**15. OAUTH2 Auth server using spring authorization server – section16**

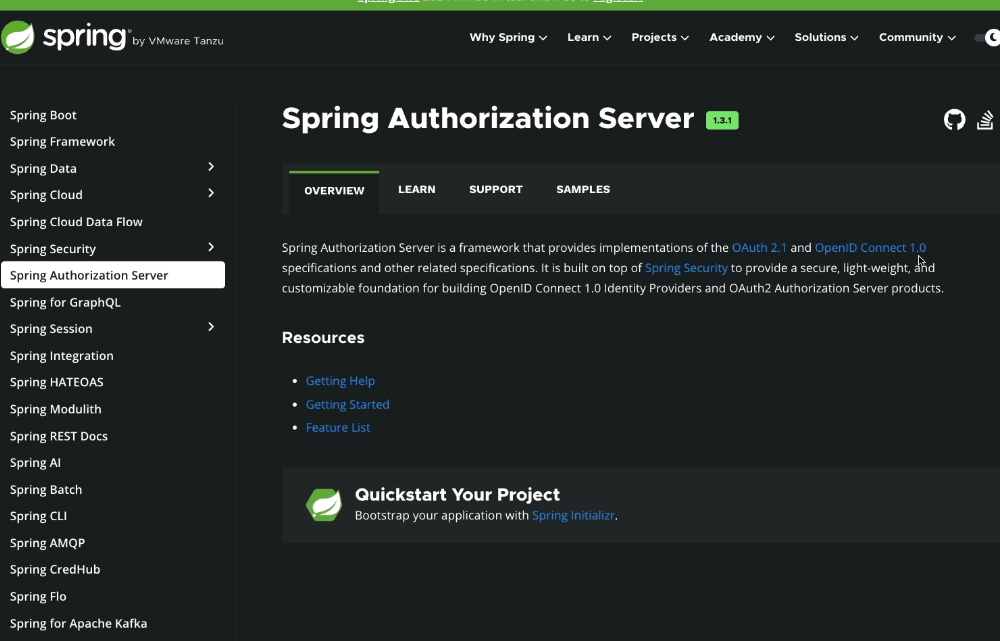
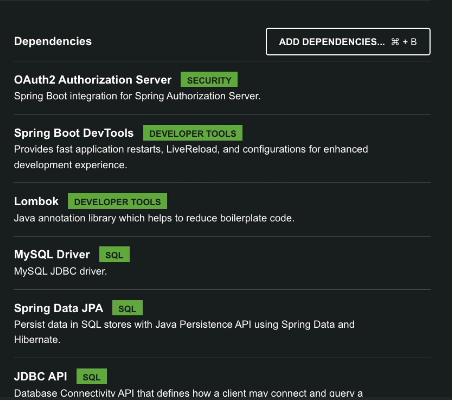
 Introduction to Spring Authorization Server

Inside this spring.io website, if you can go to the projects and click on the Spring Boot, you'll be able to see list of projects supported by the Spring framework.

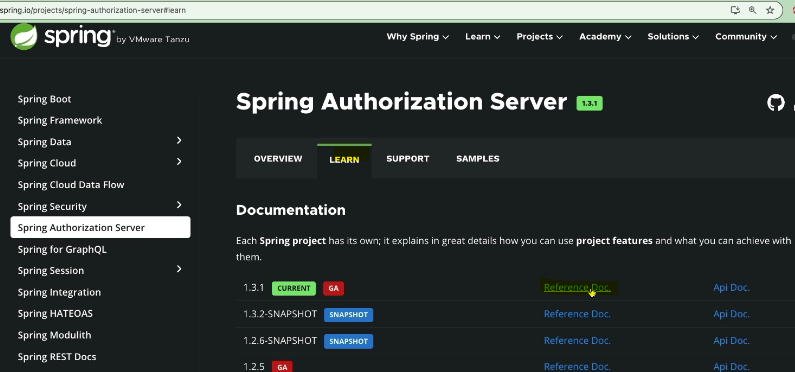
Here there is an option with the name Spring Authorization Server.

Using this project or using this framework, we should be able

to implement our own authorization server that follows the standards of OAuth2.1 under OpenID Connect 1.0.

Set up of Spring Authorization Server



right now it only supports a bare minimum amount of features.

This is not going to be a fancy product like Key Clock where we have beautiful admin UI console to perform everything.

So this project as of now, it does not support any admin UI, everything we need to achieve with the help of Java code or application properties configurations.

So here, they're trying to explain on how to set up spring authorization server-related configurations with the help of properties that we can define inside the application.properties or application.yaml. So I'm not going to follow this approach

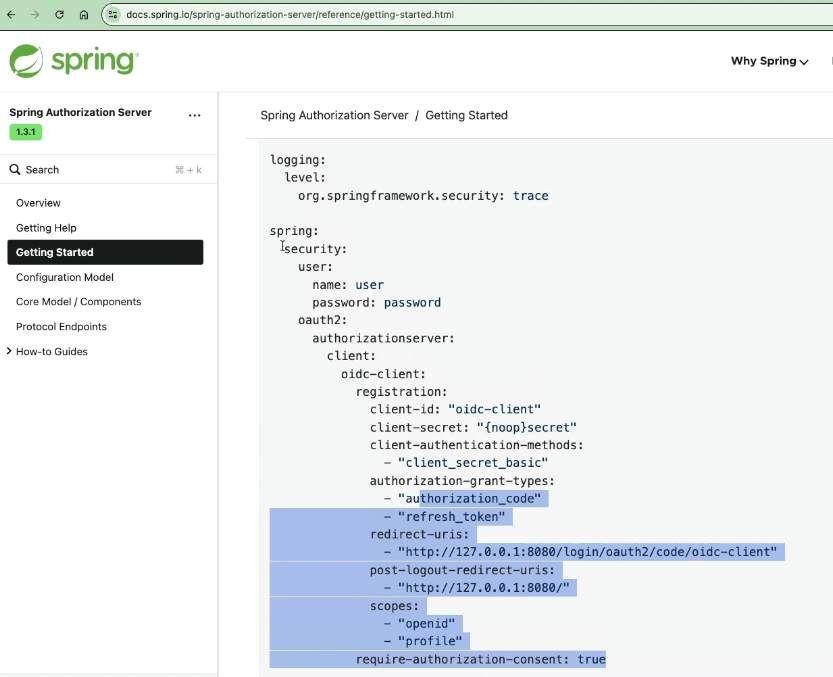
because this approach is going to be super complex and it is also not going to support majority of the complex scenarios that we may have.

But for simpler applications this properties approach may work.

For now I'm going to skip this approach and I'll go to the other approach

where we can configure everything using Java methods,

Java classes, and Java beans.



Bean approach:

So with the help of this defaultAuthenticationEntryPoint(), we're going to redirect the end user to the login page. So whenever there is an exception happens,

the redirection of the end user to the login page is going to happen.

And at last, they also trying to convert these authorizations server as an OAuth2 resource server. The reason why they're trying to convert these as a resource server is to accept access tokens for user info and for client registration purpose.

At last, they're trying to invoke this http.build().

why they have defined two different beans of SecurityFilterChain?

In the very first bean, they try to define all the configurations which are specific to the auth server.

Whereas coming to the second bean, you can see they're trying to

configure these .authenticated() and formLogin(). At the end of the day,

the authorization server also, it is also going to expose some secured APIs

and secure pages.



So all these pages, since they have to be authenticated properly, and in the case whenever we want to access these pages, we should be able to access them by entering our credentials with the help of formLogin() approach.

So that's why they tried to create two different beans.

The next bean that we have here is of type RegisteredClientRepository.

If you can recall inside keycloak, whenever we want to register a client application,

we'll go to admin console and we try to add the client details.

But coming to the spring authorizations of our framework, here we don't have admin console.

So whenever we want to register a client, this is how we need to register.

We need to register a client with the help of this registered client class.

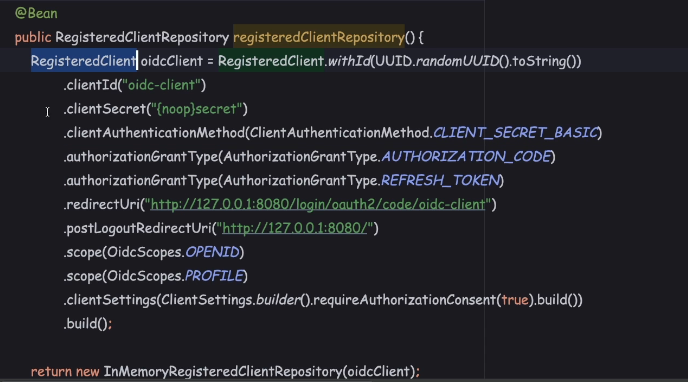
So inside this class we have so many methods like clientId, clientSecret,

what is the type of authentication method, what are the authorization grant type

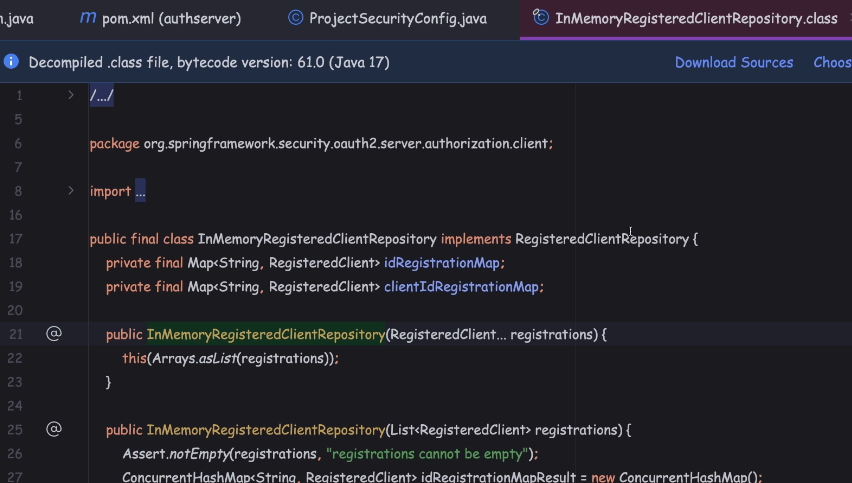
that your client is going to support, redirectUri, scopes.

