



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Spring Security

Version 2.0

Instructor's Guide



Overview

Especially as of version 2.0, Spring Security does such a good job of encapsulating the most common web security requirements that, for a day at least, we can almost forget about Java, and just see this as a generic tool that requires only a little XML configuration. And so we do: this course breaks down roughly into halves, and in the first of these we do authentication and authorization entirely by configuring Spring Security in an **applicationContext.xml**. Then the “under the hood” chapters confront the actual Java API behind all that XML, and we start doing things that really require Java coding, such as custom user realms and authorization constraints.

The “Chapter 0” on Spring is included for those audiences who don’t know Spring at all coming into the training. We guess that this will not be the common scenario, but Spring isn’t a hard prerequisite for the course. See the timeline section below for more on this.





Timeline

The following breakdowns are approximate, and every class will vary. Also, this course is designed to anticipate two different starting points, with the optional Chapter 0 providing some background on Spring for those with no prior Spring experience. The more common case will be that students do know at least a little Spring, and so our primary timeline is:

Day 1

½ hour	Chapter 0 (just for tools, environment, and setup)
2 hours	Chapter 1
2½ hours	Chapter 2
2 hours, runs to next day	Chapter 3

Day 2

2 hours	Chapter 4
2½ hours	Chapter 5
1½ hours	Chapter 6

If students do need to get familiar with Spring, they will probably not be able to complete the whole course in two days – and the course outline has clear warnings about this, so you shouldn't need to make apologies. In this case you'll likely cover Chapters 0-3 in full, and then apply a lighter touch for 4-6, skipping some labs and maybe skimming some sections:

Day 1

2 hours	Chapter 0
2 hours	Chapter 1
2½ hours	Chapter 2

Day 2

2 hours	Chapter 3
1½ hours	Chapter 4
1½ hours	Chapter 5
1 hour	Chapter 6

Or, you might choose to do one or two of those later chapters in full, while skipping others completely, depending on students' interests.





Tools Deployed with the Lab Software

This course's software requires separate setups of JDK 6, the Crimson text editor, and/or Eclipse; otherwise it is self-contained, as the lab installer sets up not only the lab software but necessary tools – these will all be found in directories under **c:/Capstone/Tools**:

- **Ant1.6.** We use Ant extensively for build and deploy operations.
- **JPA1.0.** This includes the standard JPA JAR and the TopLink Essentials provider; these JARs are copied into the Tomcat classpath early in the course.
- **JSTL1.2.** The JARs and TLDs for JSTL tags used in many of the course-exercise JSPs are found here.
- **MySQL5.0.** This is our relational database server. The JDBC driver JAR is copied into the Tomcat classpath early in the course.
- **Spring2.5.** This includes the Spring JARs and documentation.
- **SpringSecurity2.0.** This includes the Spring Security JARs and documentation.
- **Tomcat6.0.** This is our target web server for deployment and testing.
- **Crimson**, a freeware text editor, in **Crimson3.70**. This is made available so that we're sure students have a decent code editor on hand, though they can certainly use their own favorite text editor or IDE. Also, Crimson is primed with support from a second tool, the **XML Validator**, to make a nice little XML editor for working with **web.xml** and **applicationContext.xml** files. Hit F9 to save the current file and run an XML parse to check syntax; hit F10 to save and validate against referenced schema (though this requires either Internet access or a change to the schema location at the top of the file).





Ant Build Process

Though most students will be happy to let the **ant** command take care of things, some will want to understand the inner workings here a little better. Each web project in the course has its own **build.xml** and **build.properties** files; these rely on the master environment variable **CC_MODULE** to import targets and properties in a central directory, **%CC_MODULE%\Ant**. Information here defines a routine for building a web application; we won't document this in detail here, but rather summarize the interesting tasks.

