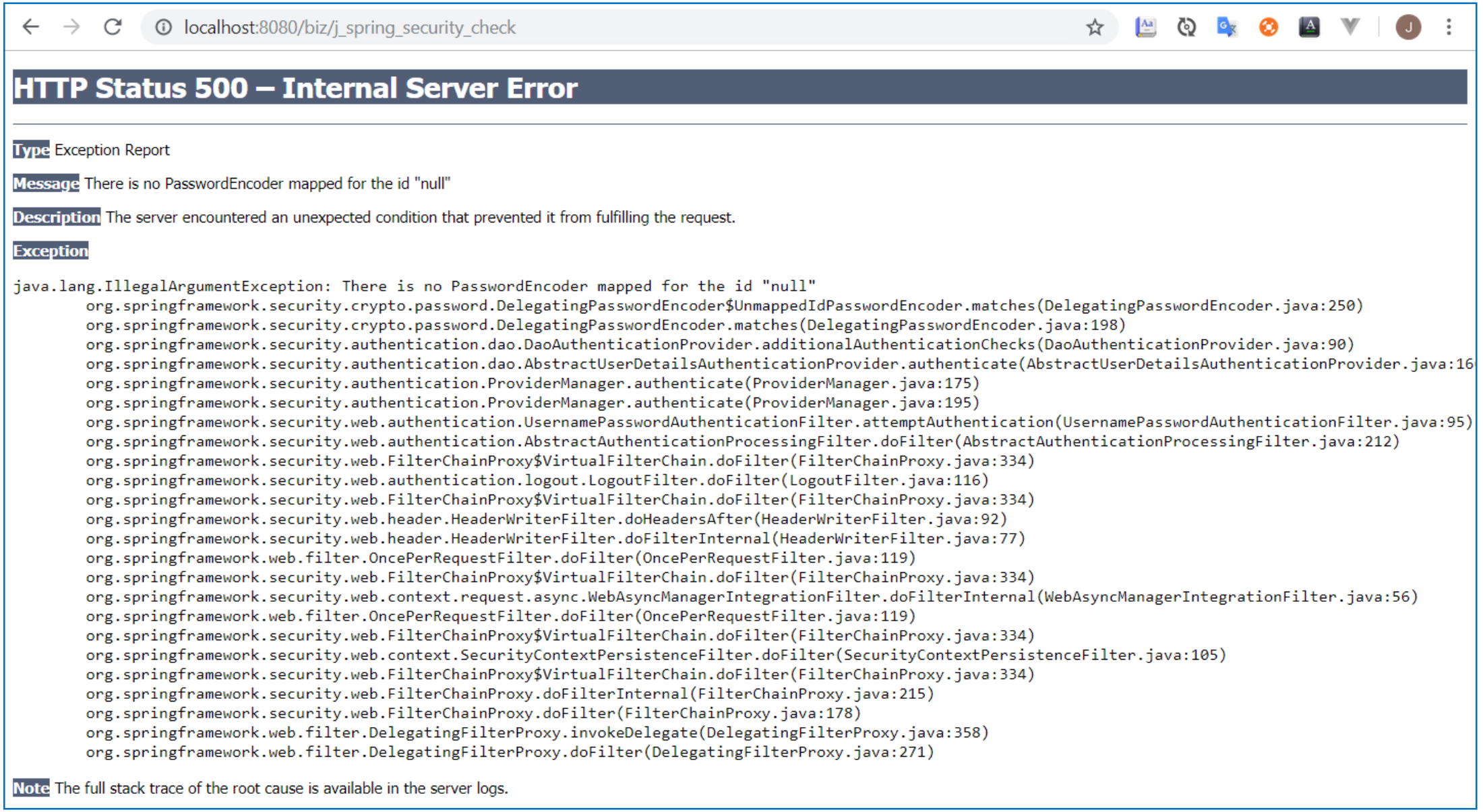
  - 나머지는 모두 **ROLE_USER** 권한을 가진 사람만 허용해준다.

Web 요청 보안 구성 (Cont.)

