Cryptography:

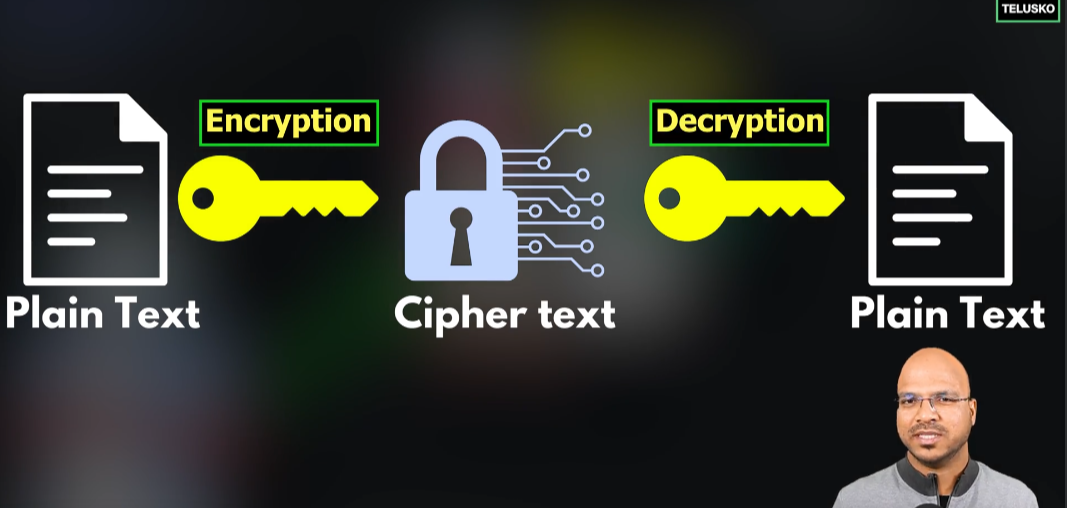
Cryptography is nothing but securing datas while transferring, by encrypting and decrypting the data.

A screenshot of a computer

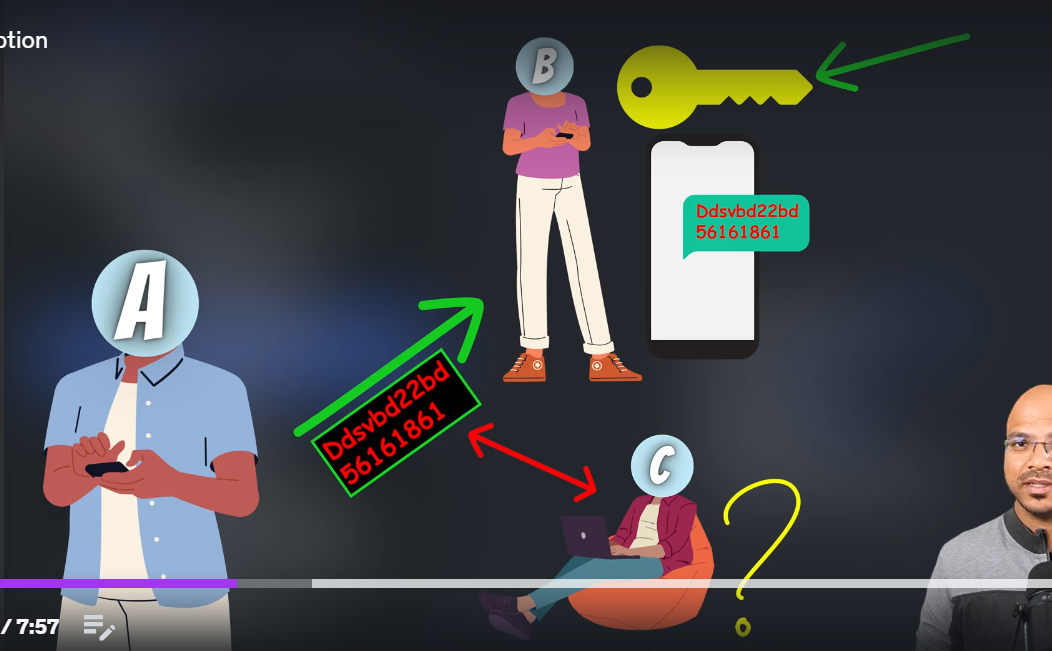
Description automatically generated

Encryption: Converting plain text to ciper text.