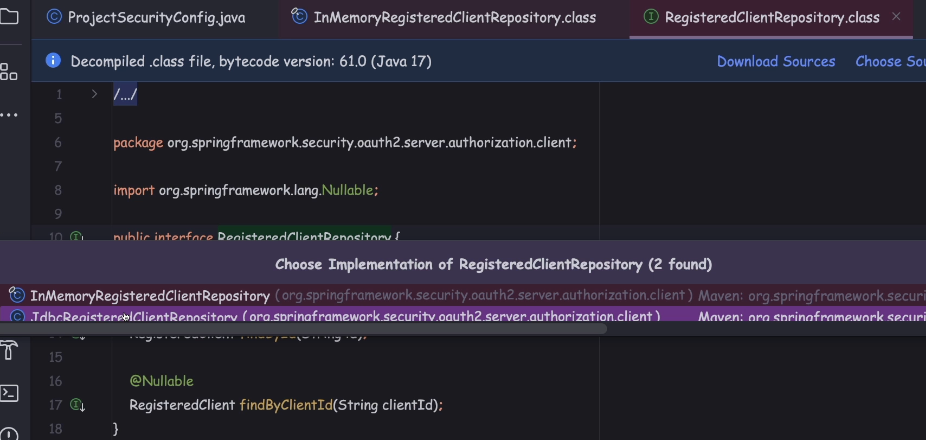
So once the registered client object is created, they're trying to pass the object of this

to the InMemoryRegisteredClientRepository. This means all the clients that we are going to configure, they're going to be saved inside the memory of the application.





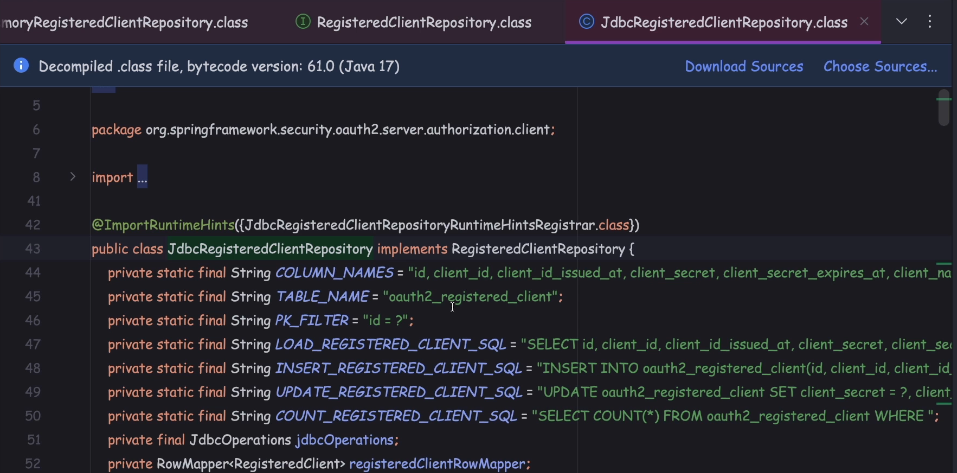




So if I can open this interface and look for the implementations, we also have a class with the name JdbcRegisteredClientRepository.

So in case, if you're looking to store all the client details inside a database,

we can leverage this class. Otherwise, if you are looking to store the client details inside a database with your own columns and table structure, then you need to implement this interface and you need to build your own client repository class.





The next two bean that we have is around the JWK source. Any auth server that is built based upon the OAuth2 standards, behind the scenes, it's going to generate

private and public certificates or keys.

So using the private key or certificate, the auth server, it is going to digitally sign the

access tokens, ID tokens, or any other tokens. On the resource server side,

they should be able to validate these tokens locally by using the public certificate or public key.

So this method or this bean, it is going to take care of generating a public key

and a private key during the startup.

So it is also going to use the helper method, which is generateRsaKey().

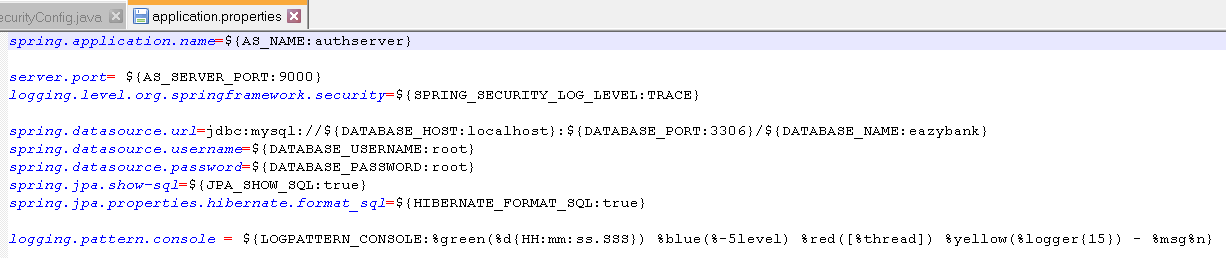
At the end of the day, these methods, they're going to help the authorization server

to generate a key pair, which is going to have both the private and the public keys.

the same, we are trying to configure as a JwtDecoder with the help of this OAuth2

authorizations of work configuration.

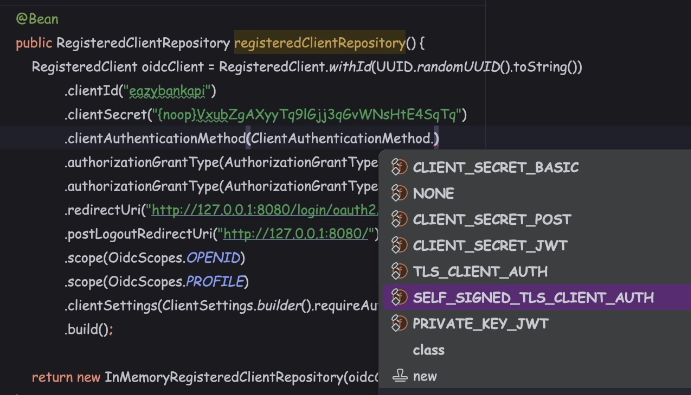
So with this, what we're trying to tell to the auth server is, so whenever you are trying to generate an access token, please digitally sign it with the help of these JwkSource.



 Creating Client Credentials inside Spring Auth Server for API-API invocation

If you want to mention a bcrypt hash value for client secret, You have to calculate the bcrypt hash value for this plain text and inside this prefix you need to mention bcrypt.

For now let's follow the plain text password.



The clientAuthenticationMethod(), So using this method, we need to tell to the auth server how the client application they're going to send these credentials,

whether they're going to send as part of the header or as part of the body.

So I'm going to remove these because I want my client application

to strictly support CLIENT\_CREDENTIALS but not any other grant type flow.

Next we can remove this redirectURI related configuration and postLogout related configurations.

The reason is whenever a client is registered with the help of CLIENT\_CREDENTIALS,

there's a good chance there won't be any UI involved, in such scenarios,

we can simply get rid of these configurations.

So these scopes are nothing but the roles that we want to configure

I'm going to invoke another method, which is accessTokenFormat().

To this method, I'm going to pass a constant which is accessTokenFormat.SELF\_CONTAINED.

If you can recall our previous sections discussions, I highlighted about the token format.

In general, there are two commonly used formats. The very first one is JWT format

and the other one is opaque token format.

So whenever you are looking for the JWT format, you need to mention the SELF\_CONTAINED. So if you see here, SELF\_CONTAINED tokens use a protected and time-limited data structure that contains token metadata and the claims of the user and the client.

JSON Web Token is a widely used format. So with this configuration, the resource server, it is always going to get the access tokens and other tokens in a JWT format

and it can locally validate the tokens with the help of public certificate.

Once the end user is trying to be logged in and once the login is completed,

the auth server, it is going to show him the consent page where the end user has to accept.

But since our client is following the CLIENT\_CREDENTIALS, we don't need these configurations.