In a typical project, Ant will build and deploy a web application to a Tomcat server:

- Undeploy any existing version of the application, by deleting the WAR from Tomcat's **webapps** directory
- Clean out and create a **build** directory, which becomes the root of a web application
- Compile Java source files to **build\classes**
- Create a staging area for assembly of a web application
- Copy resources from **docroot** to that area – this may include HTML and JSP, the **web.xml** and **applicationContext.xml** files, message resources, and custom tags – and copy compiled Java classes to its **WEB-INF/classes** folder
- Copy JARs into **WEB-INF/lib**: JSF, JSTL, Spring Security, etc.
- Build a WAR file from the assembled tree
- Deploy the WAR by copying it to Tomcat's **webapps** directory

DoEverything.bat

Over the years, we've found that this environment is reliable and reasonably easy to work with, once it's set up. However the setup has been a bit thorny in many classrooms, especially where students are not used to Windows and/or DOS. To simplify the setup process, we've included a set of batch files in the root lab directory, and most of these are mentioned in the coursebook at the appropriate times. Not described in the coursebook, but potentially even simpler for students, is a master script **DoEverything.bat**. You might want to try this out and recommend it to students as a one shot replacement for doing **SetEnvironment**, **StartTomcat**, **StartMySQL**, and the stop/**SetUpTomcat**/start sequence that are called for at various points in the book. This one command will set up and start both servers and leave one console with the right environment for Ant builds.





Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse Galileo. See the course Setup Guide for download URLs and setup instructions, and the coursebook introduces this workspace at the end of the first chapter, as an alternative to the DOS/Ant-based approach actually detailed in the book.

Instructors, use this package on your own initiative and at your own risk. You should have significant experience yourself with Eclipse web development before using the overlay package in the classroom. The workspace and projects have been tested lightly with the course but are not part of the standard product.

That said, this overlay should save a good deal of work for those who wish to use Eclipse instead of the text editor and command-line tools that are standard for the course. See the documentation inside the workspace itself for general notes on usage: when you first open the workspace you'll see a **ReadMe.txt** that directs you to deeper, HTML-based documentation. Be prepared to walk students through the first few exercises in Eclipse; the notes in this file are for experienced Eclipse users, and will not be clear to many students on their own.

See also the module notes file in each workspace – e.g. **SpringSecurity Module Notes.html** – for specific matters related to using Eclipse WTP with this course.

Again, Capstone Courseware can only offer complete technical support on the standard course, and while we hope this overlay is convenient, it is not as thoroughly tested as the core lab image at this time. If a given exercise is giving trouble, please be certain to build and run it from the command line, using the SDK tools as prescribed in the student guide, before contacting Capstone.





Teaching Notes

Chapter 0

This is the optional primer on basic Spring beans configuration discussed in the overview and timeline sections above. For those unfamiliar with Spring, this chapter is meant to illustrate just the basic concepts and the configuration techniques they will need in typical Spring Security development: the container, the XML configuration file, dependency injection, and what a web context is. We tack on a quick illustration of JDBC persistence since that plays a large role in Spring Security, even though it doesn't bring any new Spring configuration techniques into play.

Depending on how well students seem to be taking in this material, you might add a sort of quiz exercise to the end: challenge students to diagram the bean relationships that they see in **Wholesale/Web** and **Wholesale/Persistent**, in the style of the diagram given for **Wholesale/Beans**. This should help sharpen their comprehension of the dependency-injection mechanism and generally of how Spring works. Do warn them in advance that there will be some head-scratchers in there! because these applications use auto-wiring in a couple places, and we don't get into that in the chapter text.

For those who already know Spring well, note that you should still cover this chapter minimally, because it has a few notes on tools, environment, and setup sprinkled through it along with some examples. The examples won't be interesting but they still serve to prove that the environment is sound for the rest of the course work, so it's worth building and testing them. You can either take students through the setup steps as described in the coursebook, or just have them run **DoEverything** as mentioned earlier in this document.

Chapter 1

So this is really the first chapter of the course, and hence the unusual chapter-numbering scheme. This chapter gives an overview of Spring Security, shows one working example, and then takes students through the process of adding Spring Security to an application from scratch by way of a lab exercise. This also helps to emphasize the point that almost all features of Spring Security work just fine for non-Spring web applications.





Chapter 2

Here we get into details of configuring various authentication features. And there are a lot of features! One challenge in this chapter will be just sorting these out, and holding off questions about some of them while you cover the first few. We proceed as methodically as we can from HTTP BASIC to form-based login, remember-me, anonymous authentication, and logout features, persistent user realms via JDBC, hashed passwords in the database, and finally channel security and session-fixation protection.

