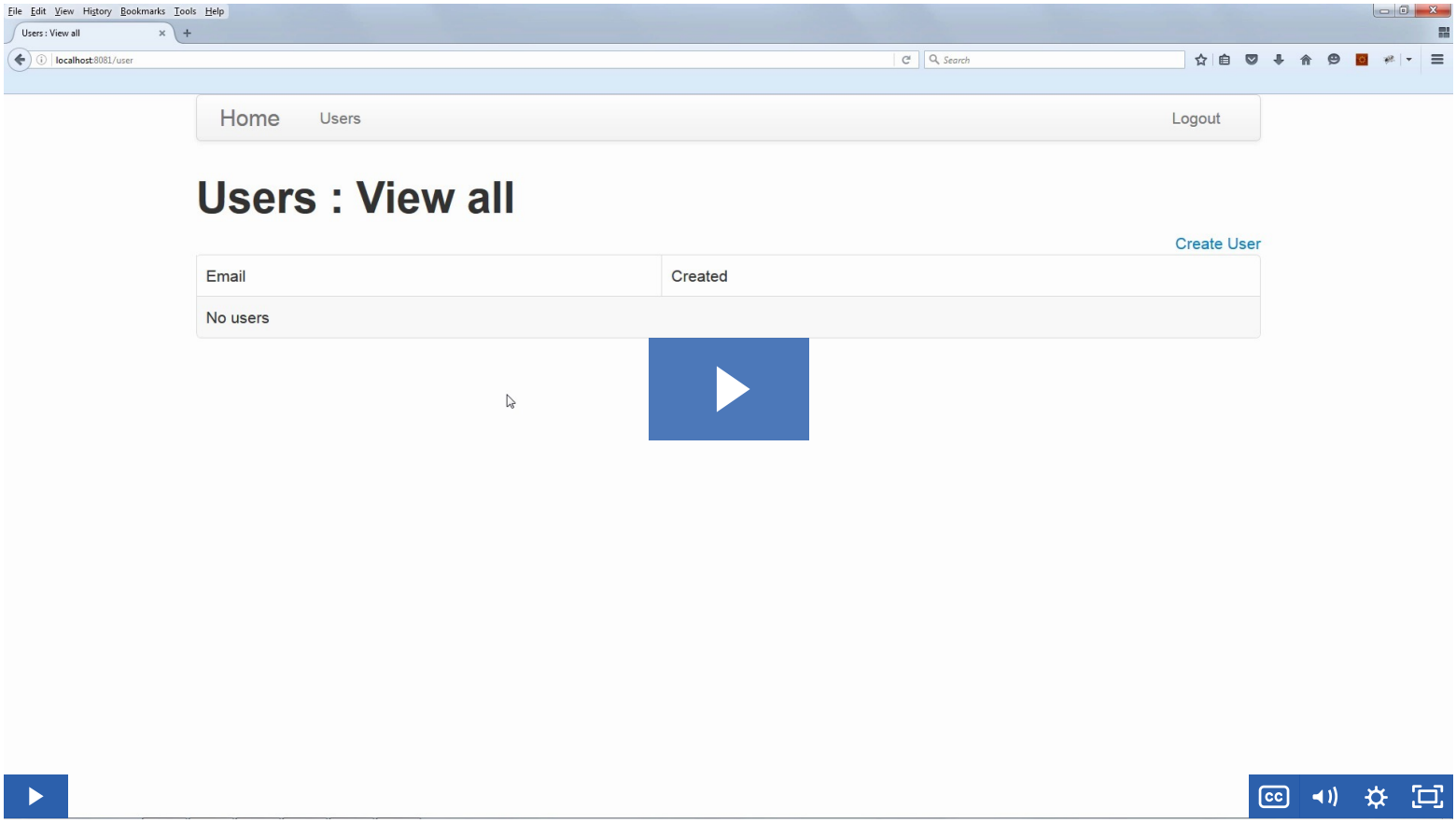
Lesson 1: Spring Security with JSP



1. Main Goal

The goal of this lesson is to show how the Spring Security tags work after temporarily making the switch from Thymeleaf to JSP.

