A Simple Two-Factor Implementation with a Soft Token

00:00: Welcome to Lesson 1 of Module 14: A Simple Two-Factor Implementation with a Soft Token. So, in this module, we're going to start exploring two-factor authentication. And in the next lessons, we are going to look at other ways to obtain the token.

00:17: Alright, so let's see what the agenda is for this lesson. The very first thing that we'll discuss is what is two-factor authentication? Even though most modern systems are using two-factor authentication, it's still really important to take a step back and understand the ecosystem. We are then going to have a quick look over the available options to implement 2FA, and more specifically the available options to get the token. And then we are going to settle on a soft token solution for this particular lesson, and we're going to look at how to set up that solution with Spring Security.

00:52: Alright, so let's talk about what two-factor authentication is. And, if you've been following the security landscape, even marginally, you definitely saw that most modern systems have adopted 2FA in recent years. The underlying principle of two-factor authentication is authenticating the user, using something that the user knows, which of course usually is their password, as well as something that the user has. So, very, very simply put, if the credentials of the user somehow get compromised, the attacker will still not be able to access the system just by using the credentials. They would also need to have access to this secondary factor, this physical object that the user is going to use to generate this extra token. So let's have a look at what that physical object usually is.

01:43: Okay, so what does the user have? What can the user use during the authentication process? So the first answer here, and probably the easiest, is their phone. The user can use their phone to get this token by using a variety of solutions. One of those solutions is of course SMS. SMS has the advantage of working with pretty much any phone out there, so the user won't necessarily need to have a smartphone. Now, depending on the target user of your application, that may be nice to have, or it may be a critical thing, because of course not all target markets will have access to a smartphone.

02:22: The second way that the user can use their phone to obtain this token is using an authenticator mobile application. So, this is the soft token solution that we're going to look at in this particular lesson, but note that this does require that the user actually has a smartphone. So, this is definitely important to understand before you start implementing a soft token solution.

02:45: The next option is a hardware token. And there are, of course, many hardware token implementations available. This is basically where the user will get an actual hardware device that they're going to use to generate new tokens.

03:00: And finally, this is something that is worth discussing at a theoretical level. Even though it's probably unlikely that you're going to use this in a practical real-world implementation, it's still an interesting thing to discuss because it expands this definition of something that the user has. The finger print is, of course, something that is fundamentally connected to the user. So, you could theoretically use a finger print scanner to generate this token.

03:25: Now, we already discussed why we generally do need 2FA. But very simply put two-factor authentication is really a powerful way to add this extra layer of security into a system. Because [03:38] sort of attacks against 2FA itself, which of course do exist, 2FA can be really effective and is also becoming widely adopted in the market. So in reality, if you have a system that holds any sort of critical data, it is really important to secure that system with a 2FA implementation.

03:56: Alright, let's now move right into the implementation. Okay, so before we get started with the implementation, let's first talk about the authentication process, and let's have a quick look at the functionality that we did touch on, but we didn't really discuss in depth. So, remember that during the authentication process, when the UsernamePasswordAuthenticationFilter is just starting the whole process, but before the flow is delegating to the authentication manager and to the authentication providers, there is this small section right here that will allow us to wire in our own logic, and basically add some extra information on that authentication request. So, that's the very first thing that we're going to do. We're going to add this extra information so that when the manager and the provider start running they are going to have access to that extra information.

04:50: Alright, so, let's do that. We are going to define this new bean, which is going to be an authentication details source. That is going to be the new source for our new extra details. So, as you can see, the bean itself is very, very simplistic. It gets the raw HTTP request, and it's going to basically return this custom implementation of the web authentication details. So, let's go ahead and create that class as well. And that is basically it. All we need here is we need to extract this code out of the request and make this available later on in the process. So all we're doing is we're extracting that code and we're storing it.

05:48: So if we go back to the filter, and if we look at this set details implementation here, you're going to see exactly how that works. So the details source gets called. And so in our case, our custom details source gets called with a request. It builds that simple wrapper for the code and that gets set on the authentication request. And let's have a look here. This is basically an object. Here we go. So this is an actual object, meaning we can store pretty much any kind of data we need in there.

06:22: Alright, so let's now get back to the filter and let's have a look at what's going to happen. So after this is set on the request, the authentication flow basically goes forward. Now what's missing here is of course wiring in this new bean and making sure that our security configuration is actually using it. So let's do that.

06:50: We've injected our custom details source and we've wired that in into the security configuration. So now if we do an authentication, this details source is going to run. Okay, so what's the next step here? Well, now that we'll have this extra information on that authentication request, let's actually start using that extra information. So with that in mind, we're going to roll out a new authentication provider where we are going to try to use this new verification token. So let's roll that out.