Decryption: Converting cipher text to plain text.



Here, A is sending a data by encrypting it using a key. This key will be shared with B so that, only B can see the data. If any attacker like C tries to see the data, he cant view or modify it without that key.



Two types of Key:

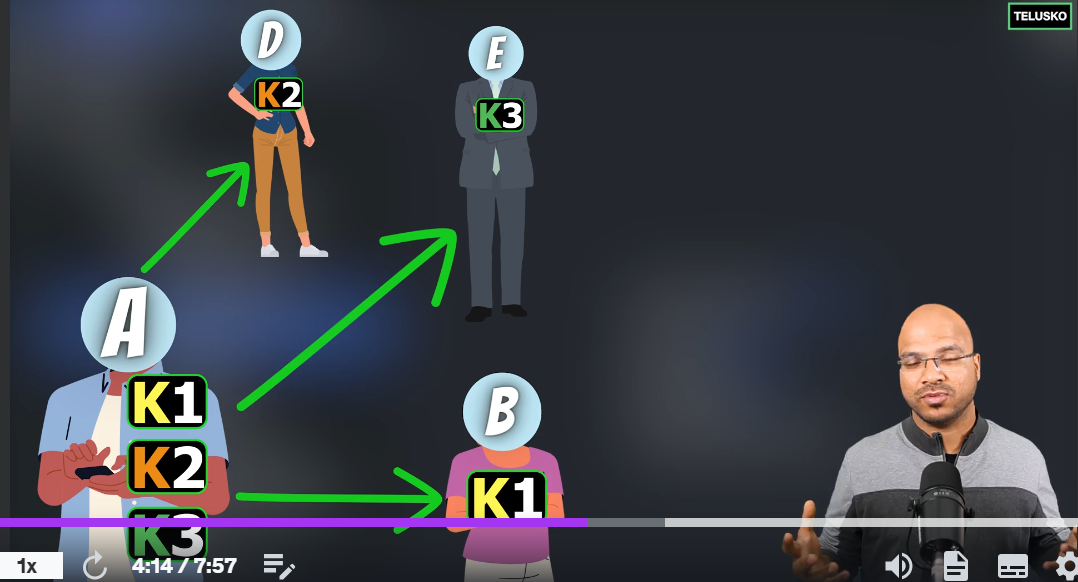
1. Symmetric key:

The same key which A used to encrypt should be used by B to decrypt it. And the key needs to be shared with B beforehand.

Problems:

(1)But if the attacker hacks the key quite easily.

(2)Also if there are multiple users, then it will be difficult to handle all the keys. For example, see below image,



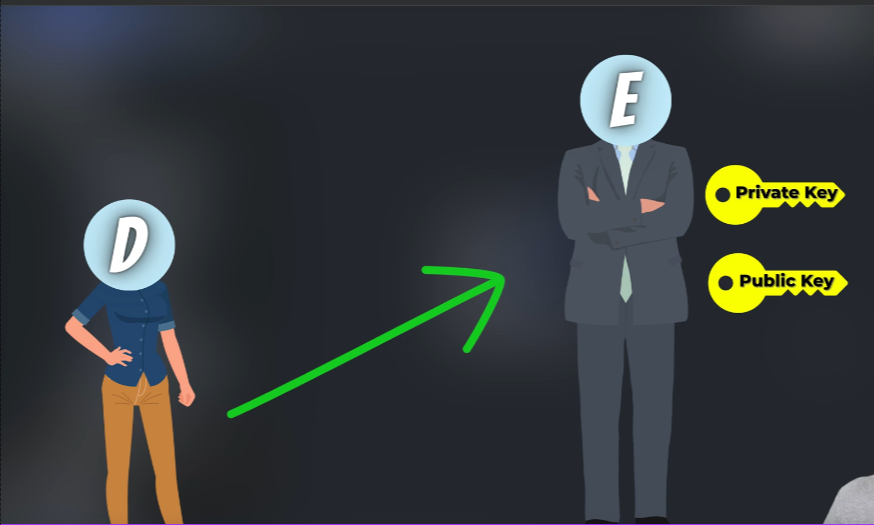
1. Asymmetric key:

To solve the problems of Symmetric key, asymmetric key is used. In asymmetric key, every user has 2 keys. (1)Private key and (2)Public key.

