Course Introduction

The Starter Class

The Starter Class contains the first 5 modules:

- Secure a Simple Spring MVC Application (7 lessons)
- A Full Registration Flow (6 lessons)
- Remember Me (4 lessons)
- Spring Security On the Client (4 lessons)
- Spring Expressions (4 lessons)

Each Module is organized in several video Lessons - containing lesson notes and other resources.

The material is meant to be experienced as video, and **the lesson notes should be used as a reference not as comprehensive documentation**.

Project Code and IDE

Each course provides the full project code - hosted over on Github.

Notice that each lesson has a small section that explains exactly which project you need to start with when you start the lesson. For example:

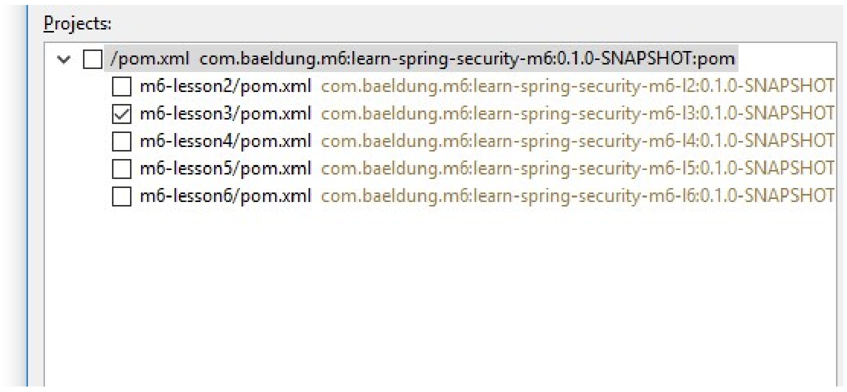
The relevant module you need to import for this lesson is: m1-lesson2

So, in case you're not sure exactly what project you should import and work on a particular lesson - that's where you'll need to look.

All projects use Maven, so you can work with any IDE you're familiar with.

A quick (optional) recommendation is using Eclipse STS.

To import the project into Eclipse, you'll simply need to go to File - Import - Maven - Existing Maven Project:


Projects:

```
☐ /pom.xml  com.baeldung.m6:learn-spring-security-m6:0.1.0-SNAPSHOT:pom
    ☐ m6-lesson2/pom.xml  com.baeldung.m6:learn-spring-security-m6-l2:0.1.0-SNAPSHOT
    ☑ m6-lesson3/pom.xml  com.baeldung.m6:learn-spring-security-m6-l3:0.1.0-SNAPSHOT
    ☐ m6-lesson4/pom.xml  com.baeldung.m6:learn-spring-security-m6-l4:0.1.0-SNAPSHOT
    ☐ m6-lesson5/pom.xml  com.baeldung.m6:learn-spring-security-m6-l5:0.1.0-SNAPSHOT
    ☐ m6-lesson6/pom.xml  com.baeldung.m6:learn-spring-security-m6-l6:0.1.0-SNAPSHOT
```

Getting the Project from Github

The project code is hosted over on Github. To clone it, simply run the following command:

```
git clone https://github.com/eugenp/learn-spring-security.git
```

And to switch the the branch corresponding to the module you're on - for example, Module 2:

```
cd learn-spring-security
git checkout module2
```

Building the Project

Let's start with the JDK - the project is using JDK 8, so you'll need to make sure that's the default JDK you have installed locally.

To simply build the entire project - just run the standard Maven build:

```
mvn clean install
```

Notice that we're not skipping tests - at least not on the first run. This is important because a full run with tests will prepare the testing jars and install them in your local maven repository. After that point, you can run the build with tests skipped if you need to:

```
mvn clean install -Dmaven.test.skip=true
```

Running the Project

The project code is generally ready to go.

The recommended way to run the project is using Spring Boot - each lesson has a simple runner called *LssAppX.java* (for example, *LssApp1.java*)

Note that the port configured for the project is 8081 (not 8080) - which you can of course change in *application.properties* if you need to.

Non-Boot Deployment

We're using Boot to deploy in all standard modules.

The reason is simple - it's easier to deploy with Boot, and deployment isn't the focus of the course, so the simpler option is best.

However, if - for whatever reason - you do need an example of a non-Boot deployment, you can access this here.

Also note that, if you want to see an example of a Boot application that results a a stand-alone war deployable - you can have a look at the first lesson of module 2.

MySQL Setup

Most of the lessons don't use or need MySQL - they rely on an embedded HSQLDB.

There are a few lessons though where we do need to switch to MySQL - and for these lessons, you'll need to set up a local MySQL Server.

After installing the server, make sure you create the default user to allow the application to access it:

```
restUser / restmy5ql
```

This should be a standard user with full permissions.

Bonus Modules

After quite a bit of feedback from students going through the material, I've planned out a few bonus modules based on the most common questions.

Most of these will go live throughout **Q4 (2018)**.

Asking Questions

I structured the course material to make sure you have everything you need to go through it, and by this point, thousands of students have gone through it.

But, of course, if you do have any question, you can always reach out and ask me directly.

I've set up a specific inbox for this and, of course, I check it daily: course@baeldung.com

Resources

The git repository of the project, hosted over on Github:

- The Module 1 branch

- The Module 2 branch

- The Module 3 branch

- The Module 4 branch

- The Module 5 branch

The Project in your IDE (NEW)

1. Overview

The primary IDE I used for the course is Eclipse.

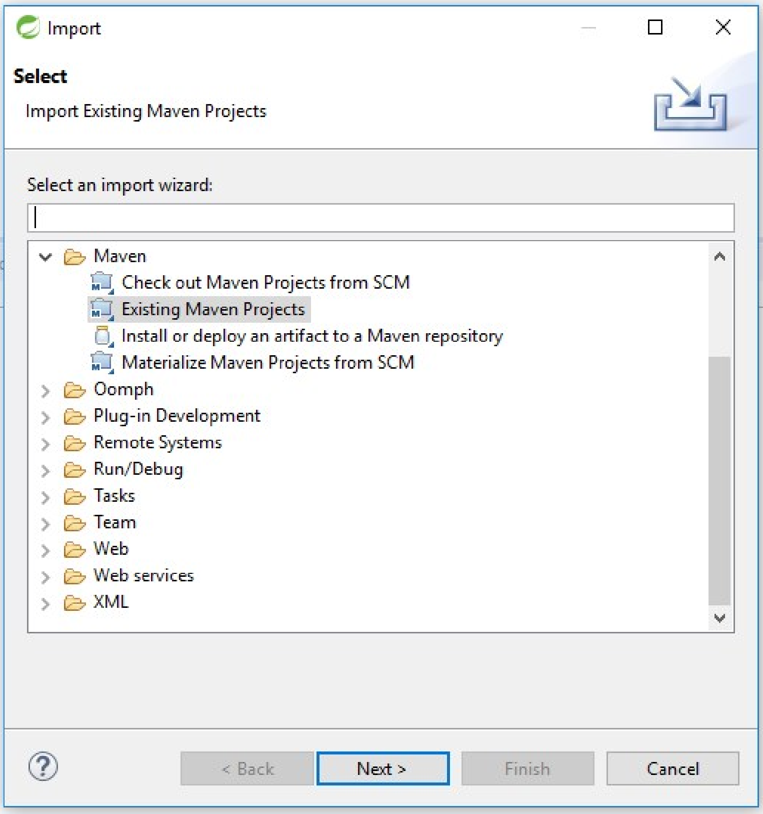
The project code is of course a Maven project, so you can use any IDE you want.

