

user request -> authentication filter -> authentication manager -> authentication provider -> user details service

||

password encoder

1. end user sends req to our REST app, auth filter intercepts this request. It is

servlet filter class which will check if the user has authentication.

2. If not, the request is passed to auth mgr to check if the user details are valid.

3. auth mgr uses auth provider where login logic is there.

4. auth provider will use user details service to retrieve pwd.

5. passcode encoder is used to encrypt pwds and store.

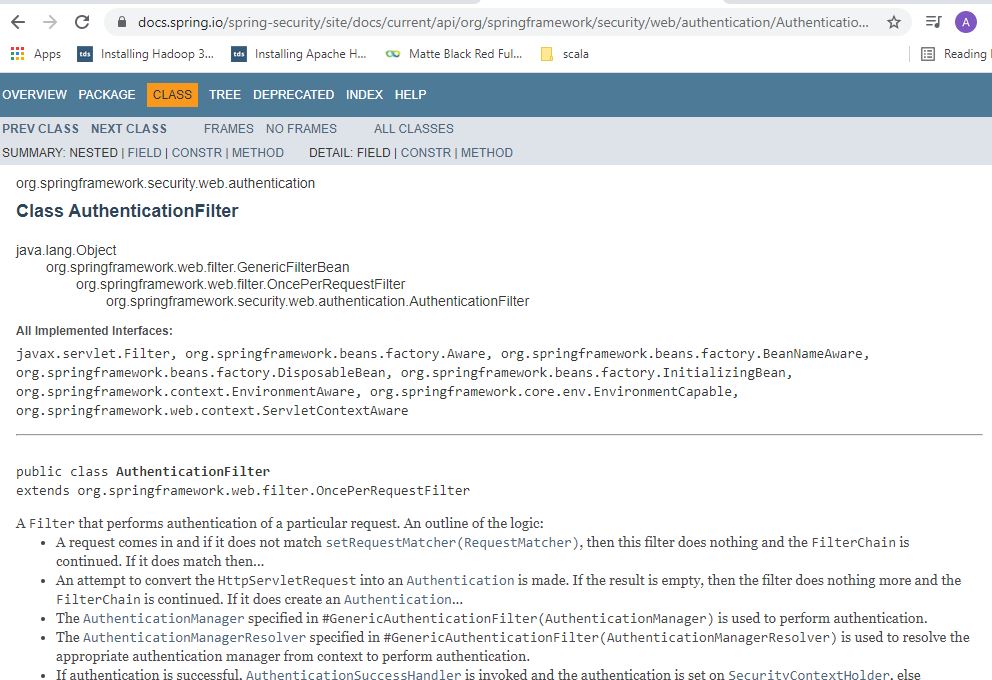
6. if the details match with user details service, auth filter will

use authenticationSuccessHandler and stores authentication info in the user

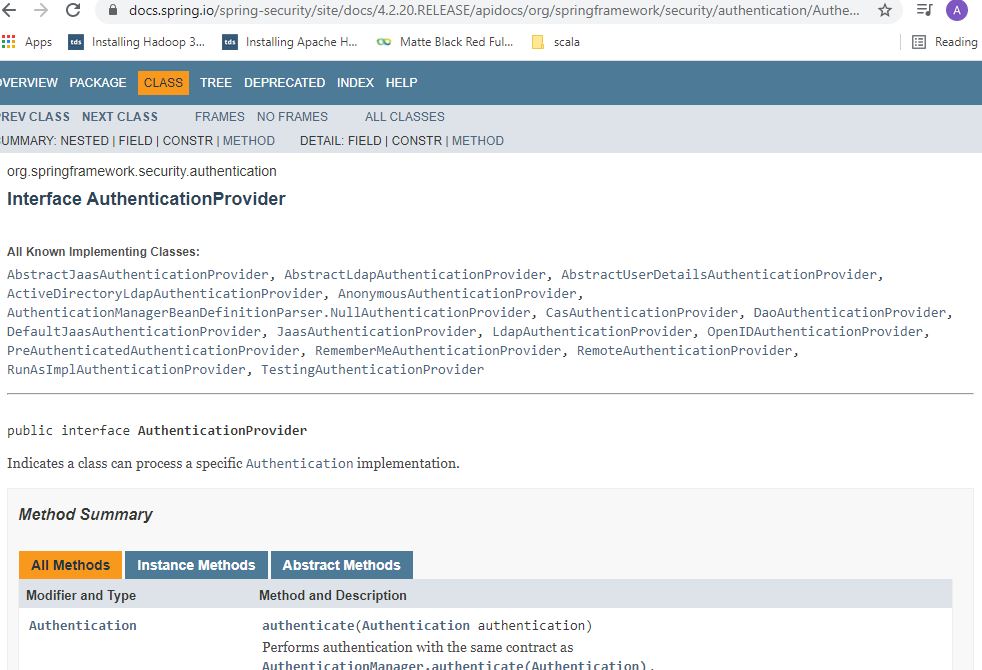
entity in security context.