Start auth server

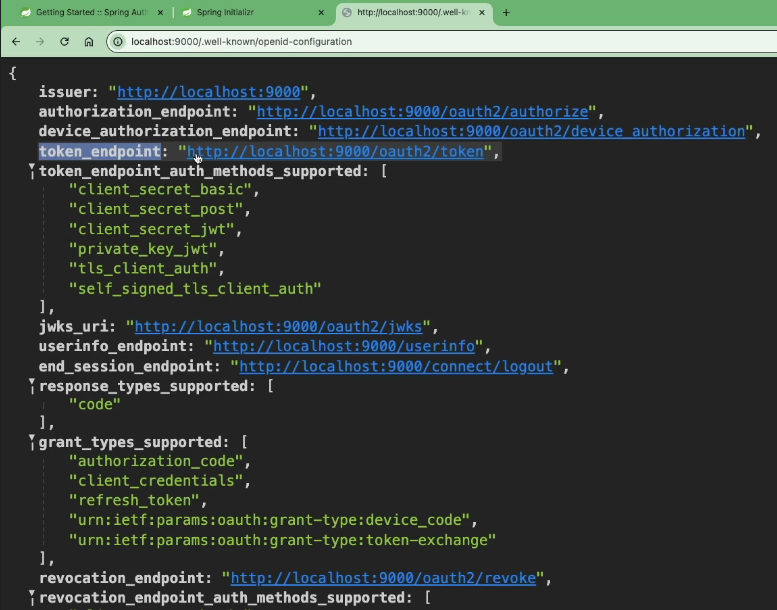
Inside the Postman, under the section\_16 folder you can open the myAccount request. So here I have populated all the required details like under the authorization, I have selected the OAuth 2.0. Coming to the token related configurations. So token name, I kept as AccessToken and the grant type is going to be Client Credentials.

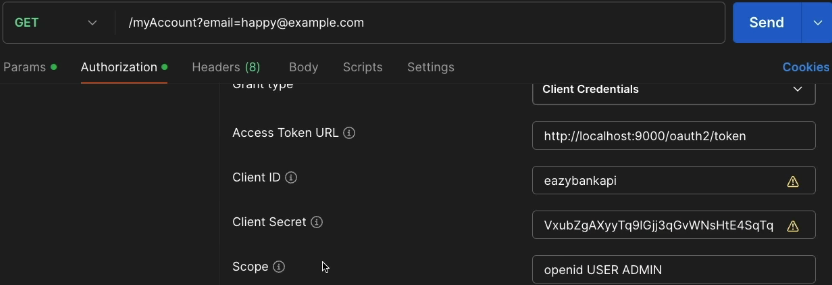
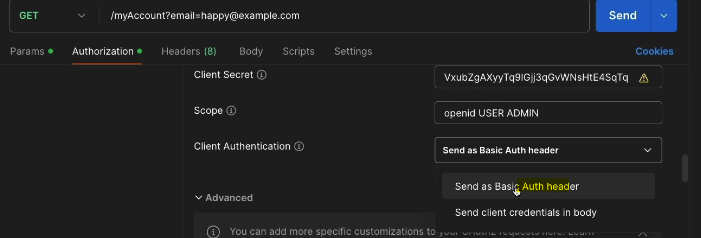
And coming to the Access Token URL, we need to give our own auth server URL.

So localhost:9000 oath2 two slash token is the URL.

You can get all the URL details that are supported by our auth server

by opening the well-known OpenID configuration URL.



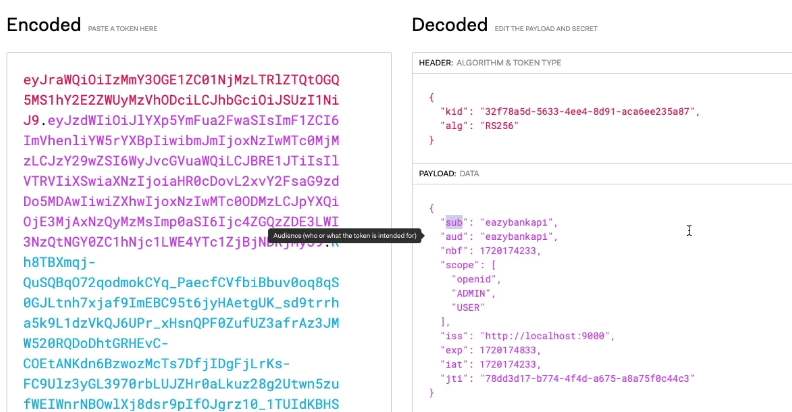
we need to select the option which is Send as Basic Auth header,

because here we have configured CLIENT\_SECRET\_BASIC as the client authentication() method.

you can click on this, Get New Access Token.

If everything goes well, you're going to get an access token and there won't be any ID token and refresh token because ID tokens is only applicable for the end users

and the refresh token, we are not going to get it because we have not enabled the Refresh Grant Type flow.



I'm going to explain how to customize the tokens in the coming lectures.

I'm assuming you are clear on how to register a client that support Client Credentials Grant Type flow. As of now, we're able to get an access token successfully.

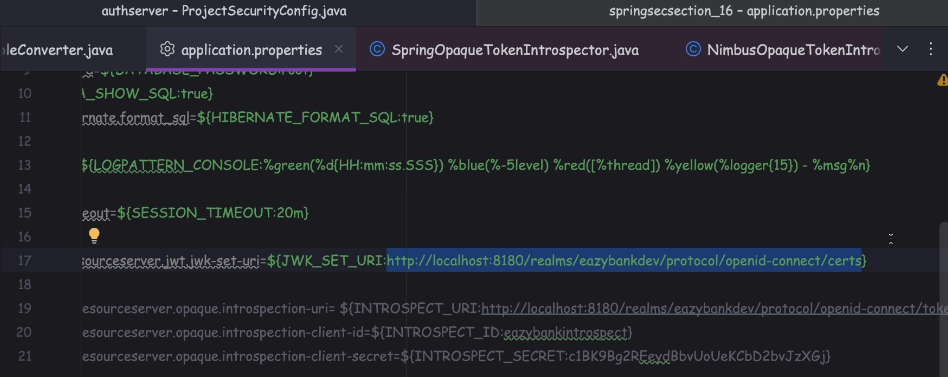
 Client credentials grant type flow demo with Spring Auth Server

To test the scenario end to end, we need to provide these access token

to the resource server. And if the resource server provides us valid response,

then it's a confirmation to us that our configurations are working end to end.

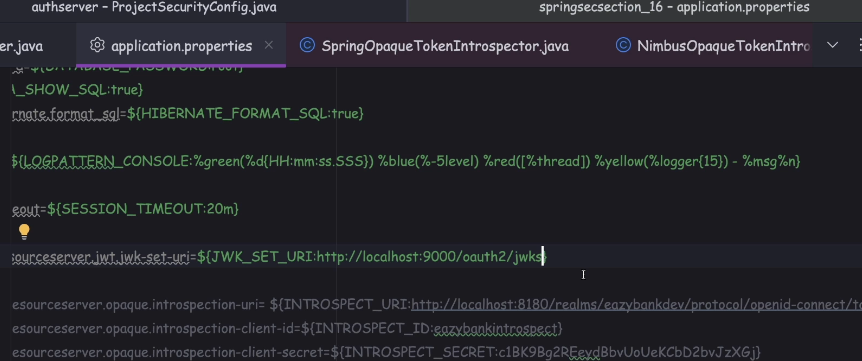
So let's try to set up the resource server from previous section 15.



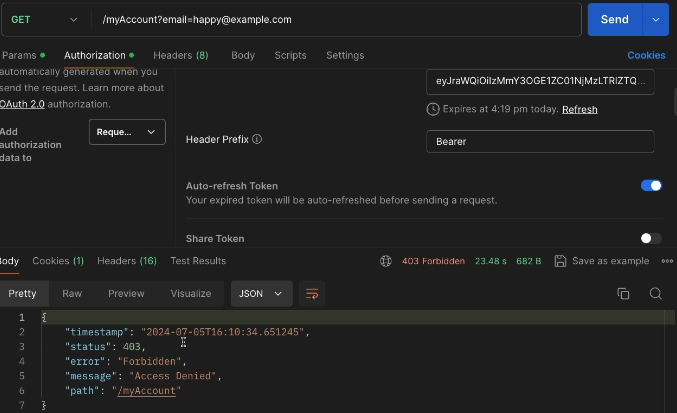
So here under the application.properties, as of now we are pointed

to the keycloak related JWK\_SET\_URI url.

In this place we need to mention our own local authorization server URL.



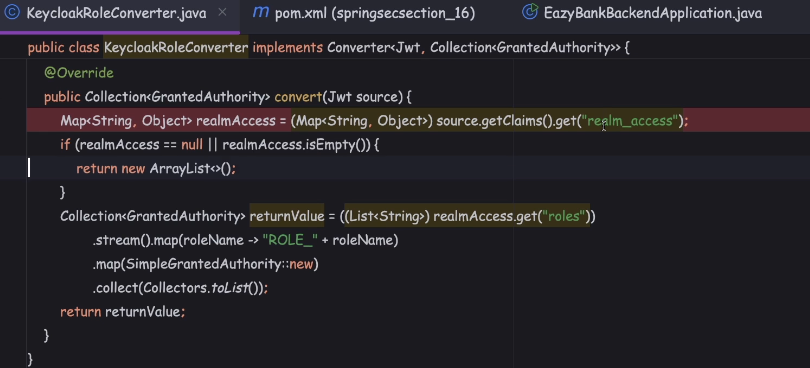
With this, we establish the link between the resource server and the authserver.