2. Lesson Notes

As you're using the git repository to get the project - you should be on the module4 branch.

Or, on the corresponding Spring Boot 2 based branch: module4-spring-boot-2.

The relevant module you need to import when you're starting with this lesson is: m4-lesson1

If you want to skip and see the complete implementation, feel free to jump ahead and import: m4-lesson2

A quick note is that this lesson is **using Spring Boot**.

The credentials used in the code of this lesson are: user/pass (in-memory).

Let's start by adding the Maven dependencies related to JSP support:

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

Next - if we are using Boot - we can easily configure JSP via the *application.properties* file of the project:

```
spring.mvc.view.prefix: /WEB-INF/jsp/
spring.mvc.view.suffix: .jsp
```

Now, the JSP files are under:

```
/src/main/webapp/WEB-INF/jsp
```

Of course the actual client-side files are them same - but they've been migrated from Thymeleaf to JSP.

Next, let's use **our first Spring Security JSP tag** - the *sec:authorize*.

We're going to first add the support for the tagging library to the *layout.jsp*:

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

And then we're going to add 2 new elements in the menu:

- one is visible to Admins:

```
<sec:authorize access="hasRole('ROLE_ADMIN')">
    <a class="brand" href="/"> Admins Home </a>
</sec:authorize>
```

- and the other visible to standard Users

```
<sec:authorize access="hasRole('ROLE_USER')">
    <a class="brand" href="/"> Users Home </a>
</sec:authorize>
```

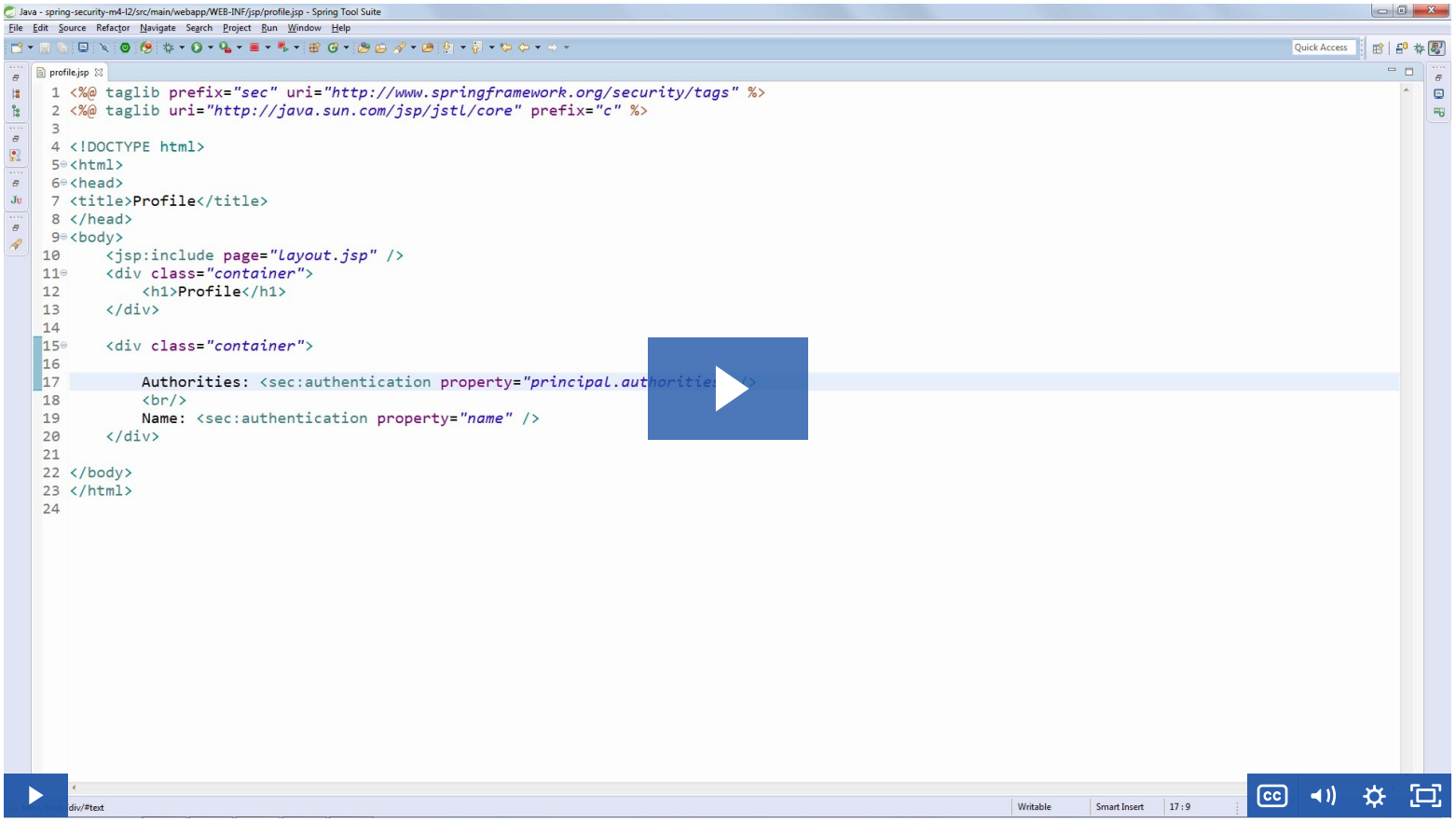
Finally, we'll use the *sec:authentication* tag - and we'll put the username right next to the logout:

```
<li>
    <a href="#" class="menu-right" > Hi, <sec:authentication property="principal.username" /> </a>
```