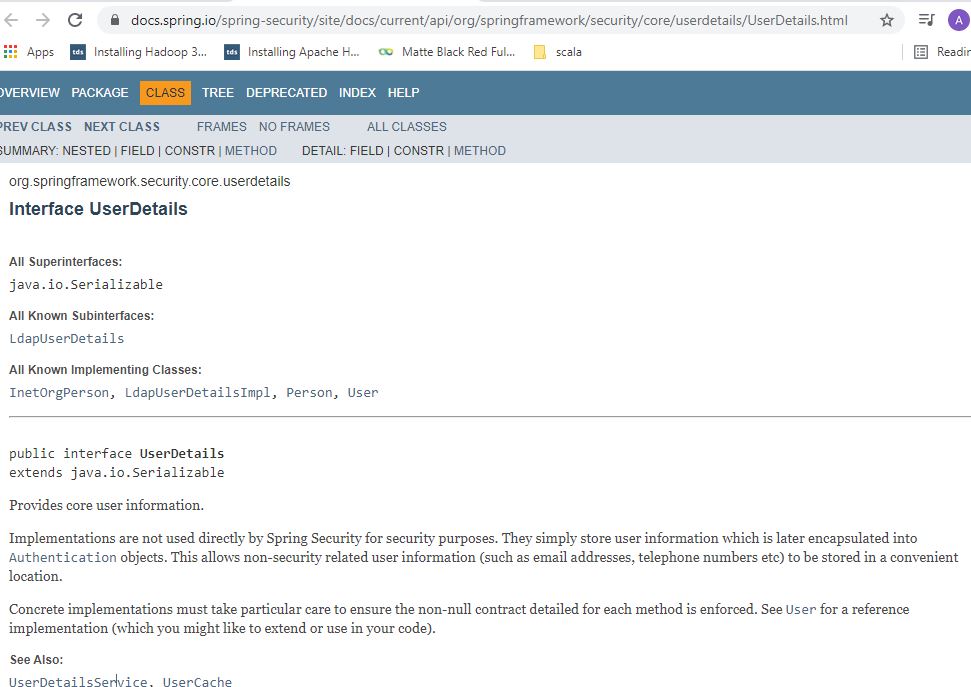
7. in case of failure, authenticationFailureHandler will be sent by auth filter

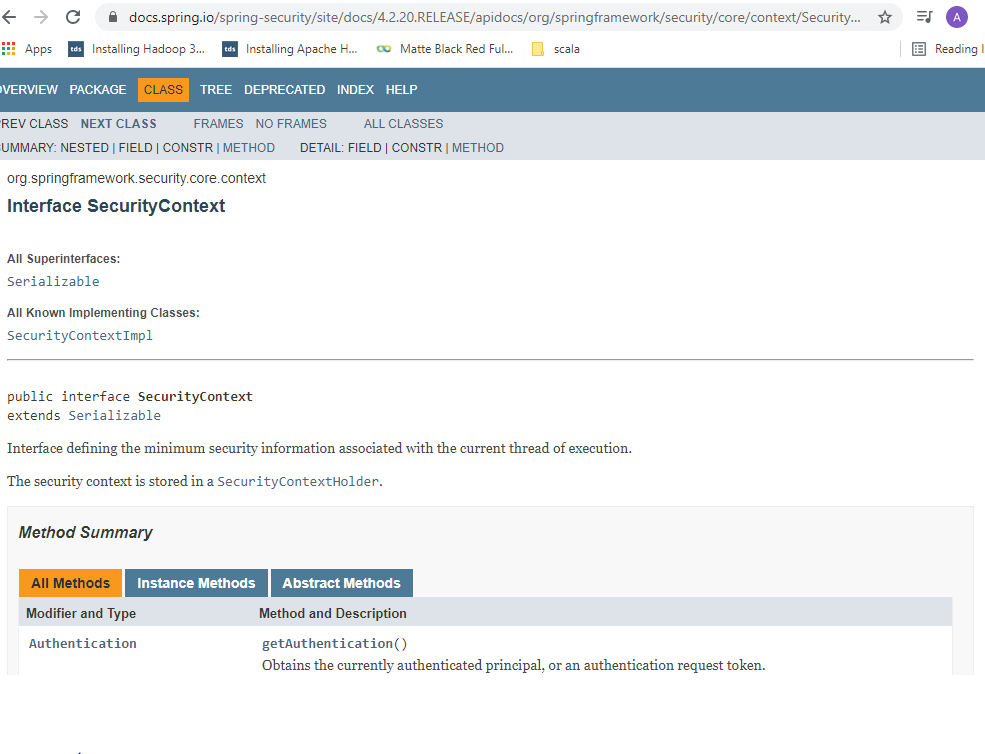
Check for Spring Authentication Filter class in google