So you can see realm access is null and with that we are going to return empty roles

for this client application. And with that what is going to happen?

We are going to get beautiful 403 error.

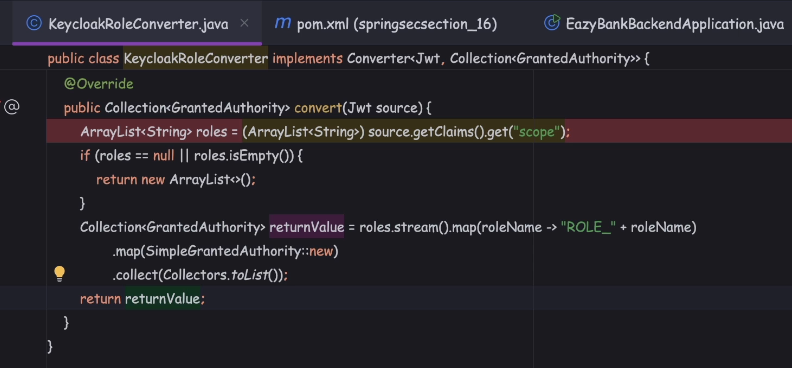


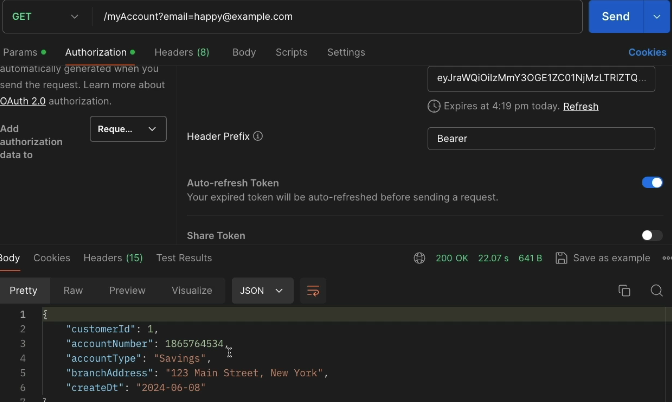
So if I try to paste jwt token in jwt.io, so the roles information, they are available inside the scope. So first I need to read the scope details from the claims.

When I read these scope details, I'm going to get a collection

of scopes or roles information. Inside my KeycloakRoleConverter,

I need to look for the client with the key scope.





As of note, the roles are being added under the scope element,

but sometimes we may get some requirements to create our own elements,

our claims inside the access token are in any other tokens.

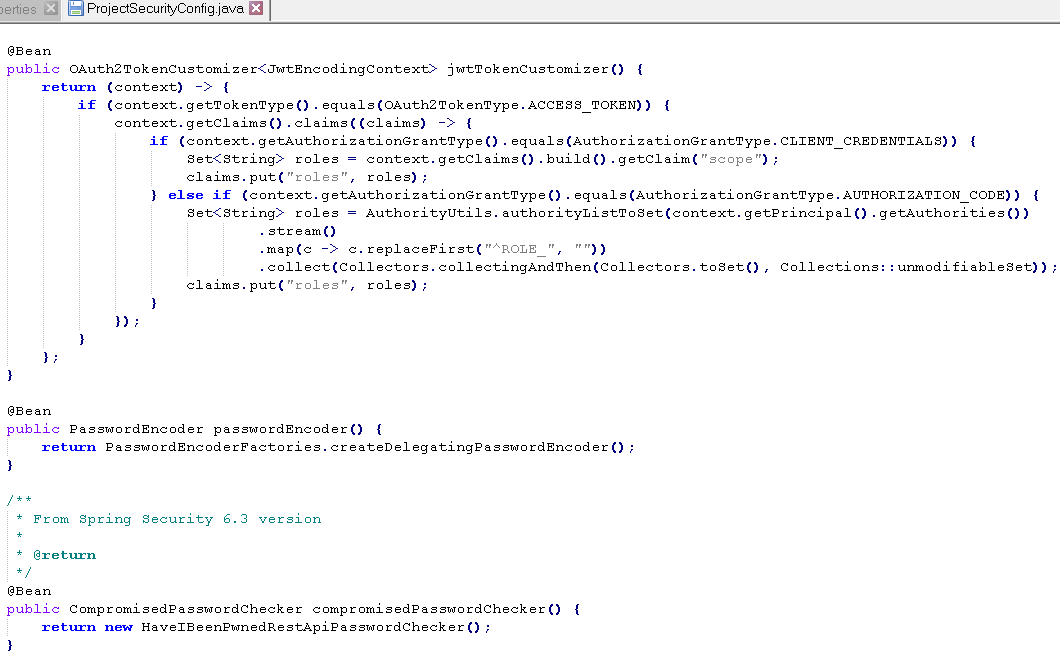
So to handle such scenarios, the spring authorization server framework

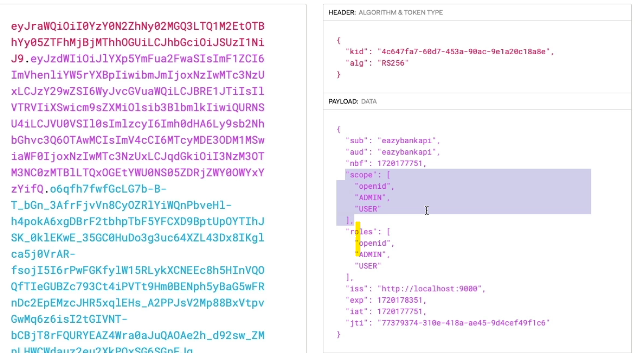
gave us flexibility to define some logic which is going to be executed.

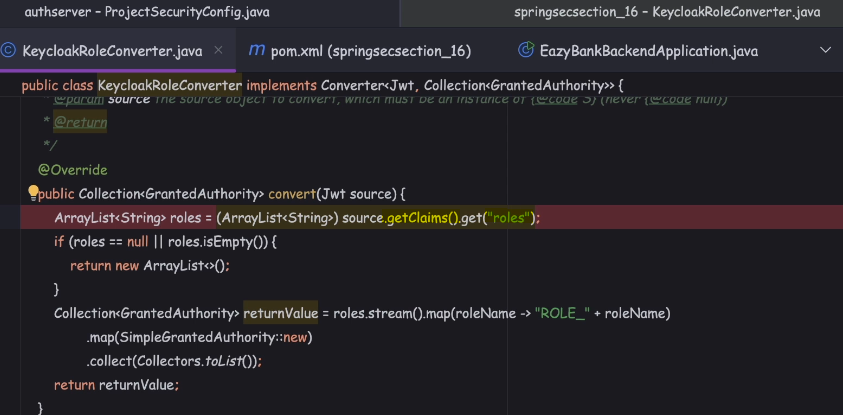
And accordingly, the token is going to be modified during the token creation process.

So let's try to understand how to implement the same in the next lecture.

OAuth2 Token Customization in Spring Auth Server

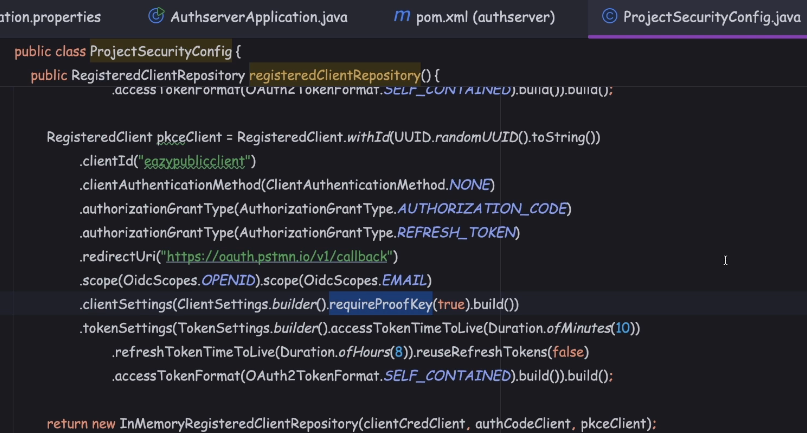






Creating clients inside Spring Auth Server for Auth Code & PKCE grant type flows





Now coming to the client authentication() method, I need to select the option which is none because we know inside the PKCE flow, clients they're not going to share any secret for authentication. Always the flow is going to be secured with the help of code challenge and code verifier.

So if you try to analyze the differences between the PKCE client and AuthCodeClient,

there are only two difference. The very first one is the ClientAuthenticationMethod.NONE is going to be none. And the next one is under the client settings, we have configured this requiredProofKey as true.

And we don't have to configure the SHA 256 algorithm as a PKCE SHA algorithm because by default, spring authorization server, it is going to consider the same.

Can we go ahead and test the token generation scenario?

Think about it, we can't test it. As of now, my auth server does not have any clue

around how to authenticate the end user, where the end user details are stored,

how to perform the authentication. So such details are missing right now.



Inside this official documentation in the example code that they have provided,

you can see here they have configured a single user inside the memory.

So with this, what is going to happen?

