Defining & Managing Users

Created	@2023년 1월 15일 오후 10:13
Class	Spring Security
Type	Lecture
Materials	
Reviewed	☐
Number	

- : Now inside this lecture, let's try to talk in detail

about the 'UserDetails' interface and its implementation.
Like we're discussing in the previous lecture,
by using the 'UserDetails' interface
and its implementation classes, we can represent the details
of the end user, like the username, password, authorities
of the end user can be stored inside the object
of the 'UserDetails' implementation classes.
So that's why these interface
and its implementation classes are used inside all
the above classes and interfaces,
like if you take 'InMemoryUserDetailsManager',
'JdbcUserDetailsManager', or 'UserDetailsService'.
So if you take any of these interface or class
which you can see above the 'UserDetails' arrow,
all those classes and interfaces,
they use 'UserDetails' inside their method return type

or inside their business logic that these classes hold.
So that's why it is very important to understand
what exactly is present inside this 'UserDetails' interface
and its implementation classes.
So let's try to understand those details now.
So like you can see here,
this is the 'UserDetails' interface.
Since it is an interface,
it'll have all the abstract methods.
The methods that are present inside this interface
are first one is 'getAuthorities',
which holds the list of authorities or roles
of the end user, so that we can use them
to implement authorization or role-based access mechanism
which we are going to discuss in the coming lectures.
And 'getPassword', 'getUsername',
they will return the password and username of the end user.
And after these basic details of the 'UserDetails',
it also supports few other methods which will help us
to identify whether the user account is expired,
whether the user account is locked
or whether the user credentials are expired
or if the user account is enabled or disabled.
So for all these kind of scenarios,
we also have methods which help us to understand the account
to details of my end user.
So if you see here, so these are all abstract methods,
you can simply read the method comments that you have here.
Like if you see, the very first one returns

the authorities granted to the user
and the second one will return
the password used to authenticate the user.
And the third one is which is 'getUsername' returns
the username of the end user.
Now coming to the remaining methods which we have
like 'isAccountNonExpired'.
So whenever this value is true, that means the user account
is valid, it is not expired, whereas false indicates that
the user account is no longer valid
because the account is expired.
So the same applies for 'isAccountNonLocked',
'isCredentialsNonExpired' and 'isEnabled'.
The Spring Security team, they have not stopped
by just defining an interface.
They also give some sample implementation of these interface
which we can use inside our projects and inside our day
to day development.
So like you can see here, they're saying go
and refer the 'User' class for a reference implementation.
If you want, we can write our own implementation.
No one is stopping from that.
But if you are good with the default implementation provided
inside the 'User', then there is no meaning
for us to write our own implementation of 'UserDetails'.
So it's up to you whether you want to use 'User'
or if you want
to write your own 'UserDetails' implementation class.
So inside this 'User' class,

all those methods that we saw inside
the 'UserDetails' interface,
the same might have overrided inside this class.
So like you can see there is a 'getPassword' method
which simply returns the password value
but here you may have a question
like how this password value is going to populate it.
So for that, if you go and check the constructors
of this 'User' class, anyone can create an object
of this class by using these two constructors,
one is without passing any Boolean values,
you just pass a username, password and authorities
and remaining all Boolean values,
they'll just populate true, true, true
which indicates that the account is not expired,
not locked and it is not disabled.
Whereas if you also want to send these Boolean values,
you also have one more constructor like you can see.
This is the another constructor that we have here.
So basically whoever is creating this user object,
they have to pass these details and accordingly
whenever someone is trying to access this 'getPassword',
'getUsername', so these details will be shared
to them based upon the method's invocation.
And one more interesting point that I want
to highlight here is if you check the list
of methods available inside the 'UserDetails'
and 'User' class, there is no setter method available,
you only have the getter methods

that means you can only read the properties present
inside this object.
Once the object is created with the help of constructor,
there is no possibility for us to override
the values using the setter methods.
That's because since this is related to the security,
once the object is created with the help of constructor
after the successful authentication,
we don't want anyone to override the values
like username, password, authorities.
So that's why there is no possibility
for anyone to override these values.
So that's one more good design pattern that is constructed
by the Spring Security team.
So if you check the method that we have inside
the 'InMemoryUserDetailsManager',
there is a method called 'LoadUserByUsername'
which will be invoked by the 'AuthenticationProvider'.
So here you can see based upon the username,
we are trying to load the details from the map.
This map might have created during the start up
of my application, based upon the 'UserDetails'
that I have configured
and stored inside the memory of the web application.
So if there is some user with the given name inside this map
then definitely those details will be populated.
And the same will be used to create an object
of user by invoking its constructor.
Once this object of user is created,