Spring Authentication provider interface







**Important note on cookies and basic auth:**

In Basic auth, once you provide the username and pwd, it will get authenticated and send jsessionid in cookies. For subsequent requests, this jsessionid is passed and hence even if you change headers username and pwd, it will work as the Authentication filter in spring uses the jsessionid to validate request.

**OAuth2**

4 workflows/ grant types:

* **Authorization code:** (E.g. Clear Tax uses google authorization server. So client app ClearTax is pre-registered with Google and has a unique client-id, also it consists of redirect-url and client-secret. Authorization server will generate a token and send it to client app).
* **Password:** (The only difference with Authorization code approach is that user inputs username and password on client app page instead of Google page. The client app will send these details along with client-id and client-secret which it obtained when it pre-registered with Google. Disadvantage is that client-app is aware about client username on Google).
* **Client credentials:** (Here user is not in the picture. It is for one app having access to another client application. Typically used in single sign-on micro-services architecture).
* **Refresh token:** (For the above 3 approaches, tokens that are granted by Auth server have a life time. Once token expire, client-app will ask the user to again login to Google. In Refresh token method, Auth server will also return a refresh token. Client token will use refresh token and ask Auth server to send another token. So user does not have to login again to Google).

After Spring boot 2.4.0 version, the OAuth2 dependency has been removed by Spring framework and they want to merge it with spring.security package. Currently it is in spring cloud package.

JWT

So far we have been exchanging plain string based tokens when the client requested for a token the authorisation server

was written in plain string based tokens.

There are disadvantages of this approach is that when the token is received, the client sends that

token to the resource server to get that data.

The resource server has to validate this token again by sending a request to the authorization server or

through a shared database.

And then it has to fetch the user details like the user authorities and all that, again from the authorization

server or through a shared database to validate the authorization of the user.

How much access does that user have

All that is overhead.

Again, the communication between the resource server and the authorization.

Server that is where JWT comes in.

JWT is it Jason based Token Syntax or Standard?

A JWT token will have three parts a header.

Which is, Json, as you can see here, see here, it has the algorithm and it says type is JWT, that

is the header portion, the payload will have the entire user information see that the user name the

scopes, the expiry of the token, then the authorities of that user as well.

This will prevent the resource server in making this call, once the token is issued to the client, the client,

sends that token to the resource server, the resource server need not validate or at least not fetch the

user details because the token already has the user details.

But how does the resource server know that this token is a valid token?

There comes the third portion of the JWT, the signature.

Every JWT token has a signature as well, so it has the header, it has the payload and then it has

the signature.

The signature is calculated by an algorithm which you see in the header here.

We can choose any algorithm, any signature algorithm.

RS 256 is a cryptographic algorithm that calculates a signature, which is nothing but a hash like value

on this payload.

So it uses this payload, calculates the hash based on a private key so we can use either asymmetric

or symmetric keys.

If we use asymmetric keys, then the authorization server will generate will generate public and private

keys.

The authorisation server will use the private key to sign the token and then the resource server will

have the public key.

The authorisation server will share the public key with all the resource servers.

The resource servers will use the public to verify the signature that comes in that token.

So it need not make this additional network call to the authorization server to fetch the user details,

not to verify that token.

That's the beauty of JWT.

It has the user details with all the role information and all that that the resource server can use.

And also it can check for the validity by looking at the signature that comes in.

So JWT token is divided into three sections.

What you see coming back is this on the left, you will not see this Json format because this whole

thing will be base64 encoded.

When you run the test from postmen, you will see the following.

What you see on the left here, the first portion you see before the dot.

This is decoded into this when we do the base 64 decoding it has the algorithm and that type, the second

portion, what are you seeing here?

The color pinkish, that is the body portion.

This guy here, which has a user detail , the last portion which you see here in light blue, that

is the signature calculated the signature we can use a symmetric algorithm which uses the asymmetric

keys.

The private and public keys or symmetric meaning when the authorization server calculates the signature

using this algorithm, this algorithm will use a private key and the same private key will be shared

with the resource.

Server the resource server will use the private key to validate the signature that is symmetric, both using

the same key instead of public and private keys.

**Java Keytool**

Private and public keys are stored in Keystore file, which is password protected.

Public keys can be exported in certificates using keytool.