A Custom AccessDecisionVoter

00:01: Welcome to lesson four of Module Nine; A Custom Access Decision Voter. This is going to be a practical lesson as well as a scenario. We are going to build a decision voter based on some custom logic and based on some common practical scenarios. So let's get right into it. We're going to start changing the default access decision management implementation. Remember that by default, we're using the affirmative based implementation here, and we're going to go with the more stricter, unanimous best implementation just to make sure that all voters either abstain or vote positive. We are then going to create our custom access decision voter and we are going to wire that in and start using it. We are going to debug through exactly what's happening and we're going to understand why we need the custom voter in this case.

00:52: Okay. So let's get to the code. We're starting off here with our standard security configuration and the first thing we're going to do, is we're going to switch from the affirmative implementation of our access decision manager to this unanimous implementation just to make sure that we are stricter in the way that the framework is evaluating access. So, let's first define the new bean.

[pause]

01:37: Okay. So we have now defined the unanimous-based access decision manager and notice that we've set it up with three voters. We have the RoleVoter, we have the AuthenticatedVoter, and we have the WebExpressionVoter. It's important to either define the voters that the old the implementation had or at least to define the voters based on what type of authorization you're planning to define. So for example, if you're planning to use expressions, you need the WebExpressionVoter. If you're planning to use role-based authorization, obviously you need a RoleVoter, and so on.

02:09: All right. So with the bean defined, we need to actually wire that bean into our security configuration. It's of course not enough to just define the bean, we need to actually use the bean. And there we have it. This is why the Java configuration API of Spring Security is so powerful. Okay. So the next step is to secure something and remember that we have our secured page that we always use when we're testing authorization. So we're gonna use that and we're going to very simply secure that page with a has a role expression, which is going to ask for admin.

[pause]

02:51: So pretty straight forward. And, of course, nothing we haven't seen before. And now we're ready to start up the application and to debug through exactly what's happening. Now that we have this new unanimous-based access decision manager. So that's what we're going to do next. Okay. So with the system running, we are now logged in to the application with our user, which of course does not have role admin. And we're now going to simply access the secured page. Okay. So we now have the web request. We can look at the object that is being secured and we can see that it's the filter invocation and not any sort of method invocation, of course. And we now have the new access decision manager.

03:33: So first of all, let's make sure that this is actually the new manager. And here we go. We can see the new unanimous-based implementation of our access decision manager. And we can also, if we look into it, we can also see the voters and we see the voters that we have just defined the RoleVoter, the AuthenticationVoter and the WebExpressionVoter. So everything is correctly set up. Let's now step into the code and go through a decide process. Okay. So this is, of course, slightly different from the affirmative-based implementation, because obviously the unanimous-based implementation requires that none of the voters vote negative.

04:15: So let's go through the three voters and let's see how they vote. So the RoleVoter has abstained, and of course, that is simply because we have used an expression to secure our page. So the RoleVoter really doesn't know how to interpret that expression, so it will not vote either positive or negative, it will simply abstain. Okay. The second voter is the AuthenticatedVoter. This should abstain as well, because it has nothing to do with our expression and there we go, this has abstained as well, so we can now move forward. Let's put a breakpoint here. We can now move forward to the main voter that we really care about, which is the ExpressionVoter. Remember that we have used an expression to secure the page, so that's really the voter that's going to determine access.

05:06: Okay. So we have the right voter here. The WebExpressionVoter and we are going to see exactly how this votes. Now, remember that the expression was asking for role admin and we are not logged in with an admin. We're logged in with a standard user, so the result here should be minus 1. It should be access denied. And here we go. We are getting the access denied and so, the unanimous-based implementation here will deny access.

05:32: All right. Let's go forward and let's get back to the browser and see that we are actually denied access. And of course, we see the forbidden and we see the 403 status code. So we are correctly denied access. All right, so the next step now is to create our custom voter. But before we do that, we need to discuss the scenario that we're aiming to implement here. So what is the scenario? Well, very simply put, we need to lock users out in real time. So remember that the user in Spring Security does have a lock flag, however the problem with that is that when a user needs to be locked out in an enterprise or a high security environment, we need that user to be locked out immediately. And unfortunately if that user has an already established session, then they will not be locked out immediately. Even if we update their persisted user and we switch the locked flag to true, it's still not enough because the currently authenticated principle, the current session will still have the old enabled flag because that get's established when the user logs in.

06:38: And if we're really talking about restricting the access of a user, even a few seconds of access are super important, let alone the 60 minutes that the typical session will last for. If the user really needs to be locked out, then they definitely need to be locked out immediately and not after almost an hour, because there is a lot of potential damage that that user can do in that hour.

07:00: Now there are a few ways to implement this, this is definitely not the only way, you can, for example, use JMX and connect and remove the session of the user, that may or may not be easy to do depending on the topology of the system, but the solution we're going to talk about here of course is a voter-based solution. We're simply going to implement a voter that looks at the locked users in the system. So if we add the user to a locked user list, to a cache basically, then our new

custom voter should simply restrict access in real time and so the user will be locked out immediately. All right, so let's get started. So the first thing we need to do is we need to implement our new custom voter. So let's do that right here in this class. All right so pretty straight forward up until this point, of course we need to make sure that it does support any configuration attribute because this is not really restricted for the time being, and we're now left with the vote implementation itself. But before we get to this vote implementation, we'll have to set up the very simple storage for locked users. Now this is of course just going to be an in-memory type of storage but in a production system, this should really be a cache.

08:12: All right, so let's get this implemented and then let's get back to the vote implementation. And there we have it. We're simply going to use a set here, we are going to keep track of the locked user names, and we're going to verify if a user is locked or not via this is locked API. So very simple and again not something you should go into production with, but this is essentially just to allow us to lock a user. All right, let's close this down and let's get back to the vote. So with the new functionality available that now allows us to check if a user is locked or not, let's now write our custom vote logic.

[pause]

09:02: And that's it. A very simple vote logic, but one that is going to in real time restrict the access of a newly locked user. All right so now the voter implementation is done, let's actually initialize the voter and let's wire that in into the access decision manager, and then let's run through a real scenario. Okay, so the voter is in and we can now start the system. Okay, so the system is started and we're now going to log in with admin, we're not going to log in with user, because we want to actually have access to the secured page, so we're going to use admin.

09:38: And now of course we'll access the secured page and we have a break point in our new voter logic. So let's access the page, here's the break point. It will of course verify that the account is not locked and it will grant access. So at this point because the account is not yet locked, we are able to see the secured page just fine. Now the interesting thing is, of course, locking the account and seeing that we're no longer able to access the page. So let's refresh here and we're refreshing just so that we trigger the breakpoint again, and now we're going to actually in real time lock the account. So let's see exactly how we're gonna do that. We are going to open the display here, this is a view that will allow us to interact with the running system, and we're simply going to lock the account by using the API of locked users. And that's pretty much it. Now the admin account is locked, we actually executed this lock admin and now admin is actually locked. So for example if we check the result of this is locked call now that we locked the admin, we can see that the result is actually true.

10:53: And now because the admin account is locked, of course our vote implementation should return access denied. Here we go. We are now seeing the access denied and let's run and let's switch to the browser. All right. So as expected we're getting the forbidden. So that is how a custom voter works, and that is of course a real time locking scenario. All right, hope you're excited, see you in the next one.