3. Resources

- JSP Tag Libraries (in the official reference)

Lesson 2: The Authentication Tag and Displaying the Current User



1. Main Goal

The focus in this module is to explore the security authentication tag with practical examples.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m4-lesson2

If you want to skip and see the complete implementation, feel free to jump ahead and import: m4-lesson3

The credentials used in the code of this lesson are: user/pass (in-memory).

Before we start, we're going to be working on a new page - the profile page - which is already set up in the system. Setting up a new page is something we've done before so there's no point looking at that in the video lesson.

Keep in mind that, in the previous lesson, we already had a simple example using *authentication*:

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

Let's now go beyond that and also display the authorities of the principal:

```
<sec:authentication property="principal.authorities" />
```

Next, let's access the *Authentication* object directly:

```
<sec:authentication property="name" />
```

This object corresponds to the *org.springframework.security.core.Authentication* interface.

A quick note here is that - the *name* property here will be resolved to the *getName* getter method on that authentication object.

What's more, every getter that's available in the API of that object can be similarly accessed as well.

For example:

```
<sec:authentication property="authorities" />
```

What's even more interesting is - we can also access custom properties if we set them up to be available.

For example, if we have a custom principal implementation and we make sure that has a new field - for example - *organization*; we can then access that field in the same way:

```
<sec:authentication property="principal.organization" />
```