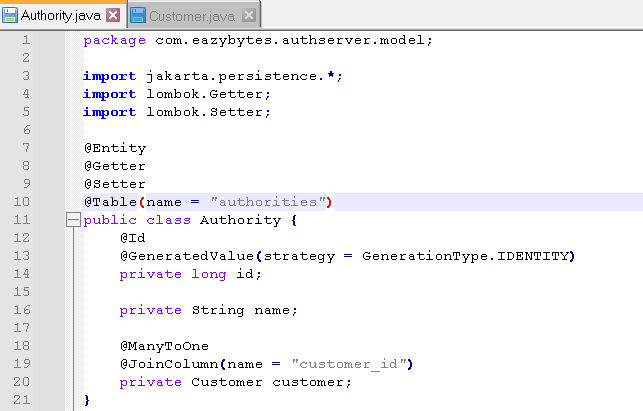
Whenever an end user is trying to test the PKCE flow or Authorization Code Grant Type flow, they are going to enter this username and password and this role is going to be assigned to the end user.

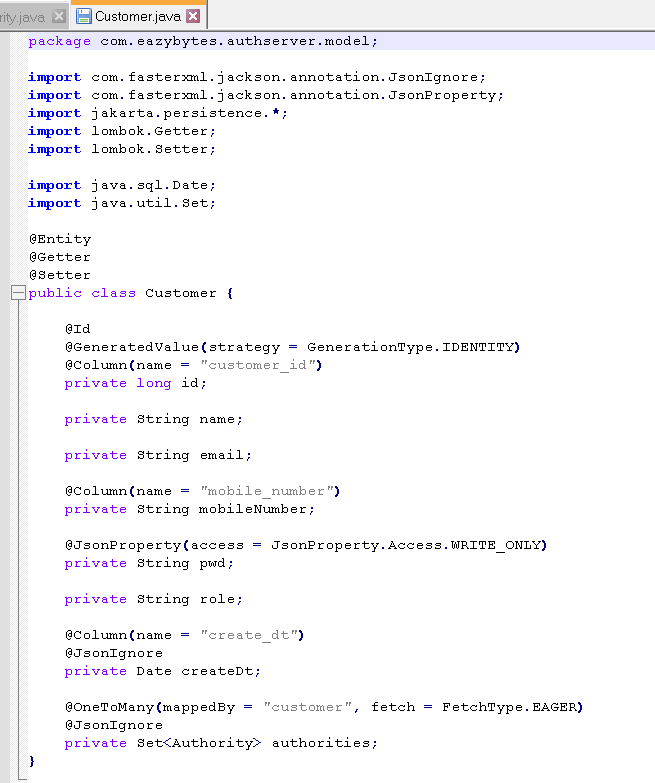
But in our scenario, since we are looking to properly use a MySQL database as a storage system, what we can do is we can try to leverage authentication provider user detail service implementation that can be leveraged by the spring security framework during the authentication process.

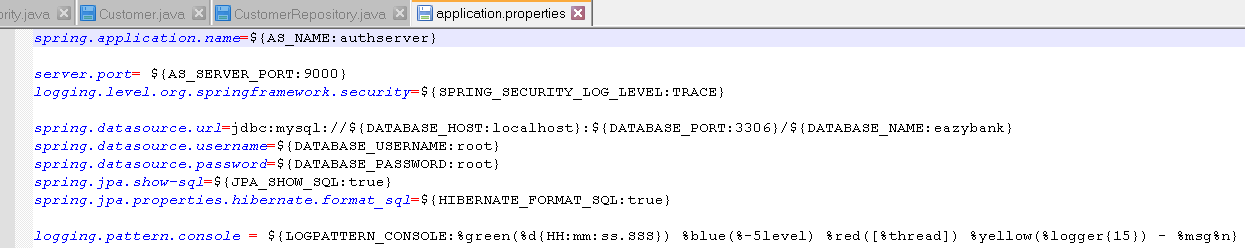
Updating Spring Auth Server to authenticate the end user using DB



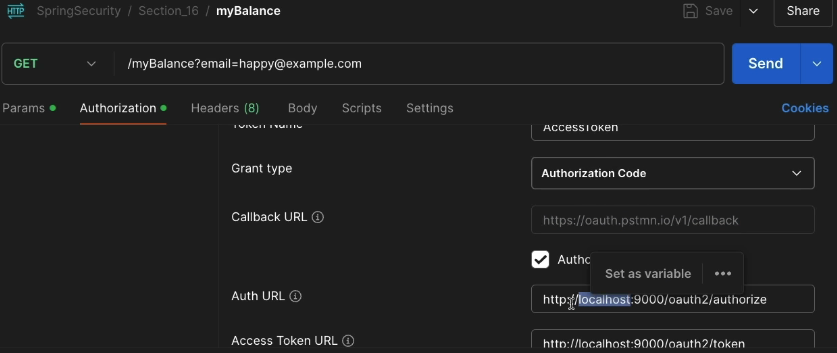
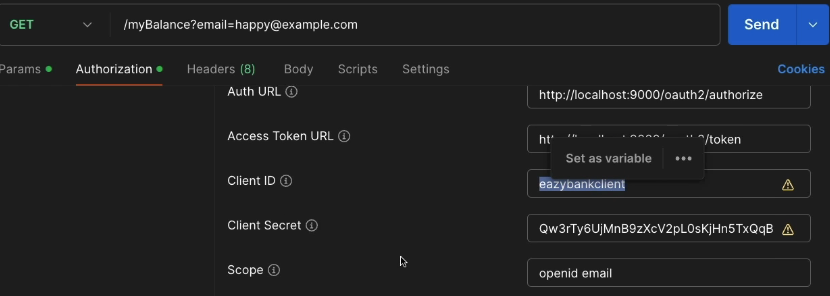


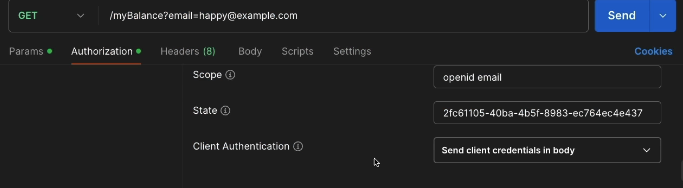
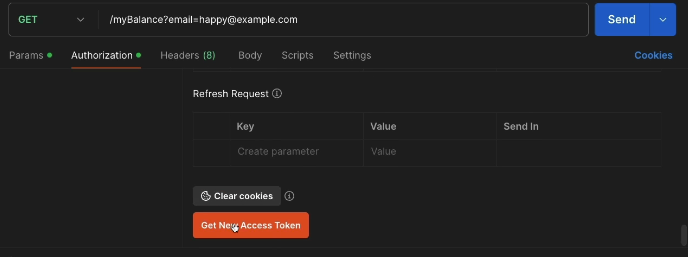




 Auth Code & PKCE grant type flows demo with Spring Auth Server

And state is a random CSRF token value, you can mention any value.

Next, coming to the Client authentication, this time you need to make sure

you're selecting the options: Send the client to credentials in body.

The reason is here we have configured this constant, which is client to secret post.

That's why we need to send the credentials inside the body.

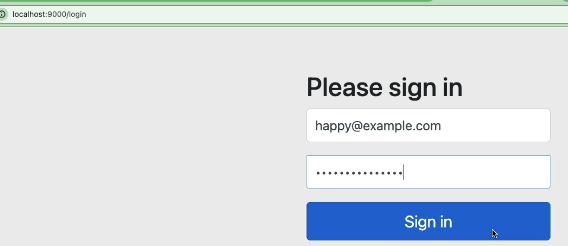
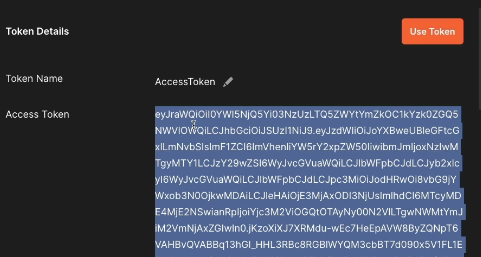
So let me click on these, get new access token and it is going to show me a login page. So here I need to enter the end user credentials that I stored inside the database. So we have end user details configured in the database

with the email as [happy@example.com](mailto:happy@example.com) and the password is easybytes@54321.

So let me click on the sign in, the redirection to the postman completed

and here you'll be able to see the access token and the ID token of the end user.

Similarly, you'll also see and refresh\_token here.

So let's take this access token and try to analyze inside these jwt.io website.