the same is being returned.
So don't worry about the typecasting issues
because this 'UserDetails'
and the 'User' have parent and child relationship.
So that's why there should not be any issues here.
So now here you may have a question
if you look at the internal flow. Inside the internal flow,
I said all the authentication details like
whether the successful authentication is happened or not
and the 'UserDetails' like username,
all those details will be stored inside
the authentication object. So that's what I said.
And you can also see at the step two,
everything is populated into the authentication object
and the same will be stored inside
the security context post the successful authentication.
So now here you may have a question
like why there is an authentication object,
why there is an 'UserDetails' interface and 'User' class,
what's that relationship between them?
For the same, you can check this slide.
So if you look at this slide,
this guy also have a similar question
like why do we have two separate ways to store
the login user details?
The very first one is 'UserDetails' and 'User' class.
And inside this class and interface, we have 'getPassword',
'getUsername', 'getAuthorities' and so and so methods.
So this 'UserDetails' and 'User',

we will use whenever we are trying
to load the details of the user from the storage system
like database memory of the application,
especially inside the interfaces and implementation classes
of 'UserDetailsService' and 'UserDetailsManager'.
Once the user details are loaded from the database,
then definitely these details will be shared back
to the 'AuthenticationProviders'.
So inside the 'AuthenticationProviders',
once the authentication is successful,
these 'AuthenticationProviders', they're responsible
to transform all that information
along with the successful authentication details
into the authentication object data type.
So that's why you can see a note here
in the down of the left hand side.
So authentication is a return type
in all the scenarios where we are trying to determine
if the authentication is successful or not.
So where do we identify that,
whether the authentication is successful or not?
Inside our 'AuthenticationProviders'.
So beyond 'AuthenticationProviders' like
inside the provider manager and inside the filters,
we always deal with the authentication object
because using authentication object,
we can store only the details related
to the authentication like username, password, authorities
whether the authentication is successful or not.

But we don't have to store the details
like whether the account is expired,
whether the account is disabled or enabled
because only if these conditions satisfied,
then only the authentication will be successful.
So that's why there is no reason
for me to carry again those user details
to the internal components of the Spring Security framework.
So that's where my 'AuthenticationProvider'
is going to do the magic
of converting these user details that we received
from the 'UserDetailsService' and 'UserDetailsManager'
into the object of type authentication.
And of course one of the implementation
of authentication that we have
is 'UsernamePasswordAuthenticationToken'.
So this authentication interface
in turn extend principal interface
which is from the Java library.
So 'Principal' interface simply represents what is the name
of an entity, name of an individual or the login id.
So from the 'Principal',
this 'Authentication' interface inherit a method
with the name 'getPrincipal' and inside 'Authentication',
we have other methods like 'getName', 'getAuthorities'.
We can quickly validate this inside the code.
But before that I just wanted to make sure,
you are very, very clear about the 'UserDetails'
and the 'Authentication',

why we are representing 'UserDetails' in two separate ways.
So this slide is the answer for you.
You can always refer the same.
So now let's try to explore the 'Authentication' interface.
So if I open this 'Authentication' interface,
it extends the principal.
So this principal is coming from the Java library.
It has only one method which is 'getName'.
So previously I was telling 'getPrincipal', but I was wrong.
So the only method that we have inside the principal
is 'getName'. Now if you go and check the 'Authentication',
inside this we have methods like 'getPrincipal',
'getDetails', 'getAuthorities', 'getCredentials',
'isAuthenticated, 'setAuthenticated'.
So 'isAuthenticated' is used by the framework to understand
whether the given user is successfully authenticated or not.
And 'setAuthenticated' is used by the framework
to set the value of 'isAuthenticated'.
Whenever the successful authentication is completed
then this method will be used to set the true value,
otherwise a false value will be set.
So if you go and check the 'InMemoryUserDetailsManager',
here we are loading the details
of the user from the storage system.
That's why the return type is 'UserDetails'.
Whereas if you go and check the 'DaoAuthenticationProvider'
where we are trying to authenticate the user,
if you scroll down, once the user details are populated
inside this authenticate method, you can see

at the end when the authentication is successful,
we are just invoking
a method called 'createSuccessAuthentication'.
So this method, what it is going to do is,
it is going to populate all the required details
from the 'UserDetails' to the 'Authentication'
and the same will be returned back.
Once this method is invoked,
you can see there is a return statement from here
and the method signature also how the authentication
as a method return type.
So this also confirms our discussion.
So I hope you are very clear
about the relationship between the 'UserDetails'
and the 'Authentication' because these are very important.
Otherwise, you'll get easily confused
for all the discussions that we are going to do
in the coming lectures.