If you are going to use Eclipse, I recommend using the STS distribution. And I generally recommend using the latest version here.

2. Importing the Project

In Eclipse, there are several ways you can import a project.

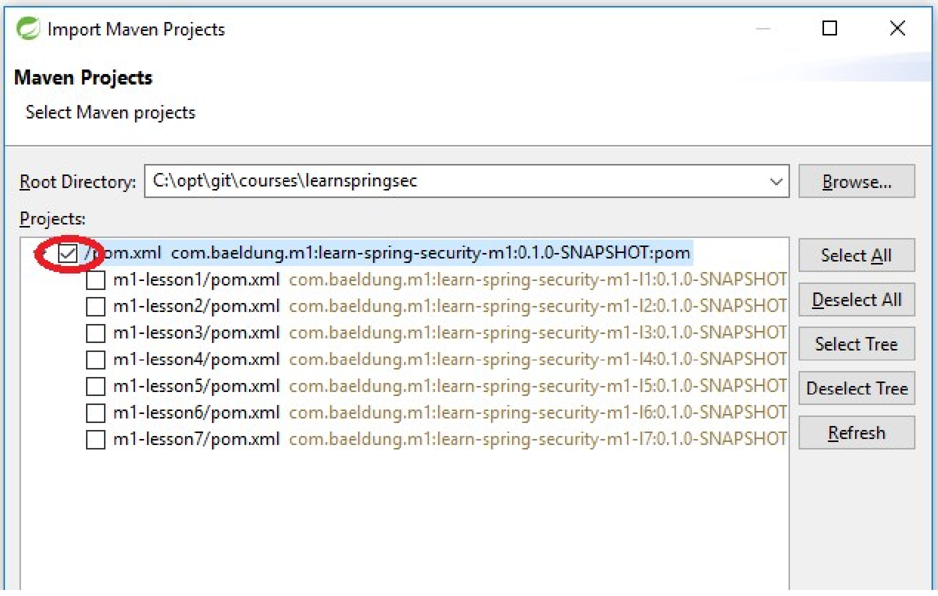
Since this is a Maven project, you can use the "Existing Maven Project" option:



2.1. Importing the Parent

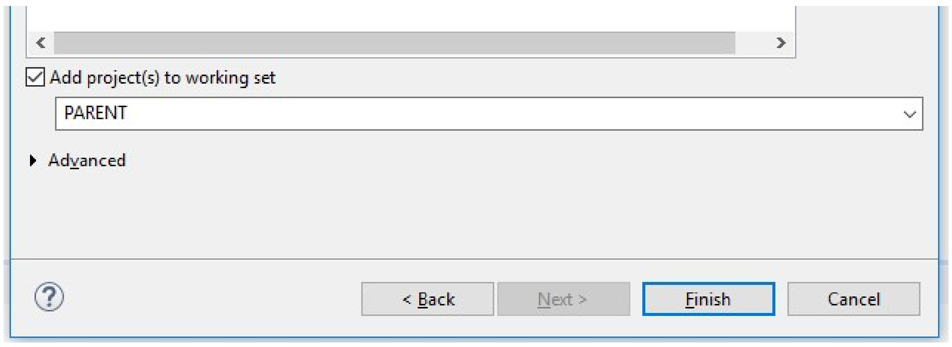
Now, you can import the parent modules or you can import the individual modules of a lesson.

To run the main build, you'll need to **import the top parent**:



Root Directory: C:\opt\git\courses\learnspringsec

- [✓] /pom.xml com.baeldung.m1:learn-spring-security-m1:0.1.0-SNAPSHOT:pom
- [ ] m1-lesson1/pom.xml com.baeldung.m1:learn-spring-security-m1-l1:0.1.0-SNAPSHOT
- [ ] m1-lesson2/pom.xml com.baeldung.m1:learn-spring-security-m1-l2:0.1.0-SNAPSHOT
- [ ] m1-lesson3/pom.xml com.baeldung.m1:learn-spring-security-m1-l3:0.1.0-SNAPSHOT
- [ ] m1-lesson4/pom.xml com.baeldung.m1:learn-spring-security-m1-l4:0.1.0-SNAPSHOT
- [ ] m1-lesson5/pom.xml com.baeldung.m1:learn-spring-security-m1-l5:0.1.0-SNAPSHOT
- [ ] m1-lesson6/pom.xml com.baeldung.m1:learn-spring-security-m1-l6:0.1.0-SNAPSHOT
- [ ] m1-lesson7/pom.xml com.baeldung.m1:learn-spring-security-m1-l7:0.1.0-SNAPSHOT