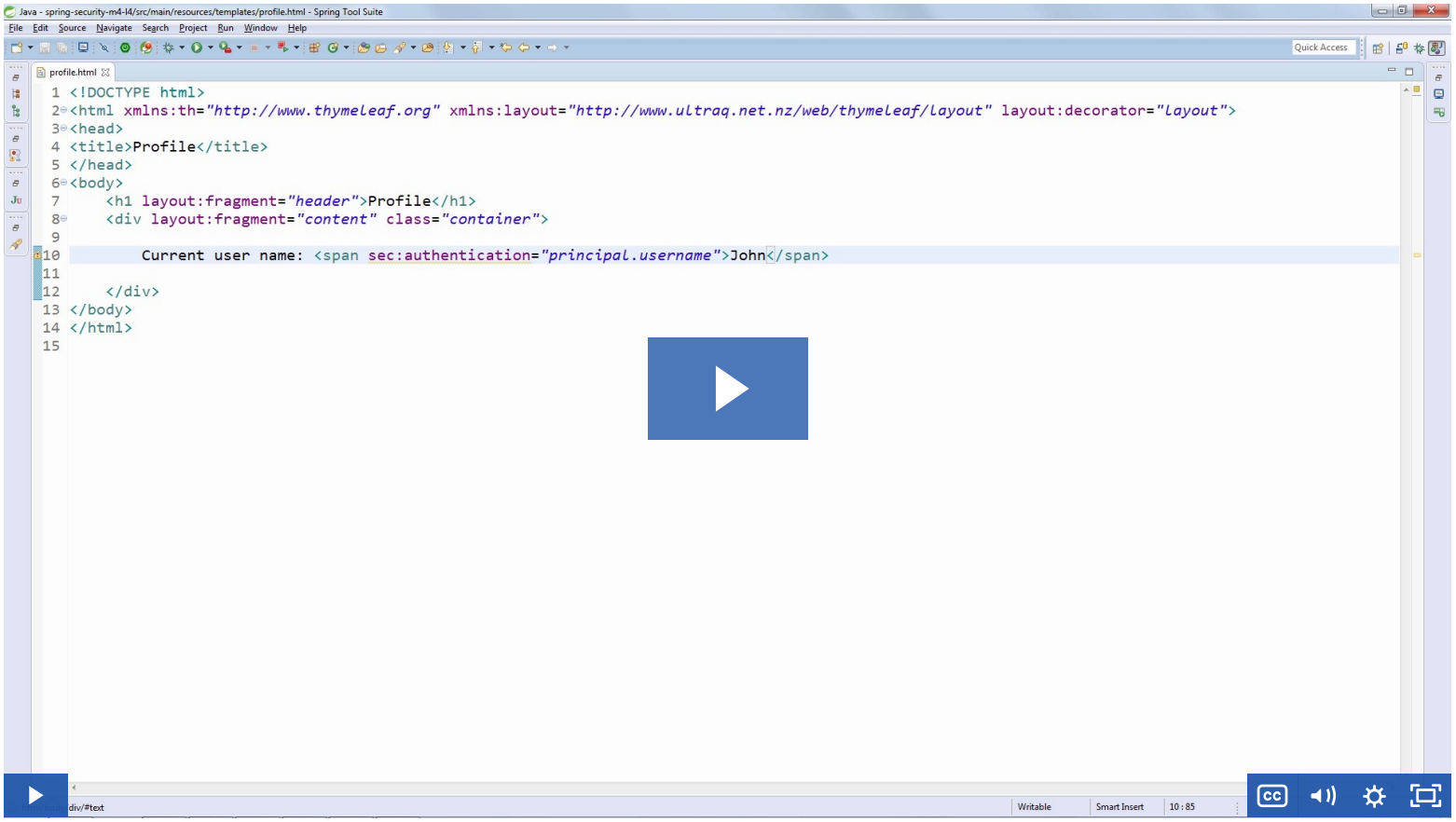
Finally, let's declare a variable and then reference it (use it) on the page:

```
<sec:authentication property="principal.username" var="currentUserName" scope="page"/>
<c:if test="${currentUserName.startsWith('u')}">
    <div>User name starts with 'u'</div>
</c:if>
```

3. Resources

- the authentication tag (in the official reference)

Lesson 3: Spring Security with Thymeleaf



1. Main Goal

The focus of this lesson is to go back to Thymeleaf and show the Spring Security client side support.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m4-lesson3

If you want to skip and see the complete implementation, feel free to jump ahead and import: m4-lesson4

And, on this lesson in particular, it's important to import the correct project. If you import *m4-lesson3*, be aware that's still using JPSs because it represents the starting point. You can remove the JSPs and basically implement the Thymeleaf support following the video here.

Alternatively, if you'd like to see the full Thymeleaf implementation, make sure you import the correct project - *m4-lesson4*.