- securityContext.xml

```
<security:authentication-manager>
	<security:authentication-provider>
		<security:user-service>
			<security:user name="javaexpert" password="{noop}12345678"
					authorities="ROLE_USER"/>
			<security:user name="admin" password="{noop}12345678"
				authorities="ROLE_ADMIN,ROLE_USER"/>
		</security:user-service>
	</security:authentication-provider>
</security:authentication-manager>
```

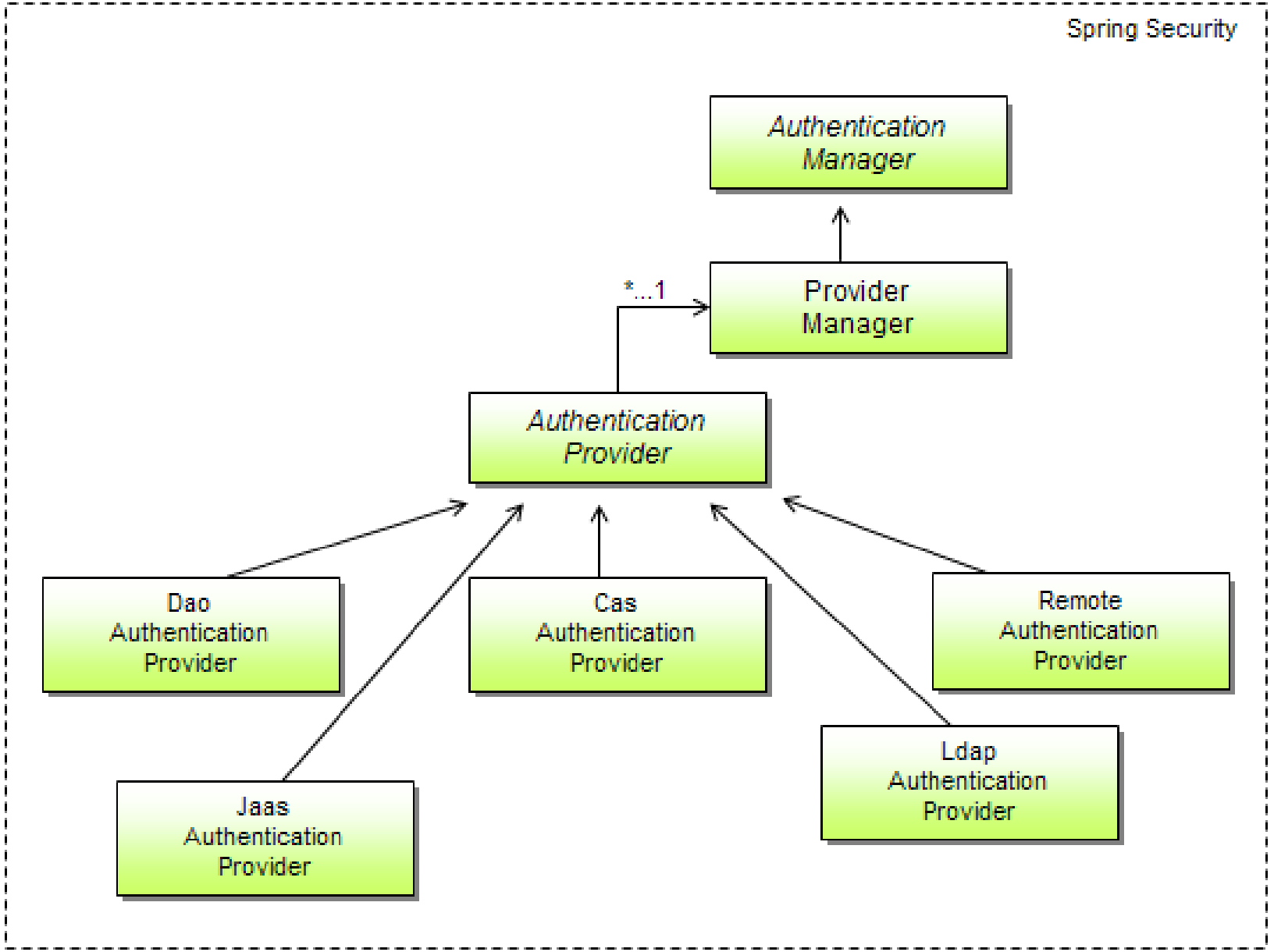
Web 요청 보안 구성 (Cont.)



Web 요청 보안 구성 (Cont.)