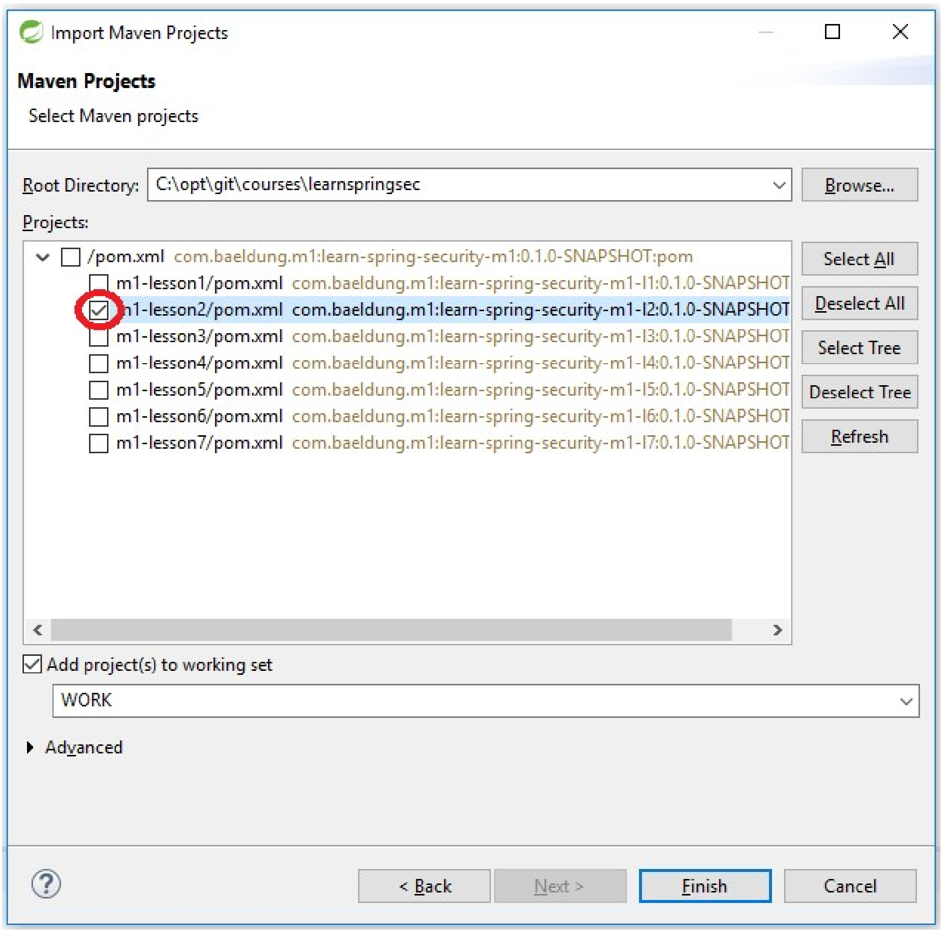
Notice how - here - I'm only importing the top parent so that we can build that.

Also notice (this is optional) - how I'm adding that to a working set called PARENT. I'm doing that because I like to keep the parent and the working modules separate.

2.2. Importing the Children

Now, you can import the children modules - the modules of a single lesson.

Let's say we want to import the final code of Lesson 2 from Module 1:


Import Maven Projects

Maven Projects

Select Maven projects

Root Directory: C:\opt\git\courses\learnspringsec

Projects:

```
☐ /pom.xml  com.baeldung.m1:learn-spring-security-m1:0.1.0-SNAPSHOT:pom
   ☐ m1-lesson1/pom.xml  com.baeldung.m1:learn-spring-security-m1-l1:0.1.0-SNAPSHOT
   ☑ m1-lesson2/pom.xml  com.baeldung.m1:learn-spring-security-m1-l2:0.1.0-SNAPSHOT
   ☐ m1-lesson3/pom.xml  com.baeldung.m1:learn-spring-security-m1-l3:0.1.0-SNAPSHOT
   ☐ m1-lesson4/pom.xml  com.baeldung.m1:learn-spring-security-m1-l4:0.1.0-SNAPSHOT
   ☐ m1-lesson5/pom.xml  com.baeldung.m1:learn-spring-security-m1-l5:0.1.0-SNAPSHOT
   ☐ m1-lesson6/pom.xml  com.baeldung.m1:learn-spring-security-m1-l6:0.1.0-SNAPSHOT
   ☐ m1-lesson7/pom.xml  com.baeldung.m1:learn-spring-security-m1-l7:0.1.0-SNAPSHOT
```

☑ Add project(s) to working set: WORK

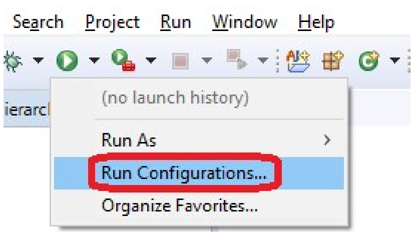
Notice how we're only importing the modules of that particular lesson.

3. Building in Eclipse

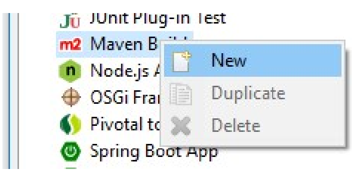
Now that we have the modules imported, it's important to know how to run a Maven build in Eclipse.

There are several ways of doing that, but I recommend following this one because it's simple and flexible.

First, let's define a Run Configuration in Eclipse:



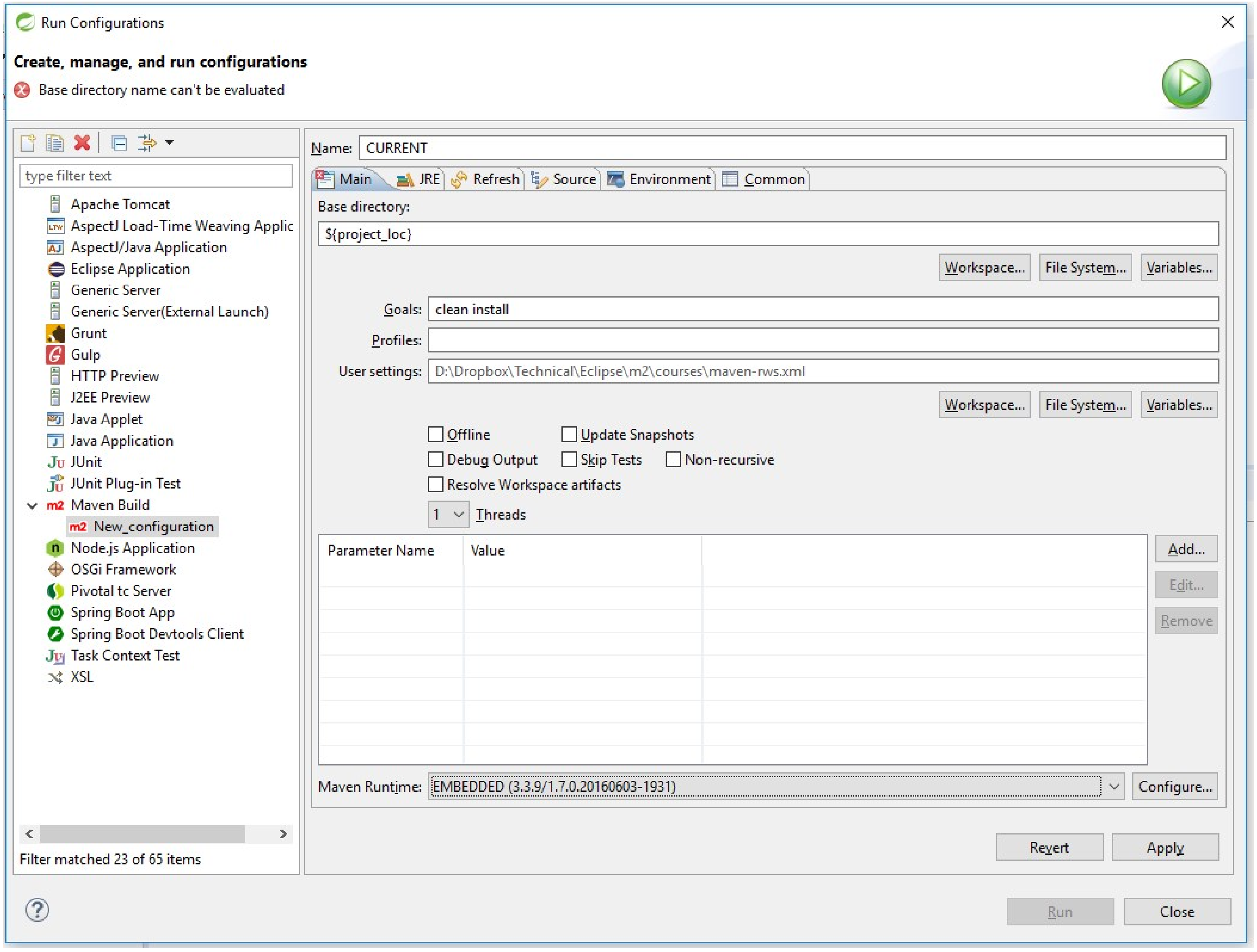
Then, create a new Maven run configuration:



Set the:

- Base directory to: *${project_loc}*
- Goals: *clean install*

Here's what that looks like:


Run Configurations

Create, manage, and run configurations

Base directory name can't be evaluated

Name: CURRENT

Base directory: ${project_loc}

Goals: clean install

Profiles:

User settings: D:\Dropbox\Technical\Eclipse\m2\courses\maven-rws.xml

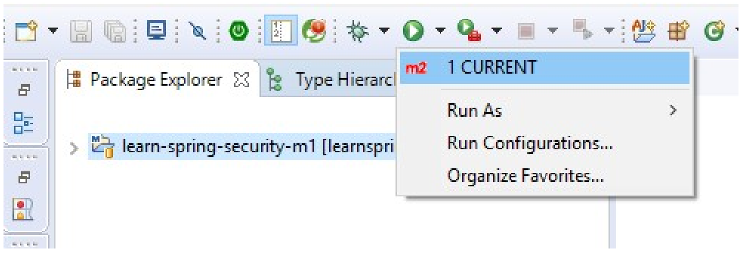
Maven Runtime: EMBEDDED (3.3.9/1.7.0.20160603-1931)

Filter matched 23 of 65 items

That's it. Now, the run configuration is ready to go and you're ready to build any project in your IDE.

What's important to understand here is the *Base directory* - that points to whatever project you have selected in your package explorer.

So - if you want to build the parent, you select the parent, and run this run config:



This is a very flexible way to define a single run configuration and use it for whatever project you have imported in your IDE.

4. Before You Start

Before you start working on a new module, make sure to import the parent of that module into your IDE like this - and run a build on it.

Don't start working if you don't have a successful build at this point.

Lesson 1: Intro to Spring Security

1. Main Goal

The goal of this first lesson is to guide you through the basic info about the course, explain the high level structure of the Starter Class and introduce you to the codebase we're going to be using.

2. Lesson Notes

As you're using the git repository to get the project - you should be on the module1 branch.

Or, on the corresponding Spring Boot 2 based branch: module1-spring-boot-2.

The relevant module you need to import when you're starting with this lesson is: m1-lesson1

If you want to revisit how the Starter Class is structured or how to get the project from Github, have a quick look at the previous lesson.

2.1. Spring Boot Usage

The general approach when using Boot is simple - you won't need Boot, but if you want to learn about it and use it, there are plenty of Boot lessons as well. These lessons will be clearly marked to make it clear that they're relying on Boot to some extent.

Also there's always going to be a clear alternative in case you want to stay away from using Boot altogether.

So, to start with - one clear and consistent usage of Spring Boot will be running the application. Of course you could deploy the app as a stand-alone deployable unit in a separate web server - but the Boot runner is more convenient for quick development.

2.2. Spring MVC Usage

The project code uses some simple Spring MVC concepts. If you have basic experience with Spring MVC, you'll be able to understand everything there, but if you want to brush up on your Spring MVC at any point - here are some solid writeups to do just that:

- Spring RequestMapping
- Basic Forms with Spring MVC
- Exploring SpringMVC's Form Tag Library