The credentials used in the code of this lesson are:
user/pass (Constructor)
john/123 (Constructor)
tom/111 (Constructor) (has ADMIN role)

And before we start, another quick note - we're back to our normal Thymeleaf setup we had in all previous lessons. We briefly switched the app to JSP so that we can explore it and now we're back.

Let's start adding the Spring Security Thymeleaf support in the pom:- now, back on the profile page

```
<dependency>
   <groupId>org.thymeleaf.extras</groupId>
   <artifactId>thymeleaf-extras-springsecurity4</artifactId>
</dependency>
```

Now we're going to implement some of the previous examples using the *authentication* tag:

```
Current user name: <span sec:authentication="principal.username">John</span>
```

This is definitely a different syntax here:

- notice how we no longer have an actual element *<sec:authentication ...*
- we now have a standard HTML element - a *span* - and *sec:authentication* is just an attribute
- also notice that we're actually providing a default value here - *John*

A quick side-note - these syntax choices in Thymeleaf make these files a lot more flexible to work on and edit in a wide variety of scenarios.

For example - if we're running this as a server rendered template within a Java application - cool - we're going to get the real value.

However, a front-end developer is still able to work on this file entirely separately and offline with no Java back-end.

OK, let's now continue adding examples:

```
<br/>
Current user name 2: <span sec:authentication="name">John</span>
<br/><br/>
Current user roles:
<span sec:authentication="principal.authorities">[ROLE_USER, ROLE_ADMIN]</span>
<br/>
Current user roles 2:
<span sec:authentication="authorities">[ROLE_USER, ROLE_ADMIN]</span>
```

Note: The video lesson is missing following namespace in the profile.html:

xmlns:sec="http://www.thymeleaf.org/extras/spring-security"

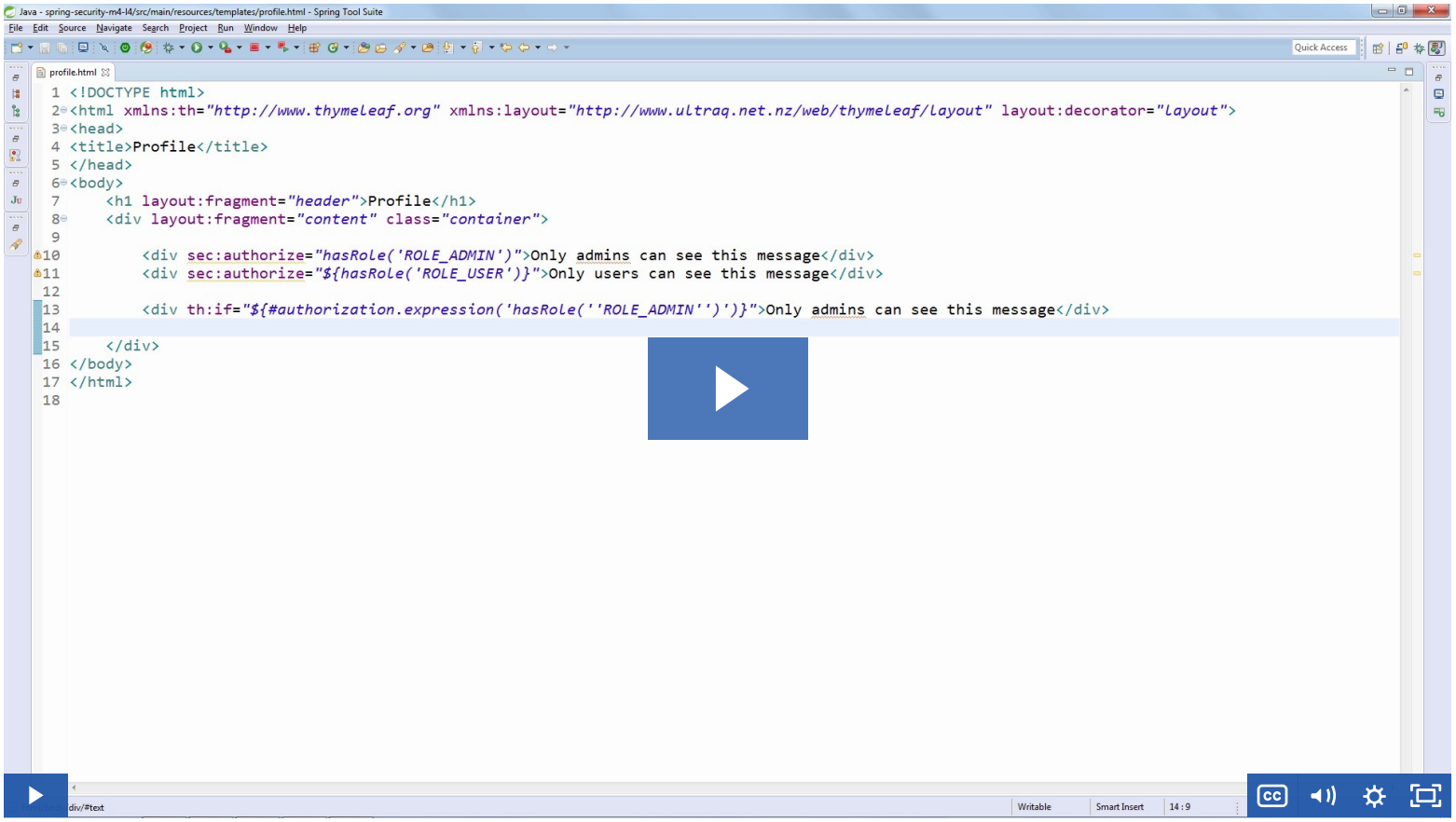
The missing namespace doesn't impact processing of the template but IDE usually flags this as a potential problem.