- **PasswordEncoder**
  - Spring Security 5부터 반드시 이용하도록 변경됨.
  - 임시로 Formatting 처리를 지정해서 Password Encoding 방식을 지정 가능.
  - https://spring.io/blog/2017/11/01/spring-security-5-0-0-rc1-released#password-storage-format
  - 만일 Password의 Encoding 처리 없이 사용하고 싶다면 Password 앞에 ***{noop}*** 문자열 추가한다.

Web 요청 보안 구성 (Cont.)



Web 요청 보안 구성 Version 4 (Cont.)

- **securityContext.xml**
  - **<authentication-manager>** 요소는 **AuthenticationManager** Instance를 구성하고 **<authentication-provider>**는 **AuthenticationProvider** Instance를 구성한다.
  - 기본적으로 **<authentication-provider>**는 Spring **UserDetailsService**를 DAO로 사용해서 지정된 사용자 이름을 참고해 사용자 저장소에서 사용자 세부 정보를 Load한다.
  - **DaoAuthenticationProvider**는 사용자가 제공한 자격 증명과 구성된 **UserDetailsService**가 Load한 사용자 세부 정보를 비교해 인증을 수행한다.
  - **UserDetailsService**는 데이터 원본(Data Source), 일반 File 또는 다른 사용자 저장소에서 사용자 세부 정보를 Load할 수 있다.

Web 요청 보안 구성 Version 4 (Cont.)

- securityContext.xml
  - <authentication-provider> 하위의 <user-service>는 <user>가 정의하는 사용자를 load하는 memory 내 UserDetailsService를 구성한다.
  - 위의 예제에서는 ROLE_USER와 ROLE_ADMIN의 사용자를 정의했다.
  - name 속성에는 사용자에게 할당된 사용자 이름을 지정하며 password 속성에는 사용자에게 할당된 암호를 지정한다.
  - authorities 속성에는 사용자에게 할당된 role을 지정한다.

```
<authentication-provider user-service-ref="memberService"/>
```

  - 사용자이름/비밀번호를 제공해줄 서비스 등록
  - 위의 예제는 In memory로 처리하고 있음.

Task 2. 로그인 페이지 만들기

Form 기반 인증

```
<form-login login-page="/login"
        default-target-url="/monitering"
        username-parameter="username"
        password-parameter="password"
        authentication-failure-url="/login?error"
        always-use-default-target='true'
/>
```

- 사용자가 만든 login page를 **login-page** 속성을 통해 Spring에게 알려준다.
- 이렇게 설정을 안하면 Spring이 기본적으로 내장된 것을 사용.
- **default-target-url="/monitering"**은 login 성공하면 이동할 page를 설정한다.

Form 기반 인증 (Cont.)

- `authentication-failure-url="/login?error"` 은 login 실패시 호출해줄 URL(login page에 error parameter를 보내준다)
- `always-use-default-target='true '` 이렇게 안하면 login 성공해도 /monitering 로 제대로 안 갈 수 있다.

```
<logout invalidate-session="true" logout-url="/logout"
                        logout-success-url="/login?logout" />
```

- Logout되면 session을 초기화한다.

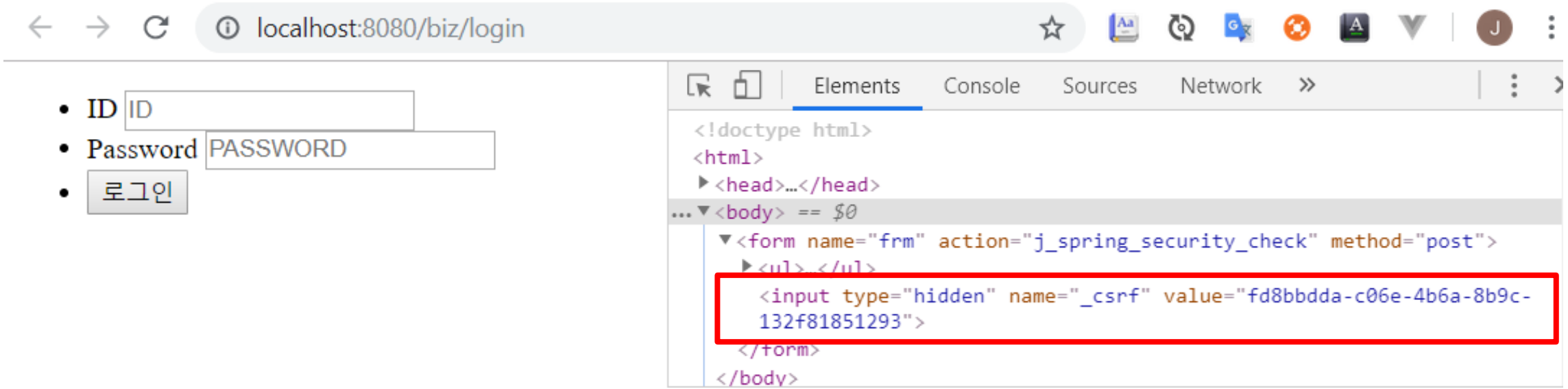
Form 기반 인증 Version 4 (Cont.)