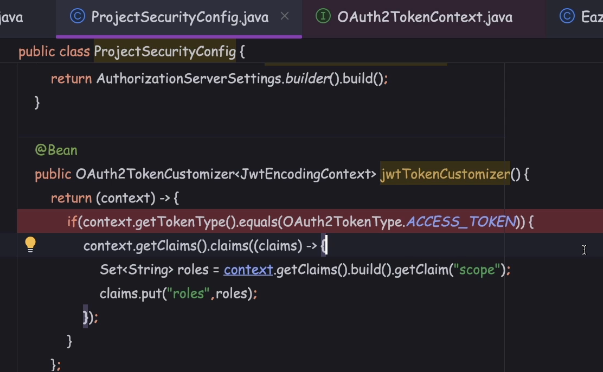


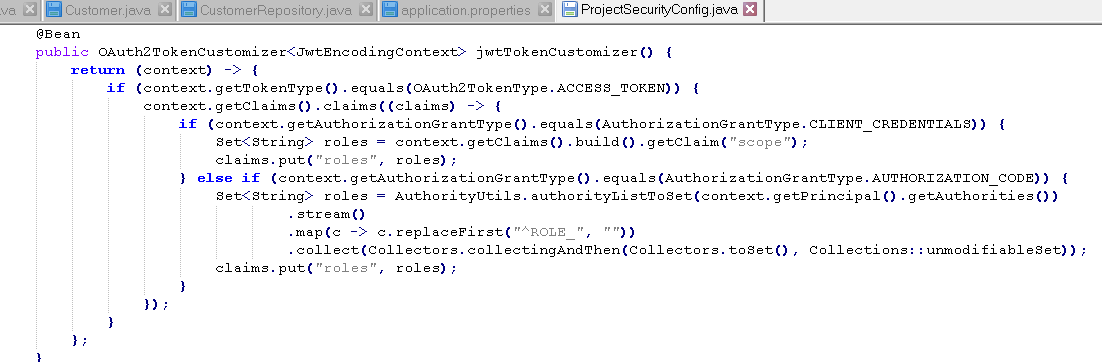
Here if you see under the roles what our scope values are present,

the same are getting copied. With the code that we have written

under these jwtToken customizer. The scopes are getting copied into the roles

but these scopes are related to the client application but not the end user.





inside this map() method is, whatever role name that I'm getting from the database

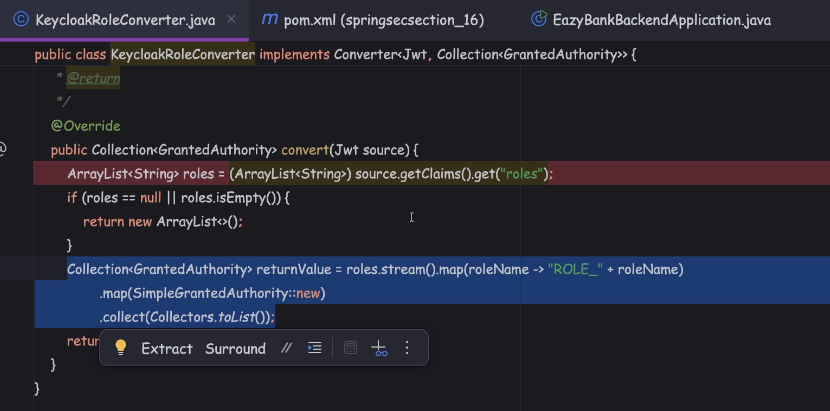
inside the same role name I'm trying to replace the ROLE\_ with an empty value.

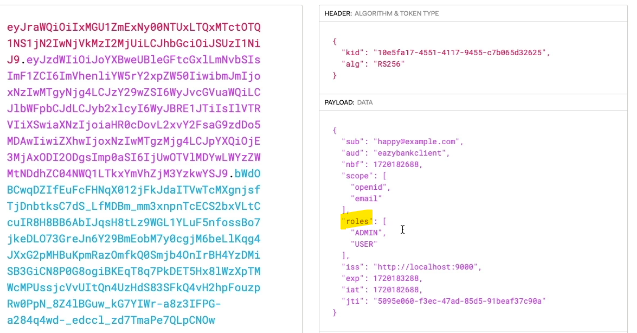
We know inside the database we are always going to store the end user roles with the ROLE\_.

But I want to remove this prefix here, the reason is inside the resource server

under this KeyclockRoleConverter, we have a logic where we are trying to add this prefix again.

If you can control this logic not to be executed for the Authorization Core Grant Type flow, then happily you don't need these removing off role underscore you can directly convert the stream() into an Set object.





Once the build is completed, I'll try to get a new access token.

Let me click on this new access token. The browser is not going to ask me

to enter the end user details because it already has an existing session in the browser with the end user credentials.

So we receive directly the access token. Let me take this access token and paste it here. This time we'll be able to see under the roles we have admin and user.

With this, our code should work end to end.

Let me select this use token and click on this send button.

So let me try to test the PKCE flow with the help of these myLoans request.

So here under the authorization I will select grant type as authorization code with the PKCE that we need to make sure to enter the Client ID as easypublicclient.

Client Secret, we can ignore blank. Code Challenge Method should be SHA - 256

and Code Verifier my postman is going to generate for me, if I left this field blank.

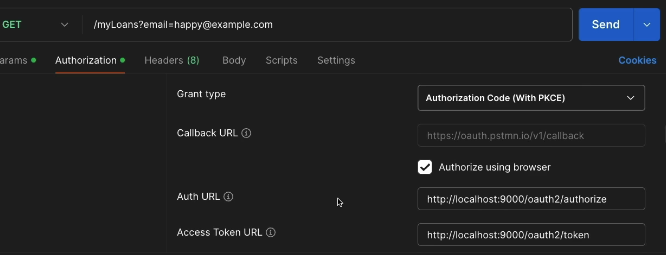
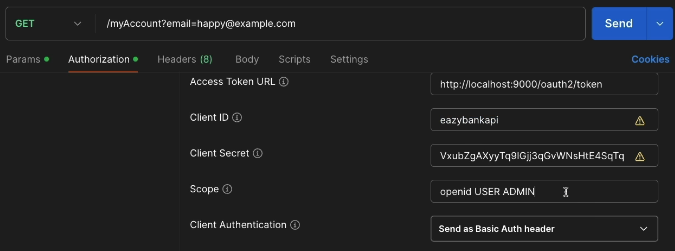
The scope, I'll mention openid email and this is the CSR of token value

and I'm going to send the credentials inside the body.

So let me click on this 'Get new access token The browser authenticated automatically Here I got the access token.

Let me use this access token and send the response and you'll be able

to see and getting the loans details.

Let me show you a negative scenario both in the Client Credentials Grant Type flow

and other grant type flow. So like I was telling before, whenever we are sending the scope, we need to send the scopes that are allowed for the client application.

So inside the software we have configured these three scopes

for the Eazybank api client.

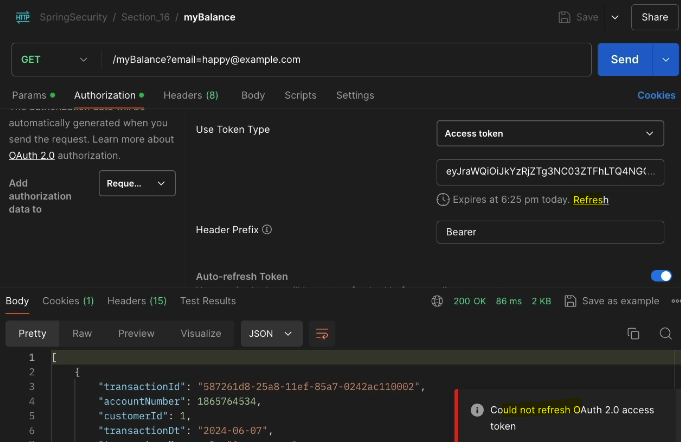
So here if I try to be over smart and try to send a scope with the name manager,

you'll be able to see and getting an error.

So if I click on this, get new access token, you'll see I got an authentication failed.

If I look inside the console, I got an error which is invalid scope.

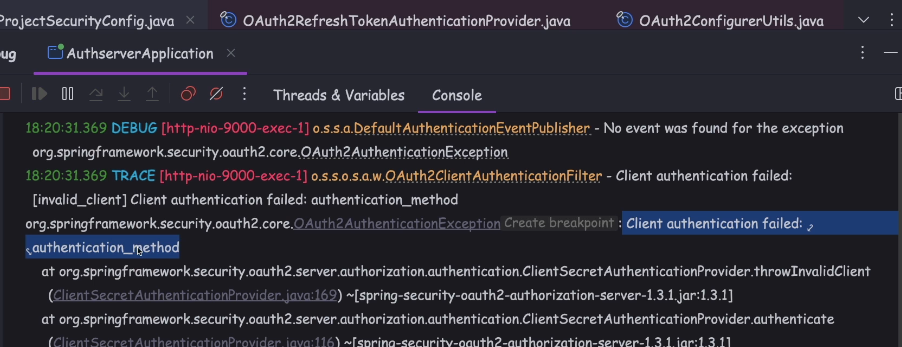
Refresh token grant type flow demo with Spring Auth Server

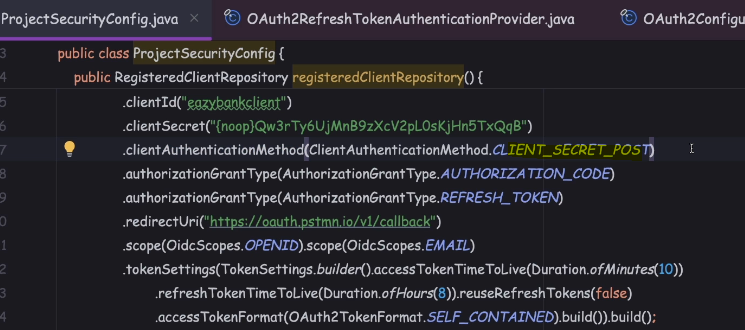


If you click on this refresh button, the refresh token flow is going

to be initiated behind the scenes. So let me click on this refresh

and you'll see I'm getting an error.





