1. What is Keycloak and How can It help me?

2. How do I use keycloak to secure my application the Merce way?

2.1 Keycloak

2.1.1 Downloading Keycloak

2.1.2 Setting up Keycloak

2.1.2.1 Creating Administrator user

2.1.2.2 Creating a Realm.

2.1.2.3 Creating a Keycloak Client.

2.1.2.4 Creating a Keycloak User

2.1.2.5 Setting Roles in Keycloak

- 2.1.2.6 Adding Role to the User
- 2.2 Testing the Keycloak setup using Postman
- 2.3 Business application
- 2.4 Test Authentication and Authorization

Reading and understanding the official documentation is essential to installing and using Keycloak in a secure manner, we highly recommend you follow the detailed information there to tune the installation and implementation to your specific use case.

Please use this document as a guide.

Some prerequisites for one to utilize this document effectively is basic understanding of Java, Spring Boot and REST concepts.

The reader should also have a basic understanding of OAuth2.0. One good reference for OAuth2.0 is <u>https://auth0.com/intro-to-iam/what-is-oauth-2</u>

# 1. What is Keycloak and How can It help me?

Keycloak is an open-source identity and access management solution that provides authentication, authorization, and single sign-on capabilities for web applications and services. It allows you to secure your applications by managing user identities, enforcing access controls, and facilitating seamless user authentication across multiple systems.

More details about keycloak and its capabilities are on Keycloak's website <a href="https://www.keycloak.org/">https://www.keycloak.org/</a>

Keycloak itself is written in Java and is completely open source. Its code is hosted on Github and is present here on <a href="https://github.com/keycloak/keycloak">https://github.com/keycloak/keycloak</a>

Javadocs are available for those who are interested on the URL <u>https://www.keycloak.org/docs-api/21.1.1/javadocs/index.html</u>

Keycloak also exposes a REST based Admin API via which one can manage all activities of Keycloak. Reference: <u>https://www.keycloak.org/docs-api/21.1.1/rest-api/index.html</u>

# 2. How do I use keycloak to secure my application the Merce way?

One approach to secure **Spring/Spring Boot** applications is what we'll discuss here.

There are a few pieces that we need to understand before we begin with this journey.

**Keycloak** supports multiple authorization frameworks including OpenID Connect, OAuth 2.0 and SAML 2.0. (Ref: <u>https://www.keycloak.org/</u>)

**OAuth2.0** (Ref: <u>https://auth0.com/intro-to-iam/what-is-oauth-2</u>) is an authorization framework that allows applications to access and use resources on behalf of a user without requiring the user to share their credentials (such as username and password) with the application. It provides a secure and standardized way for users to grant permissions to third-party applications to access their protected resources.

**Spring Security** is a powerful and highly customizable security framework for Java applications, specifically those built on the Spring framework.

It provides a comprehensive set of features and APIs to handle authentication, authorization, and other security-related tasks in a Java application.

It is the de-facto standard for securing Spring-based applications.

(Ref: https://spring.io/projects/spring-security)

So, we'll now use our Spring boot based code with Spring Security using the OAuth2 Framework and Keycloak Server to secure our application.

Keycloak is a separate server that is managed on our network. Applications are configured to point to and be secured by this server.

Browser applications redirect a user's browser from the application to the Keycloak authentication server where they enter their credentials. This redirection is important because users are completely isolated from applications and applications never see a user's credentials.

Applications instead are given an identity token or assertion that is cryptographically signed. These tokens can have identity information like username, address, email, and other profile data. They can also hold permission data so that applications can make authorization decisions. These tokens can also be used to make secure invocations on REST-based services.

# 2.1 Keycloak

### 2.1.1 Keycloak Core Concepts and Terms:

#### • Users

- Users are entities that are able to log into your system.
- They can have attributes associated with themselves like email, username, address, phone number, and birthday.
- They can be assigned group membership and have specific roles assigned to them.
- Roles
  - Roles identify a type or category of user.
  - Admin, user, manager, and employee are all typical roles that may exist in an organization.
  - Applications often assign access and permissions to specific roles rather than individual users as dealing with users can be too fine-grained and hard to manage.
- User role mapping
  - A user role mapping defines a mapping between a role and a user. A user can be associated with zero or more roles.
  - This role mapping information can be encapsulated into tokens and assertions so that applications can decide access permissions on various resources they manage.
- Realms
  - A realm manages a set of users, credentials, roles.
  - A user belongs to and logs into a realm.
  - Realms are isolated from one another and can only manage and authenticate the users that they control.
- Clients
  - Clients are entities that can request Keycloak to authenticate a user.
  - Most often, clients are applications and services that want to use Keycloak to secure themselves and provide a single sign-on solution.