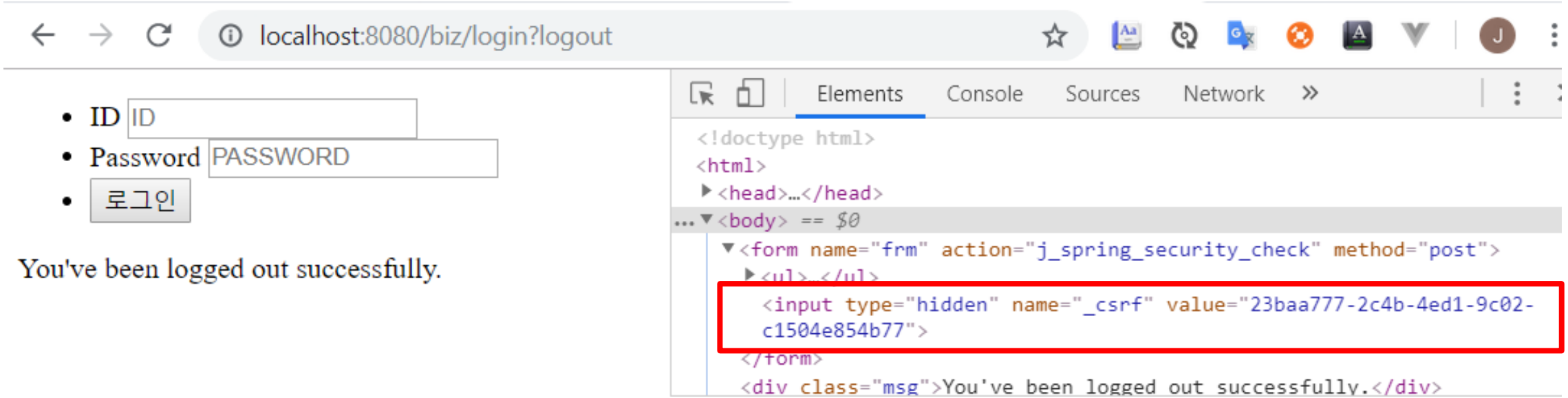
- `logout-success-url="/login?logout"` 은 logout되면 이동할 page
- `logout-url="/logout"` 은 logout을 위한 URL설정.
- 이거 안해주면 default로 `j_spring_security_logout` 해주면됨.
- `<csrf/>` 요소는 간단한 설정으로 csrf 를 통한 해킹을 막을 수 있다.
- CSRF 설명: http://tm-csc.tistory.com/entry/7-WebHacking%EC%9B%B9%ED%95%B4%ED%82%B9-CSRF

CSRF(Cross-Site Request Forgery) 공격과 Token

- Spring Security에서는 POST 방식을 이용하는 경우 기본적으로 CSRF Token을 이용한다.
- 별도의 설정이 없다면 Spring Security가 적용된 Site의 모든 POST 방식에는 CSRF Token이 사용된다.
- Site간 위조 방지를 목적으로 특정한 값의 Token을 사용하는 방식
- 사용자가 임의로 변하는 특정한 Token 값을 Server에서 Check하는 방식.
- Server에는 Browser에 Data를 전송할 때 CSRF Token을 같이 전송한다.
- 사용자가 POST 방식으로 특정한 작업을 수행할 때 Browser에서 전송된 CSRF Token 값과 Server가 보관하고 있는 Token 값을 비교한다.
- 만일 서로의 Token 값이 다르면 작업을 처리하지 않는 방식.
- Server에서 생성하는 Token은 일반적으로 난수를 생성해서 공격자가 Pattern을 찾지 못하게 한다.