3. Resources

- Thymeleaf + Spring Security integration basics

- The thymeleaf-extras-springsecurity project on Github

Lesson 4: The Authorize Tag



1. Main Goal

The focus of the lesson is to explore the *authorize* security tag with practical examples.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m4-lesson4

If you want to skip and see the complete implementation, feel free to jump ahead and import: m4-lesson5

The credentials used in the code of this lesson are:
user/pass (Constructor)
john/123 (Constructor)
tom/111 (Constructor) (has ADMIN role)

Let's start with a simple example using *sec:authorize*:

```
<div sec:authorize="hasRole('ROLE_USER')">
    Only users can see this message
</div>
<div sec:authorize="hasRole('ROLE_ADMIN')">
    Only admins can see this message
</div>
```

Note that *sec:authorize* works equivalently to a *th:if* that evaluated *an #authorization.expression(...)* expression (which we'll look at later).

The expression is actually just a standard SpEL expression; the only difference is that it gets evaluated on **a security specific root node**.

Actually, if we want to **use expression syntax** - we can:

```
<div sec:authorize="${hasRole('ROLE_USER')}">
```

Now, let's access the *authorization* object in a standard Thymeleaf tag:

```
<div th:if="${#authorization.expression('hasRole(''ROLE_ADMIN'')')}">
    Only admins can see this message
</div>
```

The *#authorization* object is an instance of *org.thymeleaf.extras.springsecurity[3|4].auth.Authorization*.

Finally, the next example is determining authorization by delegating the URL authorization - so, if the principal has the necessary authority to access that URL, then this will resolve to *true*:

```
<div sec:authorize-url="/user/delete/1">
    Only users who can call "/user/delete/1" URL can see this message
</div>
```

Looking at the security config, we need the ADMIN authority to access that URL. And since we don't have it - we're not going to see the element on the page.

Also note how the URL in the security config and the URL in the client side expressions don't match exactly - yet Spring understand they actually do match.

Finally, a less used syntax - you can actually also **check for a specific HTTP method** with the following syntax:

```
<div sec:authorize-url="POST /user/delete/1">
    Only users who can POST to /user/delete/1 will see this message
</div>
```

3. Resources

- The authorize Tag (in the official reference)