 Clients can also be entities that just want to request identity information or an access token so that they can securely invoke other services on the network that are secured by Keycloak.

There are more concepts which are good to know. These can be found in the official documentation of Keycloak on <u>https://www.keycloak.org/docs/latest/server\_admin/#core-concepts-and-terms</u>

## 2.1.2 Downloading Keycloak

Keycloak works on almost all Linux based distribution and windows.

For our case, since most of us developers are on Ubuntu, we'll proceed with Basic JDK based setup.

The minimum system and software requirements are updated on the Keycloak website. Please refer to it before proceeding with installation.

https://www.keycloak.org/getting-started/getting-started-zip

Basic steps are:

- 1. Download the keycloak zip file.
- 2. Extract the zip file to some folder. (e.g. unzip keycloak-21.1.1.zip). Note: at the time of writing this doc, the latest version was 21.1.1, so unzip file keycloak zip file accordingly.
- 3. Start Keycloak. (bin/kc.sh start-dev)

Note that Keycloak by default starts on port **8080**. Ensure it's available.

Note: There are container images also available if you are comfortable with containers. Docker: <u>https://www.keycloak.org/getting-started/getting-started-docker</u> Kubernetes: <u>https://www.keycloak.org/getting-started/getting-started-kube</u>

## 2.1.3 Setting up Keycloak

2.1.2.1 Creating Administrator user

1) Open : <u>http://localhost:8080/</u>

	<b>O</b> IKEYCLOAK			
	Welcome to <b>Keycloak</b>			
	Administration Console Please create an initial admin user to get	Documentation >	Keycloak Project >	
	Username admin Pessword ****		Mailing List >	
-	Person confirmation		🏦 Report an issue 🤉	

- 2) Fill in the form with your preferred username and password.
- Now go to the default admin console <u>http://localhost:8080/admin</u> And Login with username and password created earlier.

#### 2.1.2.2 Creating a Realm.

A realm is a space where you manage objects, including users, applications, roles, and groups. A user belongs to and logs into a realm. One Keycloak deployment can define, store, and manage as many realms as there is space for in the database.

Realms are isolated from one another and can only manage and authenticate the users that they control. Following this security model helps prevent accidental changes and follows the tradition of permitting user accounts access to only those privileges and powers necessary for the successful completion of their current task.

You create a realm to provide a management space where you can create users and give them permissions to use applications. At first login, you are typically in the master realm, the top-level realm from which you create other realms.

You can also consider Realm to be a 'Tenant'

When deciding what realms you need, *consider the kind of isolation* you want to have for your users and applications.

For example, you might create a realm for the employees of your company and a separate realm for your customers. Your employees would log into the employee realm and only be able to visit internal company applications. Customers would log into the customer realm and only be able to interact with customer-facing apps.

Another way we can think of creating a realm is by Business Entity for example:

MerceRealm : for Merce internal applications. NSDLRealm : for NSDL applications. CDSLRealm : for CDSL applications. ProteanRealm : for Protean applications.

Key idea is to decide the isolation needed for the use case.



Example of how this can look:

Out of the box, Keycloak includes a single realm, called 'Master' realm.

Master realm - This (default) realm is created during the first Keycloak installation. It contains the administrator account you created at the first login. By convention, we'll use the master realm only to create and manage the realms in our system.

Other realms - These realms are created by the administrator in the master realm. In these realms, administrators manage the users in your organization and the applications they need. The applications are owned by the users.

Use the following steps to create the first realm.

- 1) Open the Keycloak Admin Console.
- 2) Click the word master in the top-left corner, then click Create realm.
- 3) Enter myrealm in the Realm name field.
- 4) Click Create.

#### 2.1.2.3 Creating a Keycloak Client.

A Keycloak client refers to an application or service that interacts with the Keycloak server to obtain authentication and authorization services.

It represents a registered entity that wants to utilize Keycloak's features, such as user authentication, access control, and single sign-on.

So, we can have a keycloak client for our Spring boot application, another client for say our PHP application and another client for say our Front end application.

Again, like Realm, there is no defined way to use a Client in Keycloak. Since it is like a framework, you can decide to use Client in ways you may think is more feasible.

One way to use a Client is to pair it with the type of application since there could be different requirements of a Front-end application as compared to a backend application.