CSRF(Cross-Site Request Forgery) 공격과 Token (Cont.)

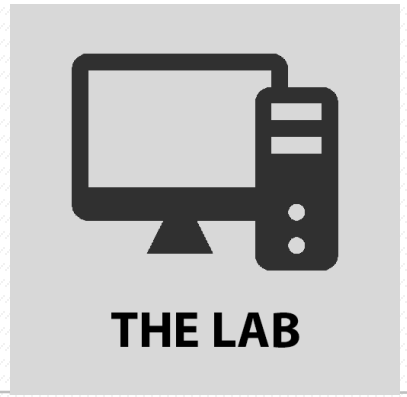




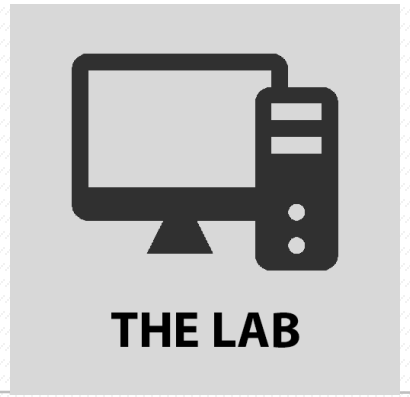
CSRF(Cross-Site Request Forgery) 공격과 Token (Cont.)

- Spring Security의 CSRF 설정
  - 일반적으로 CSRF Token은 Session을 통해서 보관하고, Browser에서 전송된 CSRF Token 값을 검사하는 방식으로 처리한다.
  - Spring Security에서는 CSRF Token 생성을 비활성화하거나 CSRF Token을 Cookie를 이용해서 처리하는 등의 설정을 지원한다.

```
<security:csrf disabled="true" />
```



Task 3. login, logout 상태 표시하기



Task 4. UserDetailsService 이용하기

UserDetailsService

- 인증과 권한에 대한 실제 처리는 **UserDetailService**를 이용해서 처리하는데, XML에서는 다음과 같이 지정할 수 있다.

```
<security:authentication-manager>
        <security:authentication-provider user-service-ref="memberService" />
</security:authentication-manager>
```

- OR

```
<security:authentication-manager>
        <security:authentication-provider>
                <security:user-service>
                        <security:user name="member" password="member"
                                                        authorities="ROLE_MEMBER />
                </security:user-service>
        </security:authentication-provider>
</security:authentication-manager>
```

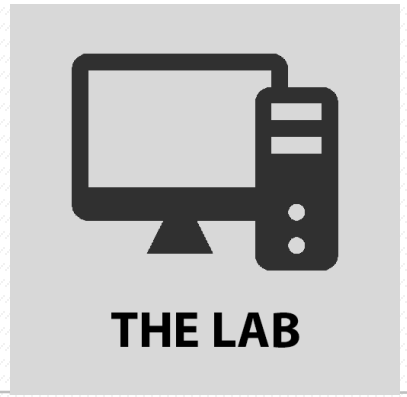
UserDetailsService (Cont.)

```
public class MemberService  implements UserDetailsService{
        . . .
        @Override
        public UserDetails loadUserByUsername(String username)
                                                        throws UsernameNotFoundException {
                . . .
                Collection<SimpleGrantedAuthority> roles =
                                        new ArrayList<SimpleGrantedAuthority>();
                roles.add(new SimpleGrantedAuthority("ROLE_USER"));
                UserDetails user = new User(username, userVO.getPassword(), roles);
                return user;
        }
}
```

보안관련 taglibs 사용하기

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

<security:authorize access="hasRole('ROLE_USER')">
      <p>현재 로그인 상태</p>
</security:authorize>
<security:authorize access="hasRole('ROLE_ANONYMOUS')">
      <p>현재 로그아웃 상태</p>
</security:authorize>
```



Task 5. 보안관련 taglibs 사용하기

Task 6. Database-based Login Authentication

Database-based Login Authentication

```
<security:authentication-manager>
    <security:authentication-provider>
        <security:jdbc-user-service
            data-source-ref="dataSource“ />
        <security:password-encoder
            ref=“customPasswordEncoder” />
    </security:authentication-provider>
</security:authentication-manager>
```