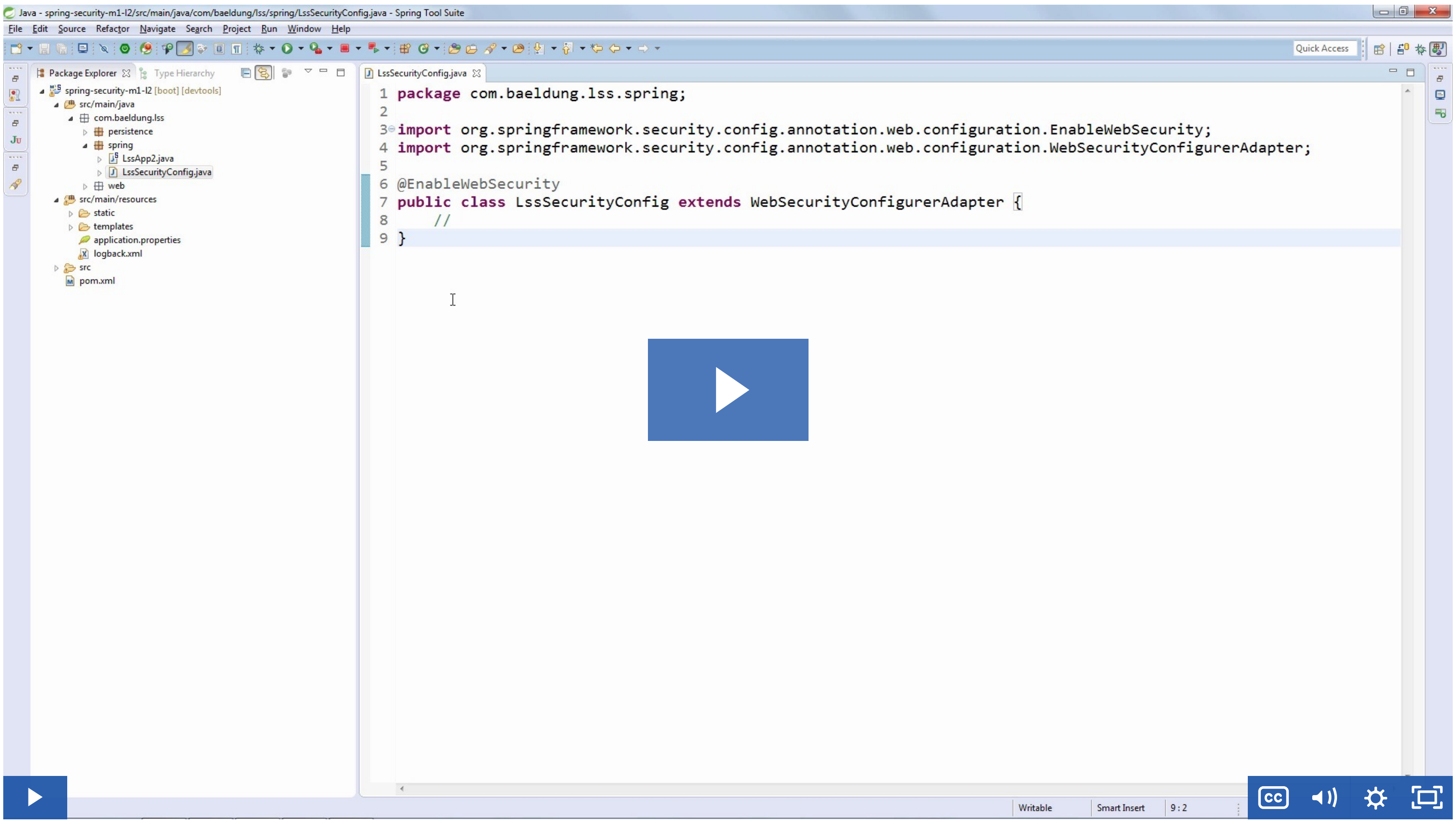
Finally, a quick note about the first two lessons - while the video is showing the code run on port 8080, the code actually runs on 8081 for the entire course.

3. Resources

- The Project Code on Github

- The Spring Security Homepage

Lesson 2: A Basic Security Java Config


Java - spring-security-m1-l2/src/main/java/com/baeldung/lss/spring/LssSecurityConfig.java - Spring Tool Suite

```
package com.baeldung.lss.spring;

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

@EnableWebSecurity
public class LssSecurityConfig extends WebSecurityConfigurerAdapter {
    //
}
```

1. Main Goal

The goal of this lesson is to guide you through a simple Java based Security Configuration for the project.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m1-lesson2

If you want to skip and see the complete implementation, feel free to jump ahead and import: m1-lesson3

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

2.1. Using Spring Boot

First, here's the Maven dependency you need to add to enable security in a Boot project:

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

As soon as you add the dependency, security will be implicitly enabled.

You can configure most of the defaults using properties in *application.properties* - for example:

```
security.user.name=user
security.user.password=pass
security.basic.authorize-mode=authenticated
security.basic.path=/**
```

2.2. Not Using Boot

If you're not using Boot, you need to define the following dependencies:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${spring-security.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${spring-security.version}</version>
</dependency>
```

2.3. The Simple Java Security Configuration

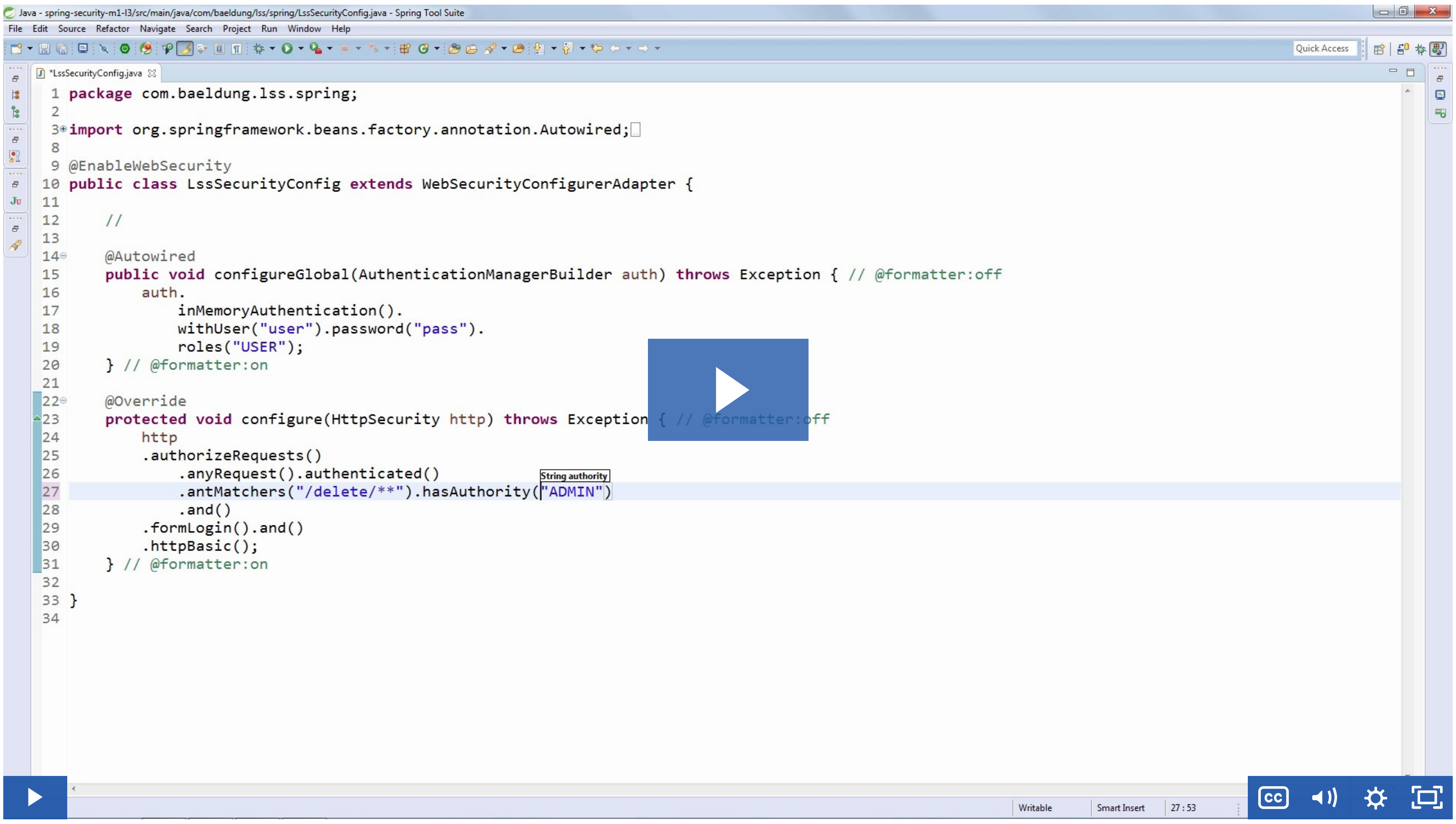
Let's start with a basic config:

```
@EnableWebSecurity
public class LssSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.
          inMemoryAuthentication().
          withUser("user").password("pass").roles("USER");
    }
}
```

3. Resources

- Java Configuration in the Spring Security Reference

Lesson 3: URL Authorization


Java - spring-security-m1-l3/src/main/java/com/baeldung/lss/spring/LssSecurityConfig.java - Spring Tool Suite
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
*LssSecurityConfig.java
String authority
Writable
Smart Insert
27 : 53

```
package com.baeldung.lss.spring;

import org.springframework.beans.factory.annotation.Autowired;

@EnableWebSecurity
public class LssSecurityConfig extends WebSecurityConfigurerAdapter {

    //

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception { // @formatter:off
        auth.
            inMemoryAuthentication().
            withUser("user").password("pass").
            roles("USER");
    } // @formatter:on

    @Override
    protected void configure(HttpSecurity http) throws Exception { // @formatter:off
        http
        .authorizeRequests()
            .anyRequest().authenticated()
            .antMatchers("/delete/**").hasAuthority("ADMIN")
            .and()
        .formLogin().and()
        .httpBasic();
    } // @formatter:on

}
```

1. Main Goal

The goal of this lesson is to explain the basics of **URL Authorization**.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m1-lesson3

If you want to skip and see the complete implementation, feel free to jump ahead and import: m1-lesson4

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

Important: Note that there is a known problem in the video - the general *anyRequest()* and the more specific *.antMatchers("/delete/**")* - should be in the reverse order (the more specific first, the more general last).

Starting from our previous security config, we are going to override the following:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
      .authorizeRequests().anyRequest().authenticated()
      .and().formLogin()
      .and().httpBasic();
}
```

Notice that this is actually the default implementation of this method - which we are going to copy and use as a good starting point.

We can change this by adding some extra configuration for the delete operation:

```
.antMatchers("/delete/**").hasRole("ADMIN")
```

Finally, we're going to look at a few examples using:

- *hasAuthority*
- *hasAnyRole*
- *hasAnyAuthority*

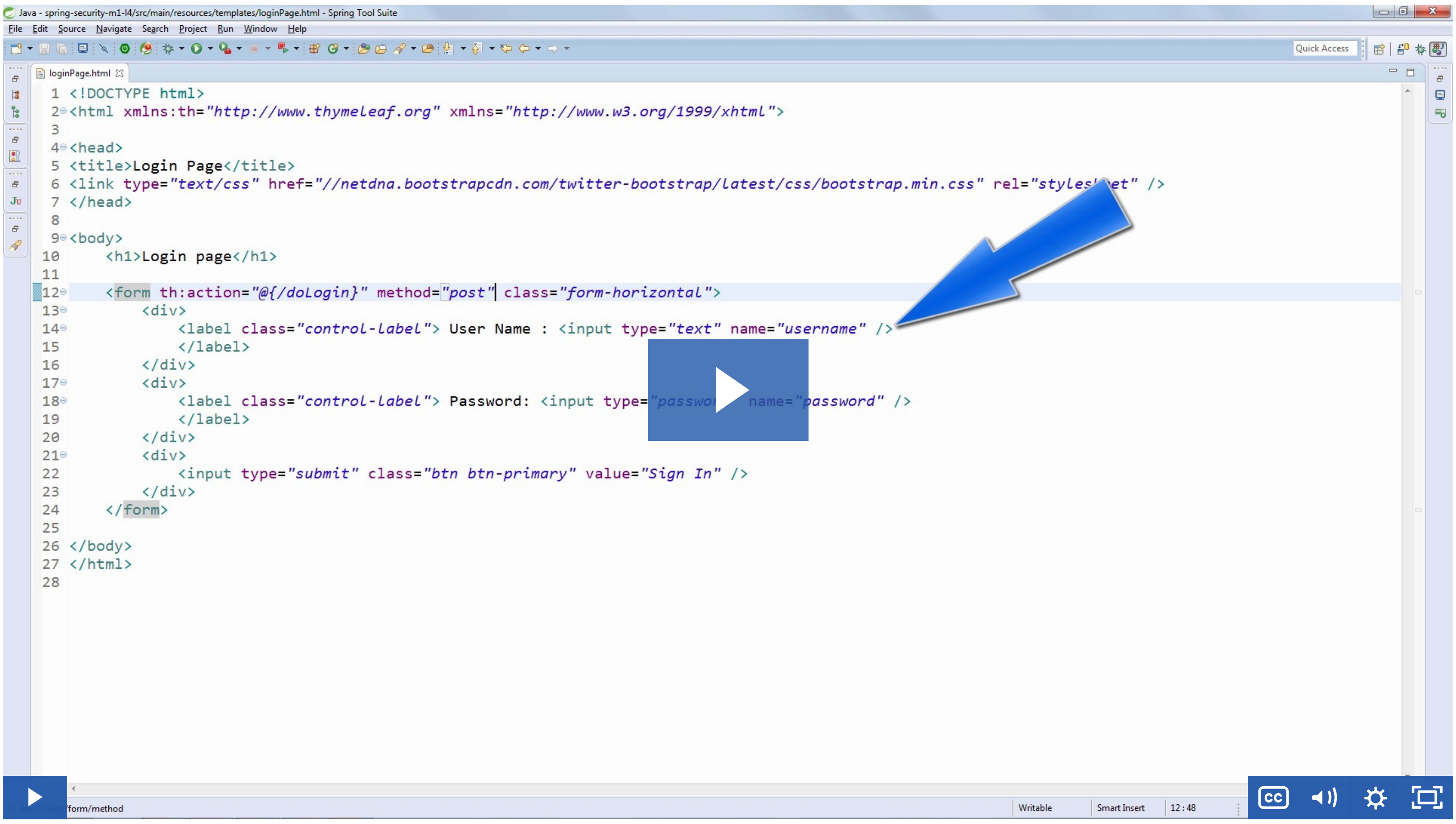
And briefly mention:

- *hasIpAddress*
- *access*
- *anonymous*
- *denyAll, permitAll*
- *fullyAuthenticated, rememberMe*

3. Resources

- Spring Security Reference - Authorization

Lesson 4: Building a Login Form


Java - spring-security-m1-l4/src/main/resources/templates/loginPage.html - Spring Tool Suite
File Edit Source Navigate Search Project Run Window Help
Quick Access
loginPage.html
form/method
Writable
Smart Insert
12 : 48

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>Login Page</title>
<link type="text/css" href="//netdna.bootstrapcdn.com/twitter-bootstrap/latest/css/bootstrap.min.css" rel="styles   et" />
</head>

<body>
    <h1>Login page</h1>

    <form th:action="@{/doLogin}" method="post" class="form-horizontal">
        <div>
            <label class="control-label"> User Name : <input type="text" name="username" />
            </label>
        </div>
        <div>
            <label class="control-label"> Password: <input type="passwo     name="password" />
            </label>
        </div>
        <div>
            <input type="submit" class="btn btn-primary" value="Sign In" />
        </div>
    </form>

</body>
</html>
```

1. Main Goal

The goal of this lesson is to guide you through building and configuring a simple login form for our MVC application.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m1-lesson4

If you want to skip and see the complete implementation, feel free to jump ahead and import: m1-lesson5

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

We are going to configure the login page and replace the default login auto-generated by the framework:

```
.formLogin().loginPage("/login")
```

We are then going to define a custom login action URL:

```
.loginProcessingUrl("/doLogin")
```

The reason we don't want to stick to the default is because the default exposes the fact that we're using Spring Security (which is not ideal)

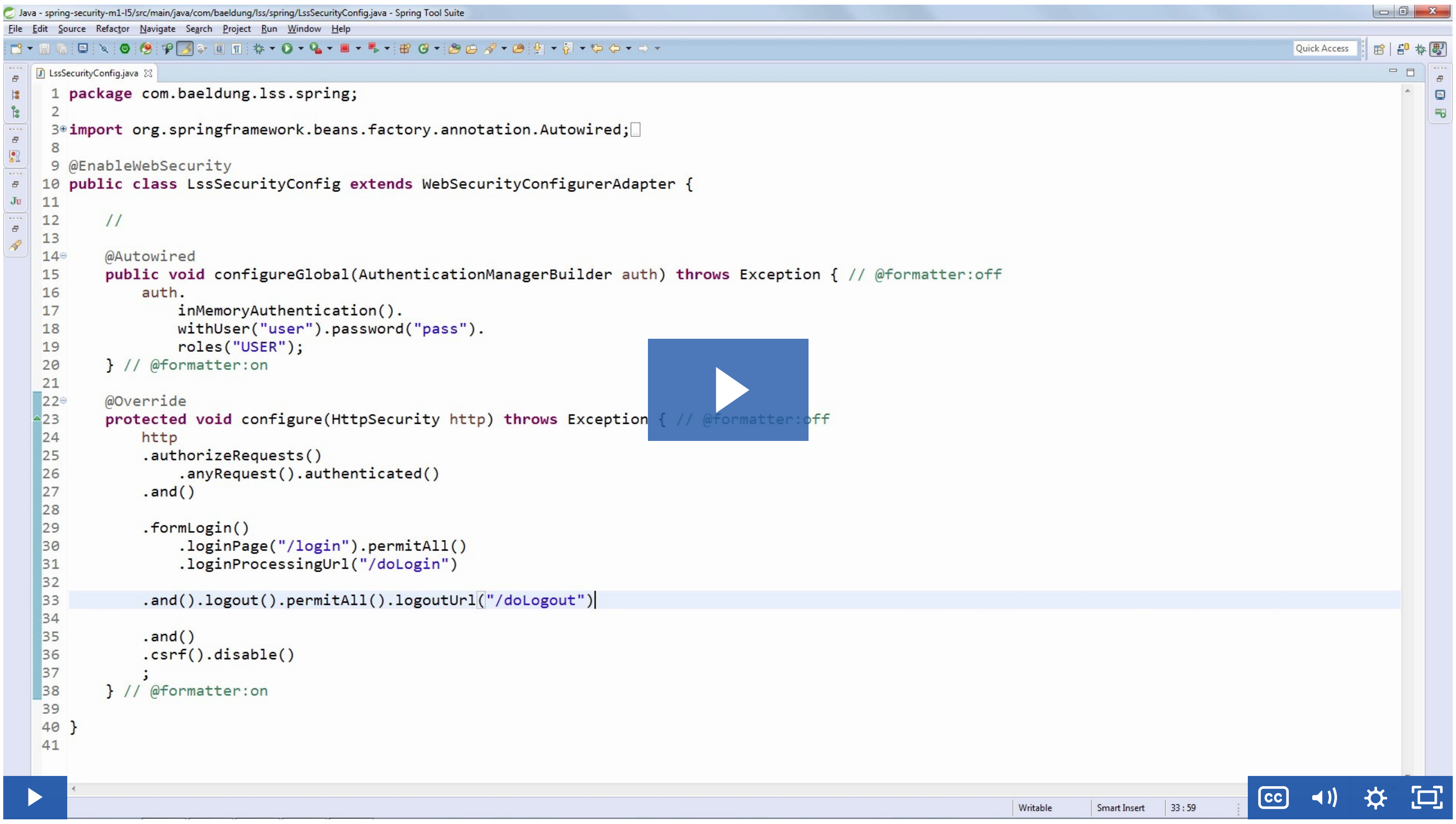
Finally, we are going to create our simple, custom login page with Thymeleaf and implement some basic error handling on it.

3. Resources

- Java Config and Form Login in the Spring Security Reference

- Spring Security Form Login on Baeldung

Lesson 5: Implementing Logout


Java - spring-security-m1-l5/src/main/java/com/baeldung/lss/spring/LssSecurityConfig.java - Spring Tool Suite
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
LssSecurityConfig.java
Writable
Smart Insert
33 : 59

```java
package com.baeldung.lss.spring;

import org.springframework.beans.factory.annotation.Autowired;

@EnableWebSecurity
public class LssSecurityConfig extends WebSecurityConfigurerAdapter {

    //

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception { // @formatter:off
        auth.
            inMemoryAuthentication().
            withUser("user").password("pass").
            roles("USER");
    } // @formatter:on