Lab 2A uses a custom JSP tag to provide an interesting glimpse of some Spring Security internals, by showing session attributes used by the filter chain and supporting beans.

Note that Health application versions starting with the answer code to Lab 2B will only work with hashed passwords; so if you have any need to jump forward or back to illustrate something with this application, and you jump over the line between **Step2** and **Step3** versions, remember to **run PrimeWithData** appropriately: with the **hash** argument for later versions and without for **Step2**. (Actually it's irrelevant for **Step1** since at that point we're not using JDBC to read passwords at all.)

Chapter 3

We jump almost immediately into a lab in this chapter, because we've been stiff-arming the whole question of role-based authorization all along while we focused on authentication, and so a lot of the concepts have already leaked out in earlier chapters (or maybe were already understood by students who, for example, already know servlets-style security). The Health application provides a good canvas on which to paint for this purpose, as it has a handful of functions each of which has different authorization characteristics.

The section on “programmatic security” sets us up for some more work in the Health case-study application, and fits nicely with the role-based presentation piece at the end of the chapter. But really most of the programmatic authorization work that we do will occur later, in Chapter 5 – and will take the form of a custom voter, which is a neater way to do real authorization work. The **isUserInRole** and **isGrantedAuthority** calls explored in Lab 3B are not authorization logic in the grant/deny sense, but rather implement dynamic page-flow and work-flow choices. I like to tell students that, but for forceful browsing, it shouldn't be necessary for most applications to deny access to a user with a 403; the sorts of role-based presentation and page-flow logic shown here should guide the user only to choices they're allowed to make in the first place.



Chapter 4

There is a marked shift at this point in the course. Up to this point we've been learning about Spring Security features and, for the most part, applying them via XML configuration. Lab 3B starts to cross the line, and in this chapter we're clearly taking a different approach, which is Java-centric and in which Spring Security starts to look more like a code library than a plug-and-configure tool. So, a lot more UML, more Java coding, and more direct use of the Spring beans vocabulary instead of Spring Security "namespace configuration."

In fact we go right for the jugular with the first couple examples: first exposing and then (almost) duplicating a typical security configuration as a graph of Spring beans in memory. One caution on these examples is that it's easy to get in too deep. The point of the "long-hand" example especially is to give a stronger sense that all these objects exist and of how they are organized; it is not to kick off an exhaustive study of each and every class and relationship that's involved. The rest of this chapter, and the next one, will focus attention on a handful of classes that really are interesting for practical application.

The Love Is Blind example shows a custom user realm, more or less in isolation from other features, as we need to read an XML file (but not a Spring beans file) to get our user records. It's simple enough, even if the means of reading this data – XSLT – may not be familiar to students. The point is just that we can use any tool or implementation strategy to derive user information, when we implement **UserDetailsService** ourselves. (And the strategy will usually be some sort of adapter or connector, whether it's XSLT or DOM or JAAS.)

Then the lab shows a further technique, which is taking advantage of how Spring Security makes it easy to find a thread-affinite context that, in turn, provides a path to a **UserDetails** object – by connecting additional user-specific data to that object. By a more or less textbook Adapter strategy, we can put whatever useful information about the user we like more or less at our fingertips, and then read it when needed from anywhere in the request-handling process.

A good extra exercise after this chapter is to have students draw up a CRC analysis – class, responsibility, and collaborator – for the major actors in the authentication system. This often helps to sharpen understanding of who exactly does what. It could be done individually or in small groups, and then everyone could compare notes and pull together a consensus diagram on the whiteboard. Classes to include would probably be:

```
AuthenticationManager  
AuthenticationProvider  
UserDetailsService  
SecurityContextHolder  
SecurityContext  
Authentication
```





Chapter 5

Though the authorization mechanism is a smaller part of the overall filter chain than authentication, it may be harder to grasp at first. There is a fair amount of indirection here, from filter to manager to voters; and the role played by config attributes isn't intuitive for most. So this takes some 'splainin', and the "free and fair elections" example is meant to clarify these things. I recommend carrying out the extra "experiments" at the end of this example, unless everyone in the room just obviously gets the picture already. In my experience it's at that point that the light really comes on.

The question of how to implement appointment-reminder protection prior to the lab exercises might produce an interesting discussion, and it's a good one to let students mull for a few minutes before proceeding to the two "answers" given in the book.