(1)Private key: Private key of one user can’t be accessed using another User.

(2)Public key: public key is a common key, where everyone can access and see the data.

Every user has private and public key. Lets say, A has these two keys and if A ecrypt the data using A’s public key, then that data can be only decrypted using A’s private key.



Here, D uses E’s public key to encrypt the data and send it. And E uses E’s private key to decrypt that data. Even if a attacker named F tries to access this data, F can’t attack it with its private key.

Algorithms:

There are different algorithms for both Symmetric and Asymmetric keys:

Symmetric keys algorithm:

1. AES(Advanced Encryption Standard)
2. DES(Data Encryption Standard) etc..

Asymmetric key algorithm:

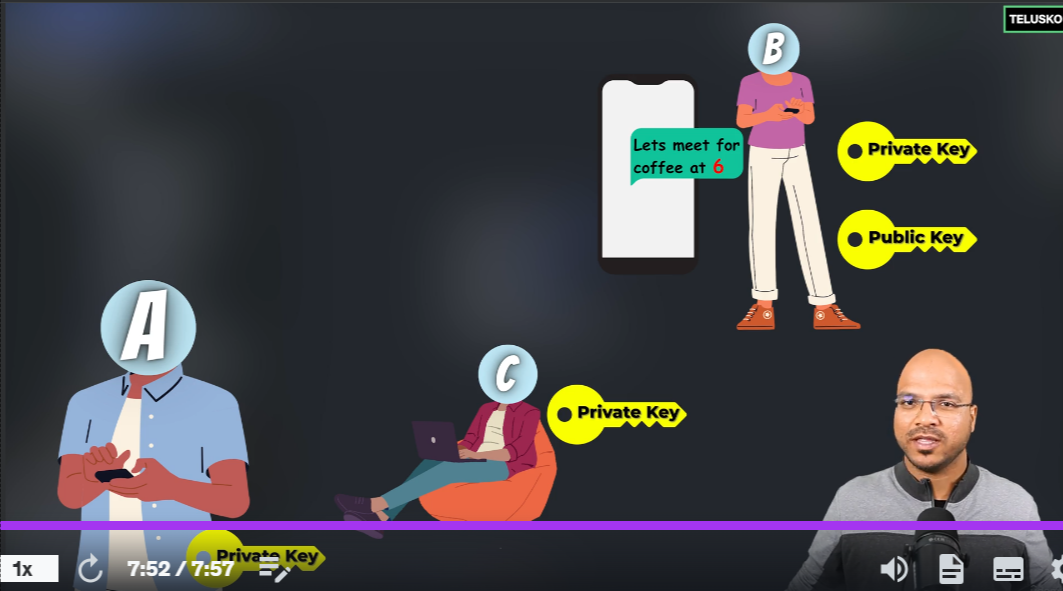
(1)RSA

(2)ECC etc..

And every algorithms have their own strength and weaknesses. Some are faster and some are more securer. Depends on the requirement, we can choose them.

Problem with Asymmetric key:

Even though we are sending datas with public and private keys, it can be hacked.



For Eg, When A sends a data to B using B’s public key, C can hack that data packet from the network and change the data and send it in the same public key which A also sent. So, that B can access it with B’s private key but it don’t know **who sent the data**.

We can solve this problem using Digital Signature.