Steps to create Keycloak Client are as follows:

- 1) Open the Keycloak Admin Console.
- 2) Click on master on top right corner and select Realm name, in our case it is myrealm
- 3) Click on 'Clients' in the menu bar on right
- 4) Click on 'Create Client' button
- 5) Fill in the details as following

Clients > Create client	Clients > Create client					
Create client Clients are applications and serv	ices that can request auth	entication of a user.				
<ol> <li>General Settings</li> </ol>	Client type ③	OpenID Connect				
	Client ID * ③	myclient-sb				
	Name ⑦	myclient-sb				
	Description ③	my client for <u>springboot</u> app				
	Always display in console ⑦	On				

- 6) Click on Next
- 7) Enable Client Authentication, Authorization as following screenshot:

Clients > Create client			
Create client Clients are applications and service	es that can request auther	ntication of a user.	
<ol> <li>General Settings</li> <li>Capability config</li> </ol>	Client authentication ( Authorization () Authentication flow	<ul> <li>On</li> <li>On</li> <li>On</li> <li>Standard flow (?)</li> <li>Implicit flow (?)</li> <li>OAuth 2.0 Device Authoriz</li> <li>OIDC CIBA Grant (?)</li> </ul>	<ul> <li>✓ Direct access grants ⊕</li> <li>✓ Service accounts roles ⊕</li> <li>action Grant ⊕</li> </ul>

- 8) Click on Save.
- 9) Now, if you go to the 'Credentials' tab you will see the client secret as follows:

		eques	t authentication o	of a user.			
Settings Keys	Credentials	les	Client scopes	Authorization	Service accounts roles	Sessions	Advanced
ীient Authenticator ত	Client Id and Secre	et					•
	Save						
Client secret					ø	li i	Regenerate
lient secret					Ø	je F	Regenerate
lient secret					Ø		Regenerate

We'll use this '**Client secret**' while connecting to Keycloak. You have now created a Keycloak Client for the spring boot app.

#### 2.1.2.4 Creating a Keycloak User

A keycloak user is the user who uses your application. Any user that will use your application, will have to be created in Keycloak. Keycloak will manage the user lifecycle. Following are the steps to create a user in Keycloak:

- 1) Open the Keycloak Admin Console.
- 2) Click on master on top right corner and select myrealm
- 3) Click on 'Users' in the menu bar on the right
- 4) Click on the 'Add user' button.
- 5) Fill in details as following:

Users > Create user	
Create user	
Username *	myuser
Email	myuser@merce.co
Email verified ①	Off Off
First name	my
Last name	user
Required user actions	Select action 🔹
Groups ⑦	Join Groups
	Create

And Click on 'Create'.

6) Now go to Credentials tab and click on 'Set password'

Users > User details					
myuser	1				
Details Attribute Cr	edentials ble mapping	Groups Consents	Identity provider links	Sessions	
	]		0		
			No credenti	als	
		This user does n	ot have any credentials. Yo	u can set passwor	d for this user.
			Set password Credential Res	d	

7) Create a password for this user and click on Save

Credentials	Role mapping	Groups	Consents	Identity provider links		
				Đ		
	Set pas	ssword	for myuse	er	×	word for this user
	Password *	r.	••••		Θ	
	Password c	onfirmation	• ••••		Θ	
	Temporary	0	(	Off		
	Save	Cancel				

You have now created a user called 'myuser' in Keycloak.

- 8) We'll also create another user 'myAdminUser' using the same step as above.
- 9) So now, we will see two users:

Users are the users in the current realm. Learn more 🗹		
Q Search user → Add user Delete user		
Username	Email	Last name
myadminuser	• myadminuser@merce.co	user
myuser	myuser@merce.co	user
service-account-myclient-sb	0-	-

#### 2.1.2.5 Setting Roles in Keycloak

For the authorization part in our spring boot application, we'll need to create different roles.

A role refers to a predefined set of permissions or access rights that can be assigned to users or clients. Roles are used to control and enforce authorization policies within the OAuth2 framework.

By assigning roles to users or clients, you can determine what actions they are allowed to perform and what resources they can access.

There are roles to be created at two levels, Keycloak Client level and Keycloak Realm level. We'll create roles at "myclient-sb" client which we created above and another role at our "myrealm" realm level.

Then we'll convert our realm level role to a composite role so that whenever we create a user, we'll just need to add one realm level role.