Here I have a logger stating that client authentication method failed.

So the authentication() method, whatever right now we configured

for the client is not enough to test the refresh token grant I flow.

The reason why we are getting this issue is as of now for the both these client's Auth code client and the PKCE client, we have configured the authentication() method

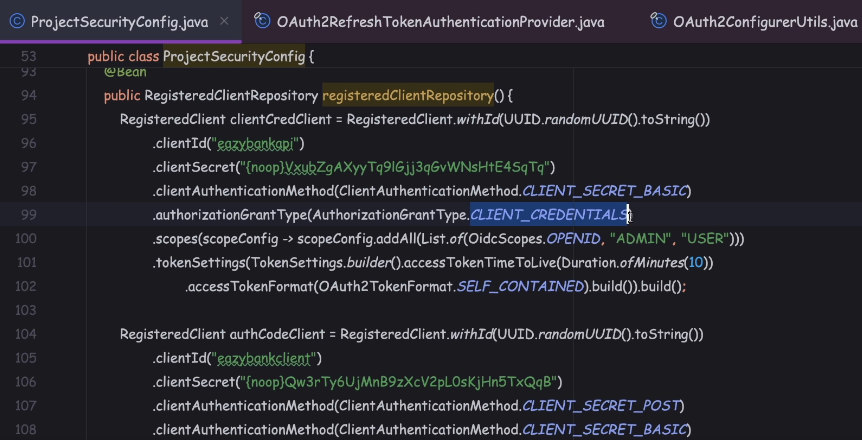
as client to secret post.

But during the refresh token grant I flow the postman it is going to send the refresh token using the http basic format. Since our client is not supporting that method,

we are facing the issue.

So what I can do is I can try to repeat this line of code and instead of CLIENT\_SECRET\_POST, I'll just mention CLIENT\_SECRET\_BASIC, the same,

I'll try to configure for the PKCE client as well.



In clienCcredentialClient, we have not enabled a refresh token grant I flow.

The reason is very simple. This client is supposed to be used by the backend services.

So whenever they're making a service call, they should get an access token

and use the same with the resource award. We should not give the flexibility of the refresh token, but the API to API communication scenario because that is not standard. That's why I have not configured the Refresh Token Grant Type flow

with the clientCredentialsClient.

So let me get a new access token because we have changed our client to configurations. That's why I want to get a new access token before trying the Refresh Token Grant Type flow.

So let me click on this new access token and I received a new access token

and along with the new access token, I also received the refresh token as well.

Let me use this token.

So as of now you can see my token is going to be expired in 10 minutes,

which is 6:33 PM, so right now the time is 6:24. So if I try to refresh this token, I should get a new token with a new expiration time.

So the Refresh Token Grant Type flow, we should be able to invoke

before the access token expiration as well. So if I click on this refresh this time,

you can see I got a new access token that is going to expire at 6:34.

You can also check the console and here towards the end, if you open this post under the RequestBody, you'll be able to see this is the refresh token

and the grant Type Refresh Token.

This confirms Refresh Token Grant Type flow is also working fine.

Demo of Opaque Tokens with Spring Auth Server

Let's make changes inside the OAuth server and resource server to accept and to process opaque tokens.

So the main difference between opaque tokens and JWT tokens is, JWT tokens are self-contained tokens, which means the resource server, they can validate the JWT tokens locally without depending on the OAuth server for each and every request.

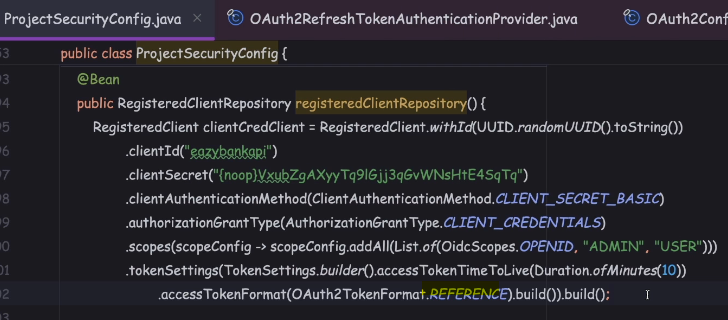
Whereas with the opaque tokens, the resource server is going to rely on the OAuth server to validate that token received for each and every request.

In most of the real scenarios, JWT tokens, they are going to be used,

because they have lot, many advantages. Opaque tokens, they are very rarely used

for super critical applications. But please remember, opaque tokens,

they are going to bring some performance issues as well since the resource server is going to rely on the OAuth server for each and every request.



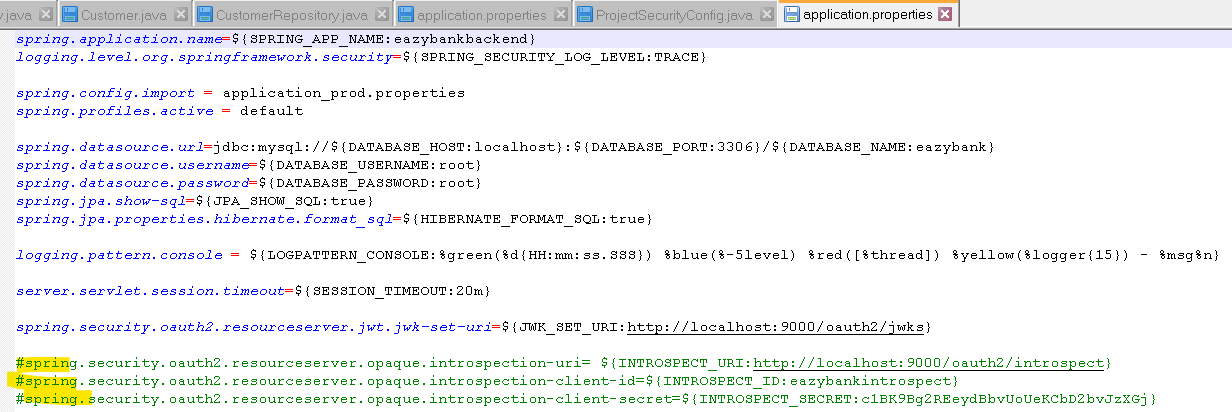
So here, under the client, which is going to support the client credentials,

I'm going to change this self-contained to reference. So whenever we configure this reference as a token format behind the scenes, OAuth server, they are going to generate the opaque tokens.

As a next step, we should also make changes inside the resource server,

so that it can connect with the OAuth server for each and every token introspection.

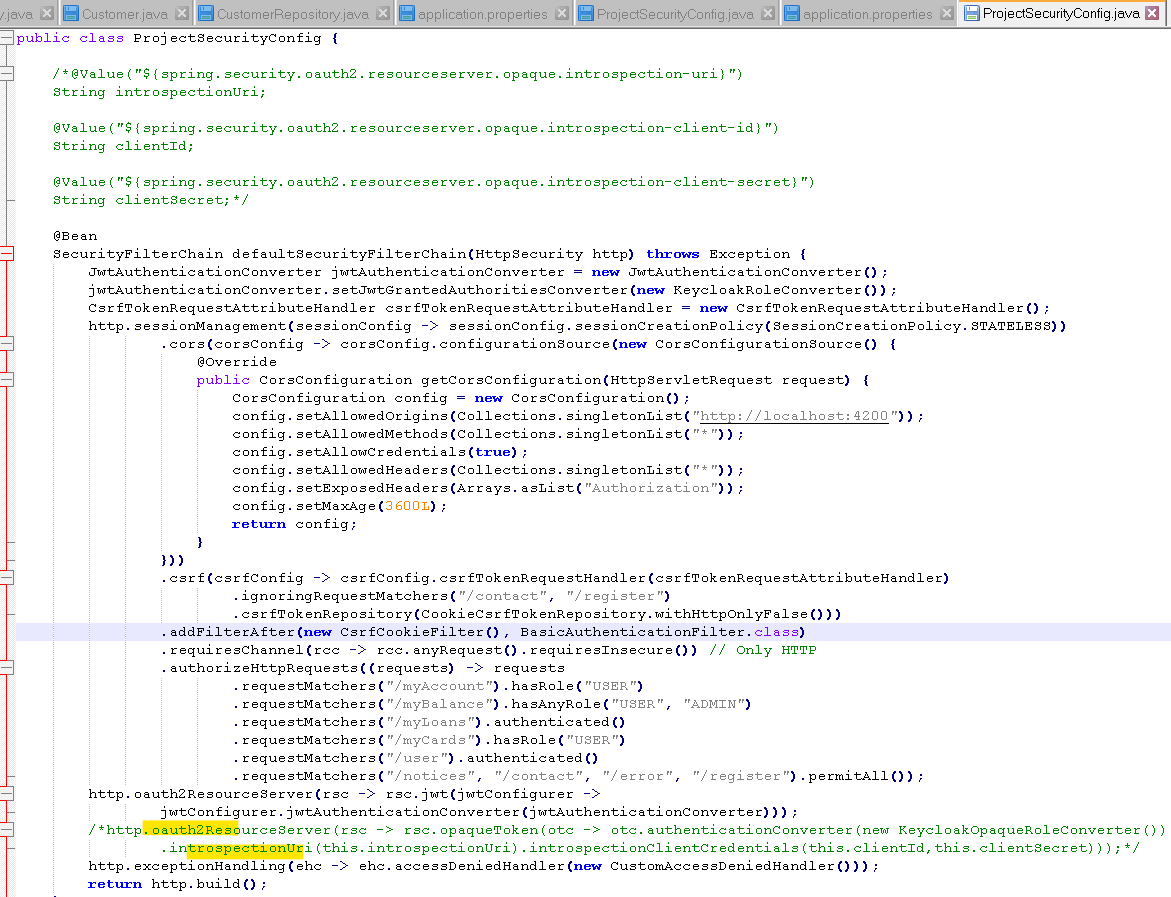
We already make these kind of changes when we are testing the Keycloak changes.