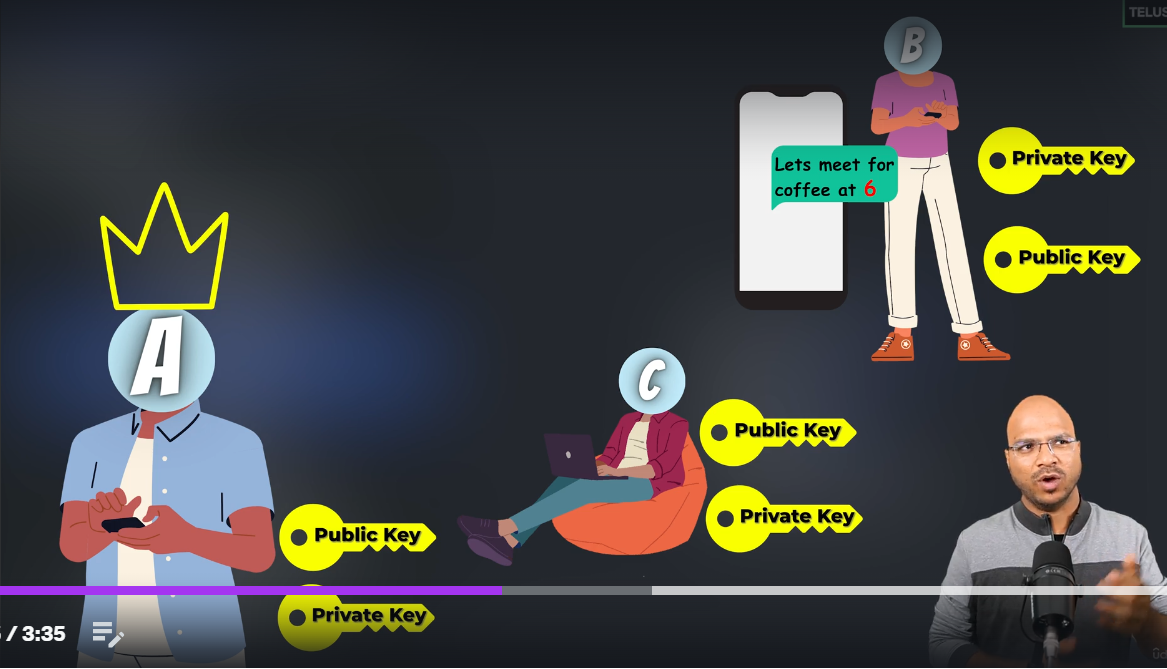
**Digital Signature:**

In Digital Signature, A uses A’s private key to encrypt the data, so that B can use only use A’s public key to decrypt the data. In that way, B can identify that only A has sent this data.

Bcoz, even if C access the data sent by A using A’s public key and change it and sent it to B, C can only send the data with its private key. So, when B receives that data, it will try to decryt the data with A’s public key.

But, it will not be decrypted as the data is sent by C’s private key and it can only be decrypted using C’s public key.

In this way, B identifies that data is not sent by A and someone has changed it.



But the problem here is, there is **no security**.

Here comes the double encryption.

A first encrypt the data using A’s private key and then again encrypt it using B’s public key. So, when C tries to hack it, it first decrypts it using A’s public key and then again it tries to decrypt it using B’s private key but C doesn’t know B’s private key. So, C can’t hack it.

In this **double encryption** way, the data is both **secure** and can be identified by the receiver that **who sent it**.

**JWT(JSON Web Tokens):**

Whenever a client login with credentials, the server will check for the credentials in the DB. In most apps, like in fb insta Zomato etc, we just login once and we can access it without logging in again from the next time.

Usually when u login a app, there will be sessionID created & stored in cookie. So that everytime you re-enter, it will check if the sessionID is active and allow us.

But, this will not work in everycases. For big applications, there will be many servers and in this case, we can’t use sessionID bcoz sessionID can be saved only in one server.

A screenshot of a computer

Description automatically generated

Another way is, everytime you can check the credentials in the shared DB of every servers. But this is not a good option.

For this only, JWT comes into action. JWT is basically a token which will be given to client by server when he logins for first time.

So that, the client can just show the token whenever he opens the app(access the server).

The token contains 3 sections:

A screenshot of a computer

Description automatically generated

Header: contains the algorithm name & type.

Payload: contains the name, issuance & expiry time

Signature: this has the signature of server, which issued the token so that no one can access the token.