    @Override
    protected void configure(HttpSecurity http) throws Exception { // @formatter:off
        http
        .authorizeRequests()
            .anyRequest().authenticated()
        .and()

        .formLogin()
            .loginPage("/login").permitAll()
            .loginProcessingUrl("/doLogin")

        .and().logout().permitAll().logoutUrl("/doLogout")

        .and()
        .csrf().disable()
        ;
    } // @formatter:on

}
```

1. Main Goal

The goal of this lesson is to explain how to set up a simple logout flow with Spring Security.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m1-lesson5

If you want to skip and see the complete implementation, feel free to jump ahead and import: m1-lesson6

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

We are going to start by setting up the logout configuration:

```
.logout().permitAll()
```

By default, the logout URL will be */logout* - we can change that:

```
.logoutUrl("/doLogout")
```

We'll then add the actual logout link on the client side of the application:

```
<a th:href="@{/doLogout}" class="menu-right"> Logout </a>
```

If we want, we can be stricter and specify the exact method that's allowed to do logout:

```
.logout().logoutRequestMatcher(new AntPathRequestMatcher("/doLogout", "GET"))
```

Note that, if CSRF is enabled, GET won't work for logging out, only POST.

Also, generally speaking, we should only use POST anyways, since logout is an operation that changes the state of the system.

Finally, we'll add an extra message on the login page:

```
<div th:if="${param.logout}">You have been logged out.</div>
```

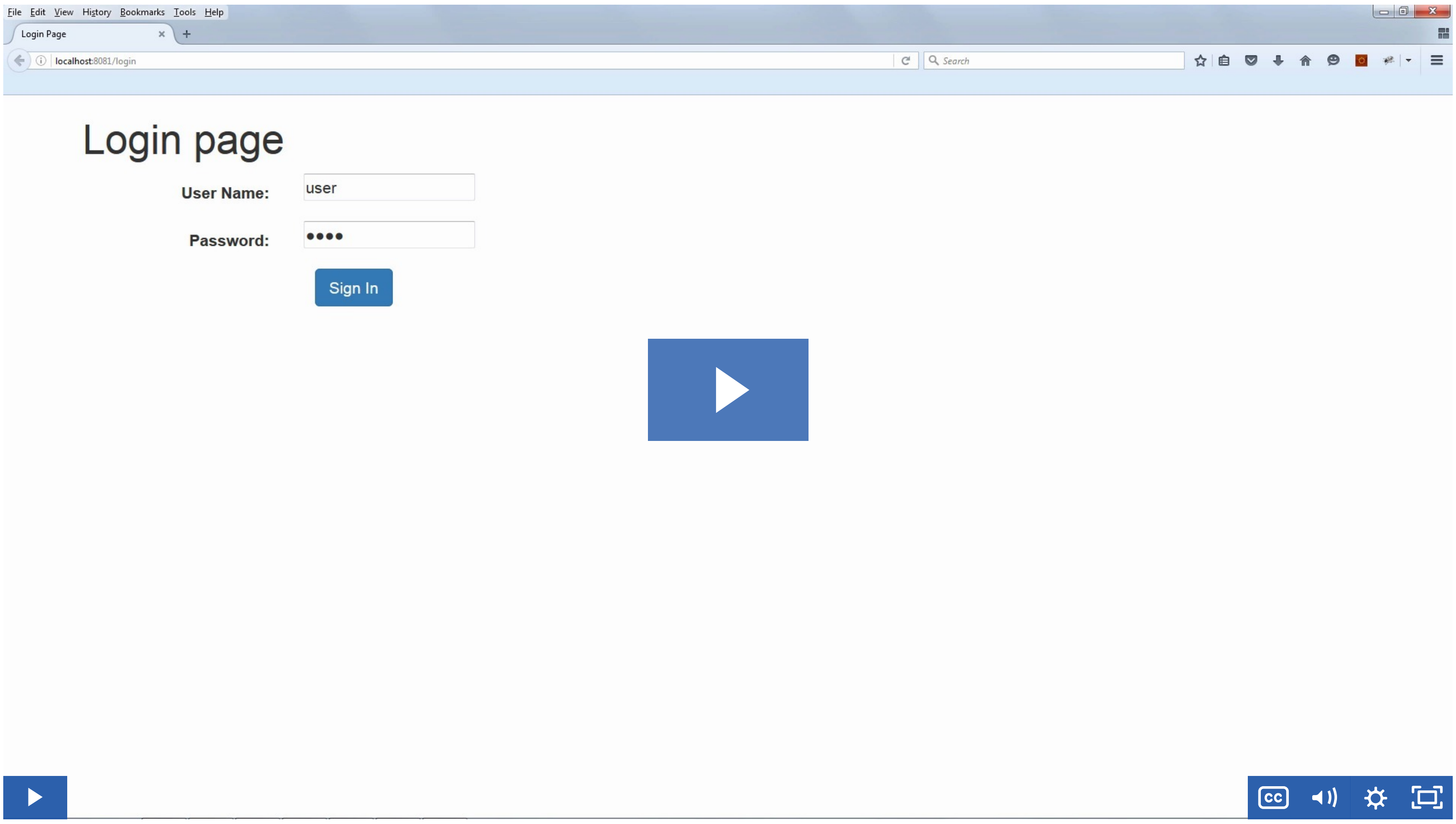
We can also define:

- change if the authentication is cleared on logout: *clearAuthentication*
- mark specific cookies for deletion on logout: *deleteCookies*
- change if the httpsession is invalidated on logout: *invalidateHttpSession*
- change the logout success URL: *logoutSuccessUrl*
- a logout handler or a logout success handler

3. Resources

- Logout in the Spring Security Reference

Lesson 6: Anonymous “Authentication”



1. Main Goal

The goal of this lesson is to explain the content of anonymous authentication and the scenarios where its useful.

2. Lesson Notes

The relevant module you need to import for this lesson is: m1-lesson6

If you want to skip and see the complete implementation, feel free to jump ahead and import: m1-lesson7

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

The concept of "anonymous authentication" is often times a source of confusion.

The underlying issue is that there there is a lot of code that relies on an principal existing, and so - when we're in an anauthenticated context - that logic doesn't work.

A couple of quick example of such logic are logging and audit.

In those cases, instead of having to manually implement null checks and deal with this scenario manually - it's much easier to have an "anonymous" user to fall back on.

3. Resources

- Anonymous Authentication in the Spring Security Reference

Lesson 7: Persistence (Bonus Material To Be Released)

All lessons marked as "bonus material" will be released after the go-live of this Class.