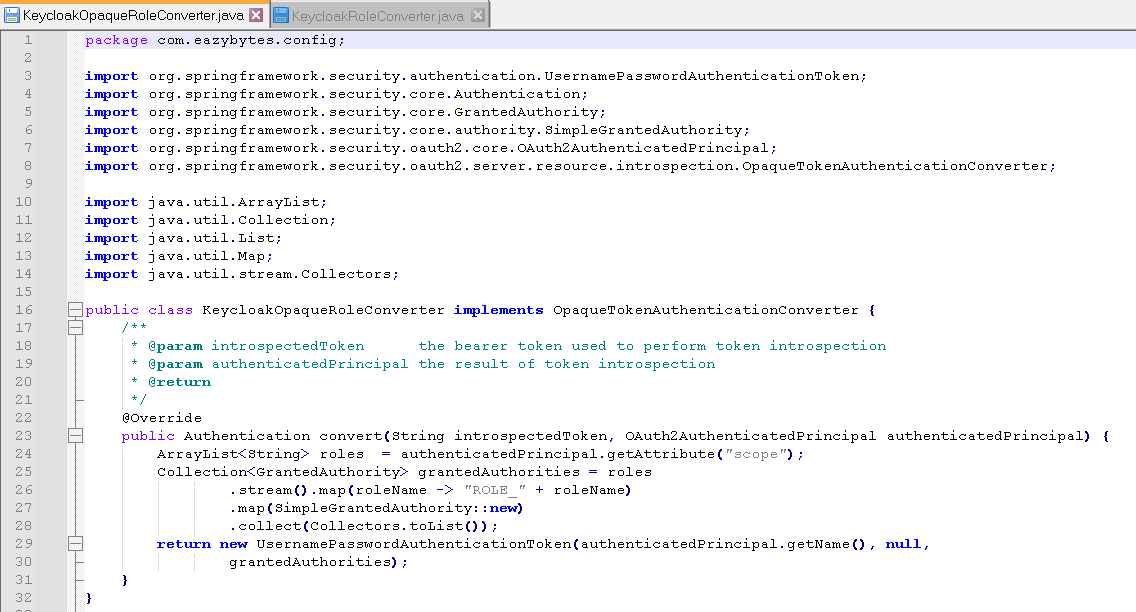


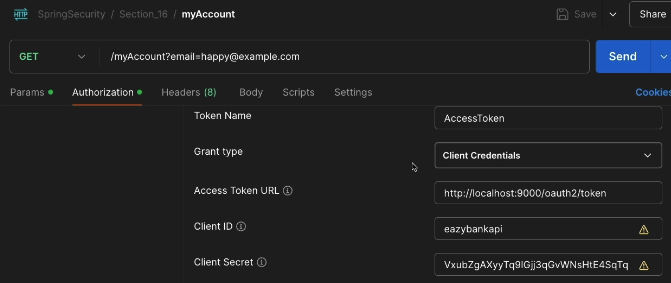
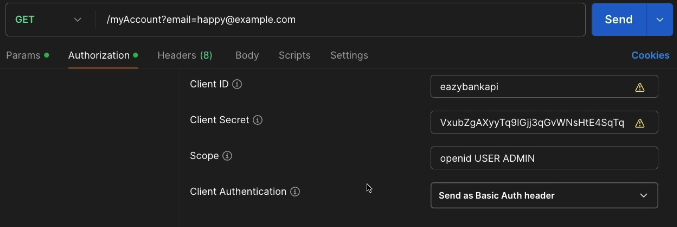
Next, we need to configure the client ID and client secret.

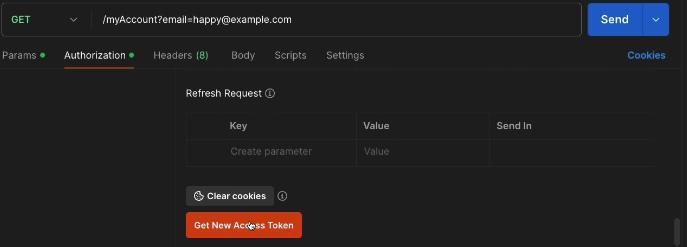
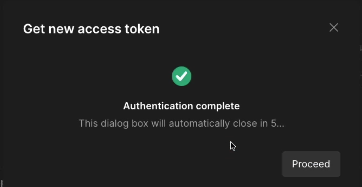
So using these client ID and client secret, my resource server is going to invoke the introspection URI. But as of now, these client details are not configured inside the OAuth server. So let me go to the OAuth server, register IntrospectClient.

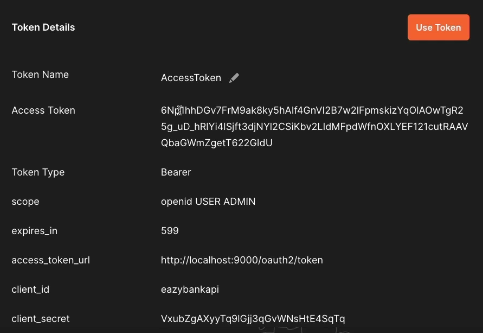






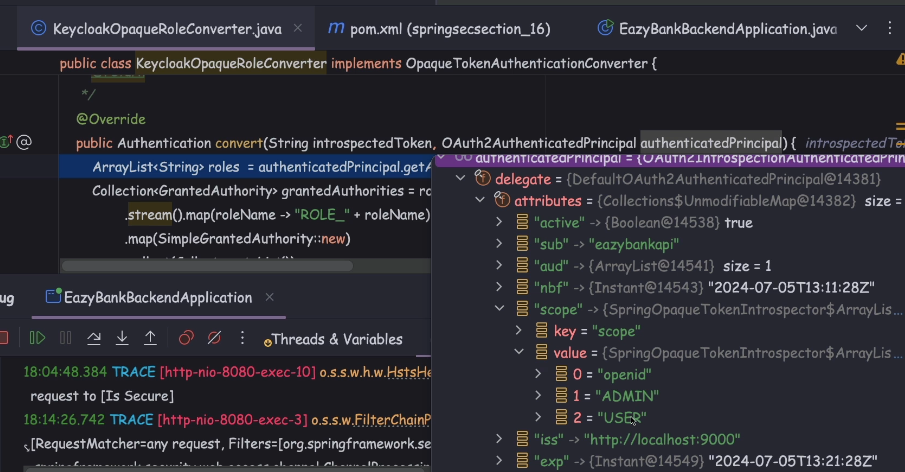


I got a new access token, but this time, you can see, it's not a JWT token.

It is a random String value. So if I try to take this access token value

and paste it here inside the jwt.io, I'm going to get an error, because this is not a JWT token. This is a simple string value, which is an opaque token.

Use the token and send



Here, you may have a question which is inside the KeycloakOpaqueRoleConverter,

why I'm trying to read from the scope, but not from the roles?

Because inside the OAuth server, we have made these configurations

to read the roles information from the roles claim. Then why I'm trying to read from the scope?

That can be a question.

**So whatever, jwtTokenCustomizer() method that we have written here,**

**this will not be applicable for the opaque tokens. So with that, this code is never going to be executed.**

That's why I'm not using the role to read the roles information.

Instead, I'm trying to use the scope to read the scope or roles information.

Since this logic is working fine for me, I didn't write any customizer method

to handle the opaque token scenario.

But I don't recommend you using this opaque token scenario.

Always try to use the self-contained or JWT token format.

Top of Form

What is the primary purpose of Spring Authorization Server?

* 

**To manage database access for Spring applications**

* 

**To provide centralized authentication services for microservices**

* 

**To monitor server performance metrics for Spring applications**

* 

**To manage OAuth 2.0 and OpenID Connect (OIDC) flows and issue tokens**

**Correct**

Top of Form

What is a key advantage of using Spring Authorization Server over traditional OAuth 2.0 implementations?

**It offers extensibility through Spring's ecosystem and integration with Spring Security**

Top of Form

Which Maven dependency is required to include Spring Authorization Server functionality in a Spring Boot application?

* 
  1. <dependency>
  2. <groupId>org.springframework.boot</groupId>
  3. <artifactId>spring-boot-starter-oauth2-authorization-server</artifactId>
  4. </dependency>

**Correct**

* 
  1. <dependency>
  2. <groupId>org.springframework.boot</groupId>
  3. <artifactId>spring-boot-starter-oauth2-client</artifactId>
  4. </dependency>
* 
  1. <dependency>
  2. <groupId>org.springframework.boot</groupId>
  3. <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
  4. </dependency>

Bottom of Form

Bottom of Form