So basically the server issues the token, with the details and signs it & gives to client. Only that client can use that server and no one else can access it.

Notes(avoid):  
This token is only secured with signature but not encrypted. So you shld not give any sesnsitive info on the payload. Only basic detail shld be there. Also payload info shld be small.

JWT can also be encrypted.

**Practical implementation:**

So basically to implement JWT, there are only 2 steps:

1. Create JWT token
2. Validating JWT token (whenever client access from the 2nd time JWT needs to be verified by the server)

**Create JWT Token:**

1. First, we need to disable the login form to register and login pages/requests. For this we need to change in SecurityConfig class,



By doing this, we can access register and login pages without login form. Rest all requests require login form.

1. Validate the User & return token when logged in (in UserController):

@PostMapping("login")

**public** String login(@RequestBody Users user) **throws** AuthenticationException {

//AuthenticationManager ->AuthenticationProvider -> Authentication

Authentication authentication=authenticationManager.authenticate(**new** UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword()));

**if**(authentication.isAuthenticated()) {

**return** jwtService.getToken(user.getUsername());

}

**else**

**return** "Login Failed";

}

First we need to authenticate the username & return the token to client.

(3)Create separate JwtService class for JWT token:

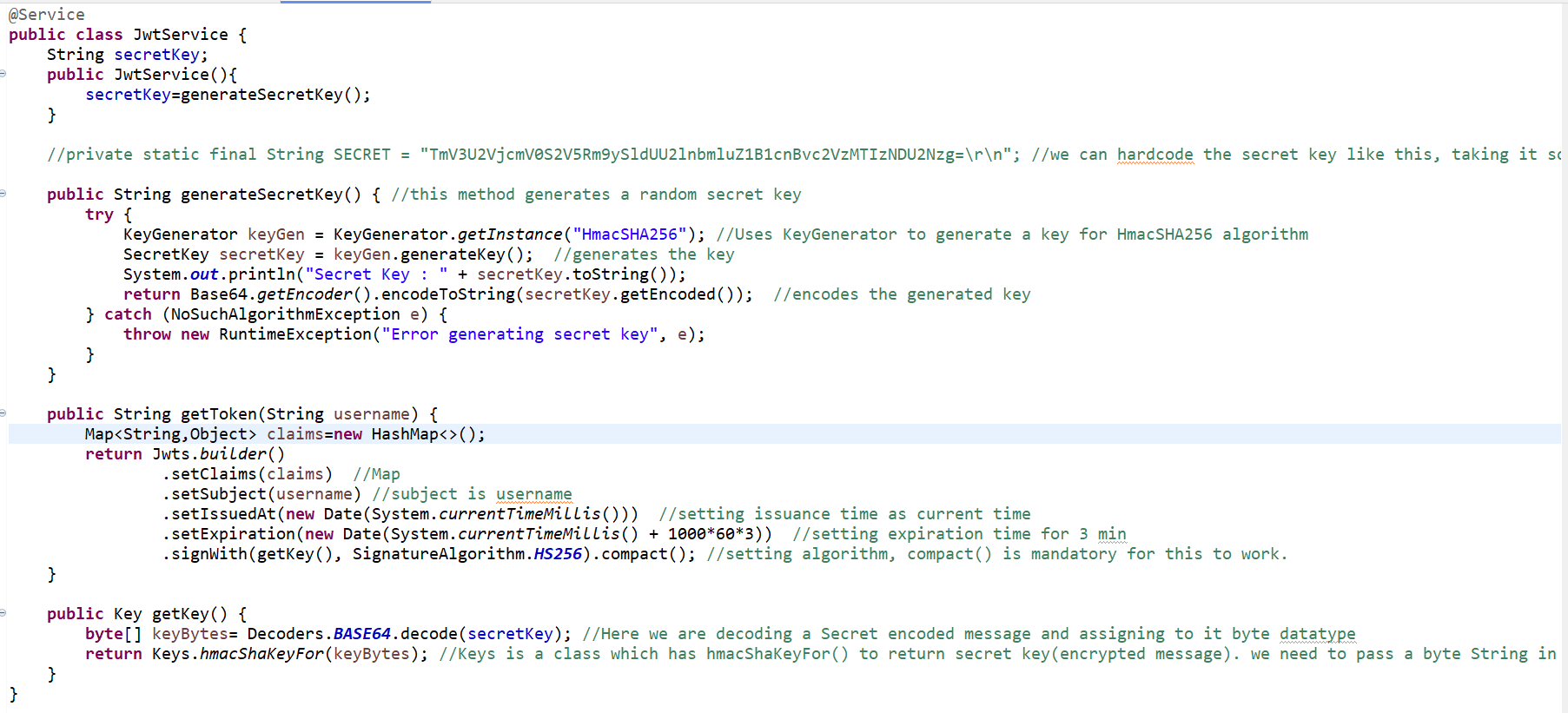
Note: don’t go deep on the code as we will implement these security only once. Just understand the concept.

