A Custom Authentication Provider

00:00: Welcome to the First Lesson of Module Eight: A Custom Authentication Provider. So in the previous module, we saw how the authentication flow works and we saw how the authentication providers are getting used by the authentication manager and how these are ultimately the drivers of the whole authentication process.

00:20: In this lesson, we're going to role out our own authentication provider. So first, we're going to implement this custom authentication provider. We're then going to wire that into our security configuration and basically replace the old one, the DaoAuthenticationProvider that gets bootstrapped by the framework. And of course, we're going to end the lesson by actually authenticating, following that flow, and understanding exactly what's going on. Alright, so let's get right into the code.

00:49: Alright, so we are starting in the security configuration. And of course, the starting point is we do not have a custom authentication provider. The system is using the DaoAuthenticationProvider and the provider manager is our main default authentication manager. We saw both of these and we saw how they worked in the previous module. When we did a walk through of the authentication flow, we saw exactly how those wire in together and how they work. So now we're going to role our own authentication provider and we're going to try to replace the DaoAuthenticationProvider that Spring Security sets up for us.

01:26: Now an authentication provider is not huge but it's still a fair amount of code to type out. So instead of typing that out, I'm going to simply paste it, and we're going to of course discuss the implementation. So here we go. We have now defined this custom authentication provider, and the main thing we care about here is this authentication method right here and the contract that the authenticate method needs to adhere too. The contract party is important because this provider is part of the broader Spring Security configuration. So, of course, this provider needs to behave in such a way that the framework can work with it. So what we need to do of course is we need to check out the Javadoc of the interface. And we need to make sure that we're following the contract that is specified in that Javadoc. So let's first have a look at the Javadoc.

02:25: And the contract looks very simple. First of all, if we are able to authenticate, we will of course return a fully authenticated object. If this particular provider does not support the authentication object, so basically if we cannot even attempt authentication, then we're going to return null. And if we can attempt authentication however authentication fails, then we're going to throw the authentication exception. Now when we're talking about the authentication exception, it's important to understand that this is actually a base class of an entire hierarchy of exceptions. So if we have a look at the authentication exception here, first of all we can see that this is an abstract class, so right off the bat it's clear that we need to work with the children of this exception, not with the exception itself.

03:14: And now, let's have a look at the hierarchy. And you can immediately see just how many children, just how many individual exceptions are actually built on top of this base class, a lot of very specific and very semantically rich type of exceptions. So in our case, we are only going to work with one or two. But in a production implementation, if you're doing a custom authentication provider, it's important to be as specific as possible. So it is important to look through these exceptions and essentially figure out which one fits the use case best and throw

that. So definitely, don't just throw the bad credentials exception here. This is one of them and this is of course a common one. But there are many other very specific exceptions that make more sense in other scenarios.

04:02: Alright, so let's get back to the implementation, and let's have a look at our particular and very simple implementation, but that is still following the contract of the interface. So first of all, we're looking at, do we support this particular type of authentication? And of course, this implementation in particular is just an example implementation, which is why this is simply returning false. But the point here is of course to use your specific logic and decide if you can handle authentication or not.

04:31: So for example, if your custom authentication provider is just one provider in a list of providers, then you may want to look at some details of the authentication request here. And you may want to decide, should I or should I not support authentication here? However, if this provider is the only provider, then this implementation can simply return true because this is the only provider. So the answer to the question is of course, yes, we are going to attempt authentication. In this case, let's actually switch this to returning true. And with that in mind, this provider will actually try to authenticate.

05:08: Now moving forward, the next step is to actually attempt authentication and then to interpret the results. And in this case, again this is an example authentication provider, so there's no actual third-party system that we're trying to work against. But of course in a real work scenario where we would try to authenticate against some third-party system, what we're going to do here is if the authentication is successful we're simply going to return the token back to the authentication manager. And if the authentication was unsuccessful, we are going to throw this authentication exception. And getting back to the hierarchy of exceptions, this is a simple scenario. The more information about the failure we have from the third-party system, the better our exception handling can be.

05:56: So for example in this implementation, we assumed that the third-party system will just say yes or no because this is just a boolean. However, in a real world scenario, this is not likely the case. It's very likely that the third-party system will return a lot more data. So if the authentication fails, we'll have more data. So we will be able to throw a more specific exception as to why the authentication actually failed.

06:22: Alright, so that is our custom authentication provider, a very simple one but one that is following the contract and that should behave correctly. Now, the other note is that we'll have to make this small change and return true in the implementation just to make sure that we can actually log in.

06:40: Okay, so now that the provider is defined, let's wire that into our security configuration here, and let's start using it. So as you can see, much like everything else in our security configuration here, setting this up is not very difficult. Let's now start up the system, try to authenticate, and check out the exact providers to see our new custom provider being used in the authentication flow. Okay, so with the system running, we are going to authenticate and we're going to trigger a break point in our custom authentication provider. And then, we're going to step

all the way through the authentication flow and see exactly what happens. And here's the break point being fired. And immediately, because the break point is in our implementation here, we can see that our implementation is actually in use.

07:37: But before going through the implementation, let's have a look at the authentication manager and let's understand the new structure of that manager. Let's see what providers we have in that manager. Okay, so here's this manager. First of all, notice that this is the parent manager because it does not have a parent of its own. And then, let's look at the providers. So the only provider here is of course our custom authentication provider which has, as we expected, replaced the default DaoAuthenticationProvider. So what that means is that we now have a single provider and it's our custom one.

08:14: Okay, with that in mind, let's go back to our authenticate implementation here and let's step all the way through that. So we see the name and the password, we check if we support this particular type of authentication, which is the username and password authentication token, and of course we do support that. Then we're going to attempt to authenticate and in our case, as you've seen, we're simply returning true and we are done. We are now returning the token back into the authentication manager.

08:47: Okay, so we're back in the filter. We'll now finish the debug session, and we'll go back to the application. And there we have it. We are now logged in as expected. Alright, hope you're excited. See you in the next one.