An interesting point to discuss after the lab: how do we feel about the fact that our new technique exposes the patient's health-plan ID as part of the reminder URL? And if that's not a great idea, what could we do instead? A few possible fixes: (1) keep a map from reminder ID to patient in the database, with obvious performance disadvantages, (2) parse the reminder (or dig it out of the database, which is more likely in the real world) to see whose data it is, and again this will be expensive, or (3) hash the plan ID before putting it in the URL. I like (3) best, but there are other schools of thought, and this should be a good discussion.

The end of this chapter is another good spot for a CRC exercise, this time focused on authorization actors:

```
FilterSecurityInterceptor  
ObjectDefinitionSource  
AccessDecisionManager  
AccessDecisionVoter  
ConfigAttribute  
ConfigAttributeDefinition
```





Chapter 6

After a couple of more challenging chapters, we finish up with something pretty easy, which is method authorization – plus domain-object authorization, which is trickier but also mostly out of scope and kept to a quick introduction. Method authorization requires no Java coding, though it does cross a different line, in that applications must be using Spring to manage their secure objects in order to apply this feature. (The same of course is true of domain-object security.) But, with this condition met, method authorization is pretty clean, and it's mostly a matter of deciding between the four (!) different ways that you can configure it. The big divide here is between annotations and external XML configuration, and there are the usual tradeoffs though we don't go into a big discussion of this in the coursebook.





Revision History

Version 2.0 is the initial release of the course.

Troubleshooting

If you run into any trouble with code exercises, the first and best thing to do is to double-check that the classroom machines have been set up precisely according to the course setup guide. Especially, the wrong version of a tool can cause significant problems; don't wander off-book in this way unless absolutely sure you can support the software that you prefer and that we haven't tested. Check environment variable settings carefully, too; these are the cause of a great many classroom glitches.

Below are some specific pitfalls that have come up in previous offerings of the course:

- Remember that the default constraint under `<http>` is no constraint at all! So a URL pattern that is too narrow or just misses because of a mistype will result in a wave-'em-on-in: no authentication required.
- Mistyped role names are another common culprit that Spring Security can't catch when it validates the context configuration. Remember that (as described towards the end of the course) roles must begin with "ROLE_" by default.
- Students may try listing multiple URLs or patterns separated by commas; it would be lovely if this worked, but it doesn't, and none of the URLs listed will be protected.
- Most other typos and low-level errors in the configuration will be caught and announced in the server console, either by the Spring container (referring to a non-existent data-source bean, e.g.) or by Spring Security (not specifying a user details service, using a config attribute without providing a voter that understands it). It's a good idea to keep the server console, or at least the tail of it, visible while building and deploying, so that these sorts of errors will draw your eye easily when they occur.





Errata

The following issues have arisen in the classroom since the latest release, and will be fixed on our next release:

- The Eclipse module notes should say the same thing about testing **Wholesale/Persistent** that they say about testing **Health/Step1**: that is, that it's necessary to do an **ant build** and **run PrimeWithData** from DOS, and then refresh the Eclipse project, before proceeding.
- In the final versions of the Health application (answers to alternate Labs 5A and 5B), when visiting the appointment reminder, one logs in and then sees a blank screen. Reloading from the browser at that point does show the reminder page, but it doesn't come up the first time as it should. This hasn't been reproduced yet and may be browser-specific.

Feedback

We very much appreciate whatever feedback we can get on our courseware – especially from the instructor's perspective. Naturally, the more specific, the better, and we strongly encourage you to make notes on issues you may encounter in the classroom, whether they're typos, missing files, or suggestions for clearer language to explain a concept. We can't guarantee that we'll act on every suggestion, but we're aggressive about stamping out problems and try to be highly responsive. Hopefully this means that when you give us good feedback, you get a better course the next time you need to teach it.

Please direct all courseware feedback to

Will Provost
Capstone Courseware
<mailto:provost@capcourse.com>
877-227-2477

For anyone who's interested, we have a very informal defect-tracking system, based in Excel spreadsheets with columns to capture defect location, nature, status, and author feedback. Ultimately, feedback goes into these sheets, so if you want a template, we'll be happy to provide one, to facilitate the reporting process.