Add these 3 dependencies,



These are the libraries by which JWT is implemented.

Then create JWTService class,



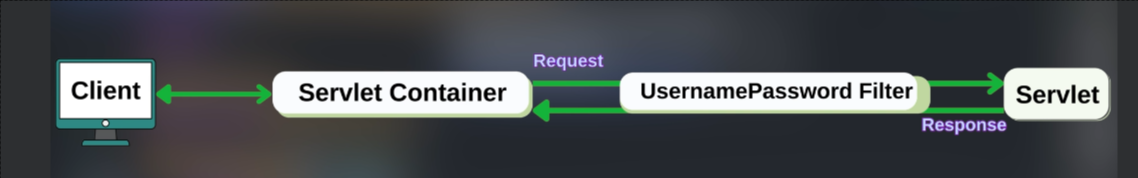
To generate token, we are using getToken() method, and in this method, we need key. And to generate that key, we are creating getKey() method.

In getKey(), we need a secret key. For that we are creating a method called generateSecretKey() method.

That’s it. Token is generated. Now when you login, you will get a token. Now we have to validate that token.

**(2)Validating the token:**

Usually Spring Security by default uses inbuilt UsernamePasswordAuthenticationFilter to validate the credentials.



But we have to add our own filter(JWTFilter) to validate the token.

A screen shot of a computer

Description automatically generated

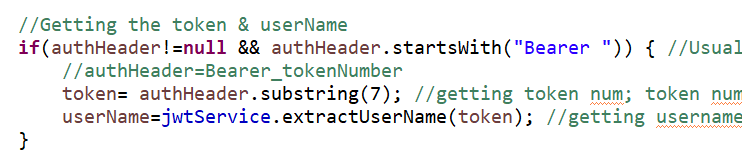
First tell in the SecurityFilterChain in SecurityConfig class, that you are going to add a filter.



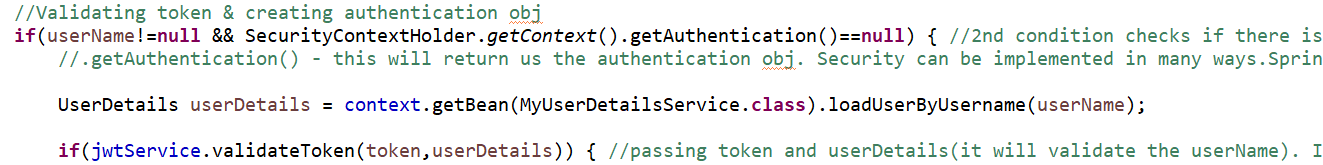
Create JwtFilter class:

(Refer JwtFilter.java for more detail)