07:29: Now before we look at this one, a quick side note is that whenever you're doing a new provider, it's usually a good practice and a good idea to start with an existing implementation. So, for example, you could start with the DaoAuthenticationProvider. This is not entirely flexible, so most of the time you won't actually be able to extend it, but it's still a great place to start. So even if you cannot extend this one, you can copy it and basically add your own logic on top of it. Of course, that's only if this particular implementation fits. And in our case, it would certainly fit because we are still using Username and Password, we're just adding an extra check, an extra piece of logic.

08:14: Alright, so let's get back to our implementation, which is of course much simpler. So as you can see, this particular implementation is not very complicated. We don't need to go to production with it, we only need to make sure that first of all, it's doing the standard check, the Username and the Password check, and then it's going to allow us to plug in extra logic. And this is where we are going to do the check. If you have a look here, you can see that we're already extracting the code. So that was the whole point of having the code available. We want it to be able to extract it here and then we want it to be able to use it. But of course, before we are able to do that, we'll have to make some necessary preparations. And more specifically, we'll need to first add a library that's going to help us with this check. So let's do that now.

09:10: This simple library is going to give us the tools we need to check that the token that the user send is actually the right token. Now a quick note here is that our verification code is using an algorithm and a technique called Time-Based One-Time Password. So this is how these tokens are generated using this particular technique. But we will talk about the generation later on. For the time being, we are assuming that the user was able to generate the code, and of course he's sending that code with the authentication request. Alright, so now let's get back to our implementation here and let's add the actual check. Let's actually check that the code is valid.

09:50: And so you can immediately see just how easy it is to do this check when you're working with a proper type of library. So the check is very simple. We are first instantiating this simple API that's going to allow us to do the verification. We are then actually verifying here. If the code doesn't match, we're going to throw this bad credentials exception out of Spring Security. And if there is any other problem, we are going to throw the exact same exception. Now what's the missing piece here? Well, obviously, this is not yet compiling. The user, when they register, will get a secret and will expose this secret here. For the time being, we haven't done that yet. But let's still create the field in preparation for that logic.

10:44: Alright, so we've created a field and now the provider fully compiles. So next and before we move on, let's wire in the provider and so let's make sure that it's getting used. Alright, so let's do that.

11:02: And here we go. We have now wired the provider into the security configuration. So now everything is wired in and ready to go. Alright, so now it's time for the live demo and it's also time to see the frontend part of this entire flow. We're skipping over the implementation of that for a couple of reasons. First of all, it's mostly front-end code and second of all it is of course fully available on the code base. Alright, so let's go through this entire process from registration to log in, and let's see exactly what's going on. So the very first thing we're going to do is we're going to register. And as you can immediately see, this first step is exactly as it was before. So we're not seeing anything extra. Alright, so let's register.

11:49: And so now after hitting Register, we are prompted by this new page. So this is the point where the Google Authenticator app is going to come into play. So what we need to do here and what I'm doing right now, is I'm picking up my phone, I'm starting the Google Authenticator app, and I am scanning this barcode. Alright, so the barcode is scanned, and I'm done. I can now navigate away from this page to the login page. Now, let's log in. First off, let's log in without the code.

12:24: So the credentials are correct, but notice that I'm not adding any sort of code. And of course, that doesn't work. We're getting this message, "Invalid username, password or verification code." And it is good practice, not to actually say, which of those is the root cause of the problem. We don't want to get potential attackers any sort of specific information as to what went wrong. So we shouldn't say "Invalid username," for example, because that will mean that an attacker now has a way to verify if a username exist or not. And so, that opens up a variety of potential attacks. Similarly, we don't want to say, for example, that the verification code was the problem. Now, obviously in our case, that wasn't the case. But even in general, you don't want to specifically say what the problem was. You want to have this kind of generic message that says something went wrong, but doesn't give any extra detail. Alright, so now let's log in and let's actually try out an invalid code.

13:32: So we are passing in a code, but it's an invalid one. And as you can see, we're getting the exact same result. So coming back to how much information you're actually providing the user, the application is not going to specifically point to the verification code being wrong. Alright, so let's now actually try to authenticate with a proper correct verification code.

13:58: And for that, of course, we'll have to go back to the application. And we'll have to paste in this time-sensitive code that gets generated by the application. So in our case right now, it's 787343.

14:12: And here we go. We are logged in and we are past authentication. So this is our first simple implementation of 2FA with a soft token. Alright, hope you're excited. See you in the next one.