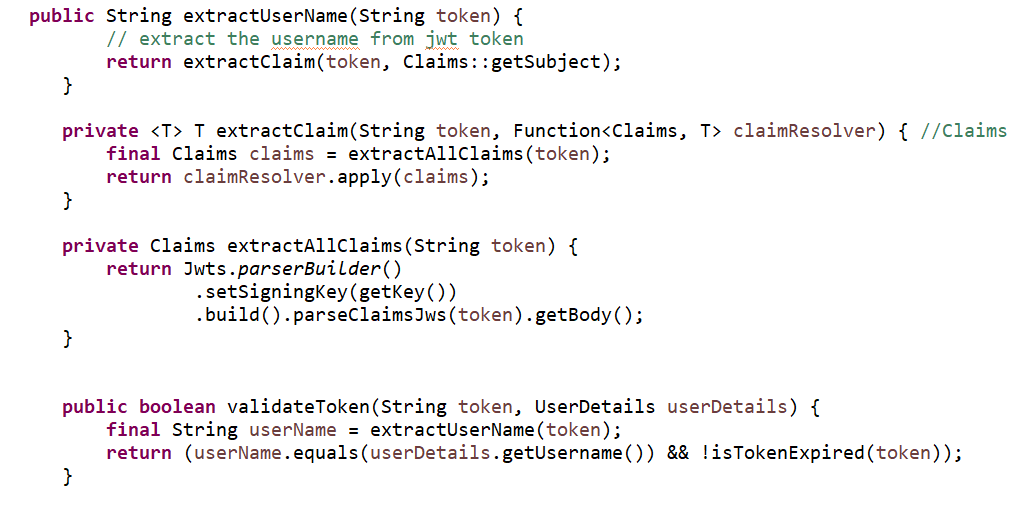
(1)Getting the token & username:



(2)Validating token:

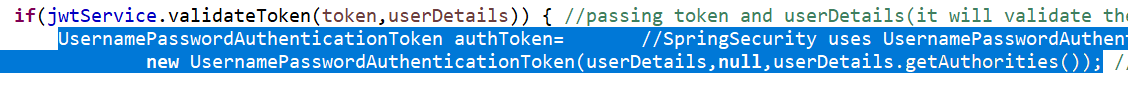


validateToken() method is defined in JwtService class.



Also other methods from which we need to extract data from token, we defined it in JwtService class.

(3)Create Authentication object:



(4)Pass the filter to the next one:



That’s it. We have created and Validated the JWT token.

**OAuth2(Open Authorisation):**

In most social media platforms, when you register or login, there will be a option called Sign in/up using google account. Implementing this is called OAuth2.

Here you don’t need to verify anything. Security will taken care by google acc itself. You just have to add that option using OAuth2.

Just 3 steps to implement OAUth2:

Step 1: Create custom SecurityConfig for OAuth2. 

Here instead of default pop-up login form, it will display OAuth2 login form.

Step 2:Create new app/credentials in Google cloud console:

<https://console.cloud.google.com/> -> 3 dots in left top corner -> API’s & Services -> Credentials -> Create Credentials -> OAuth Client ID

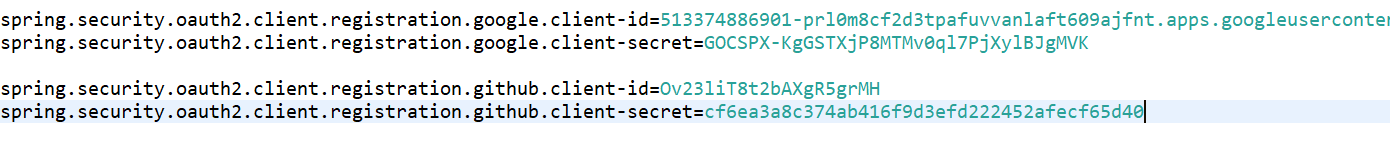
Here, give application type as web application. Name – anything.

In **Authorized redirect URIs,** give “http://localhost:8181/login/oauth/code/google” (base link with http,localhost&port number followed by /login/oauth/code/google).

Once you create app with this, you will get a client id and secret.

Step 3:

In applicatin.properties, add client id and secret,

****

That’s it. Oauth2 login is ready.

Same you can do for github login also. For that you need to login ur github acc -> Profile -> Setting -> Developer Settings -> Create a app and here Authorised redirect URI will be in the name of **Authorization callback URL.** In Homepage URL, give <http://localhost:8181>. That’s it. Client ID and secret will be created. Give in properties file.

Refer SpringOAuth2Demo project .