- 1. Create Client level role
  - a. Open the Keycloak Admin Console.
  - b. Click on master on top right corner and select myrealm
  - c. Click on 'Clients' in the menu bar on right
  - d. Now click on "myclient-sb" Client we previously created.
  - e. Click on the 'Roles' tab.
  - f. Click on the 'Create role' button.
  - g. Add role name as 'admin'

Clients > Client details > Create role				
Create role				
Role name *	admin			
Description	admin: Client level role			
	Save Cancel			

- h. Click on Save.
- i. Now create another role 'user' using the same step as above.

Clients > Client details > Create role Create role		
Polo pomo *	unar	
Role name	user	
Description	user : Client level role	
	Save Cancel	

j. Now under the client -> Roles tab we can see 2 custom roles we created as follows:

Clients > Client myclient-s Clients are appli	details ; <b>b</b> Openl ications ar	D Connect nd services that	can reque	est authentication /	of a user.					Enable
Settings	Keys	Credentials	Roles	Client scopes	Authorization	Service accounts roles	Sessions	Advanced		
Q. Search role b	oy name	$\rightarrow$	Create ro	ble						
Role name					Composite		Description	n		
admin					False		admin: Cliev	nt level role		
uma_protection					False		-			
user					False		user : Clien	t level role		

- 2. Create Realm level role
  - a. Open the Keycloak Admin Console.
  - b. Click on master on top right corner and select myrealm
  - c. Click on 'Realm roles' in the menu bar on right
  - d. Click on 'Create role' button
  - e. Create a role 'app-admin' as follows:

Realm roles > Create role				
Create role				
Role name *	app-admin			
Description	app-admin: Realm level role			
	Save Cancel			

- f. Click on save.
- g. Create a role 'app-user' as follows:

Realm roles > Create role					
Create role					
Role name *	app-user				
Description	app-user: Realm level role				
	Save Cancel				

h. Now, under Realm Roles, we can see 2 custom roles we created above as follows:

Realm roles Realm roles are the roles that you define for use in the current realm. Learn more 🕑						
Q. Search role by name → Create role		1-5 * 《 >				
Role name	Composite	Description				
app-admin	False	app-admin: Realm level role				
app-user	False	app-user: Realm level role				

Note in above screenshot, we can see that under the "Composite" column, roles are termed as 'False', which means they are not a composite role at this point.

- 3. Convert Realm level role to a Composite role
  - a. To convert Realm role to Composite role, we'll select a role 'app-admin'
  - b. Click on Action on top right corner and click on 'Add associated roles' as follows:

h i	
	Action
s Users in role Permissions	Add associated rol
	Delete this role
app-admin	
app-admix: Realm level role	
Seve Revert	
	app-admin: Realm level role

c. Here, from the drop down, select 'Filter by clients' as follows:

Realm role	s > Role details				
app-a	dmin				
Deta"	Assign roles to app-admin account			×	٦
Role na	▼ Filter by realm roles	<i>→</i>	1-5 *	< >	
Descrip	Filter by clients	Description			
	app-admin	app-admin: Realm level role			
	app-user	app-user: Realm level role			
	default-roles-myrealm	\${role_default-roles}			
	offline_access	\${role_offline-access}			
	uma_authorization	\${role_uma_authorization}			
			1-5 *	$\langle \rangle$	
	Assign Cancel		_		J

d. And select 'admin' client level role we previously created as follows and click on 'Assign' button:

Assign roles to app-admin account	×
<b>T</b> Filter by clients $\bullet$ <b>Q</b> admin <b>X <math>\rightarrow</math></b>	1-2 👻 < >
Name         Description	
✓ myclient-sb admin admin: Client level	role
realm-management realm-admin \${role_realm-admin	n}
	1-2 ▼ 〈 >
Assign Cancel	

e. You can now see the role is now a Composite role, whereby the 'Composite' column is visible as 'True'.

Realm roles Realm roles are the roles that you define for use in the current realm. Learn more 🗹					
Q Search role by name → Create role					
Role name	Composite	Description			
app-admin	True	app-admin: Realm level role			
app-user	False	app-user: Realm level role			

- f. We'll repeat the steps for 'app-user role to convert it to a composite role
- g. Click on the role 'app-user', Click on 'Action' on top right corner and select 'Add associated roles'
- h. Select 'Filter by clients' in the drop down

Realm roles	> Role details					
app-us	ser					
Deta"	Assign roles to app-u	ser account				×
Role na	▼ Filter by realm roles ▼	<b>Q</b> Search by role name	<i>&gt;</i>	1-5 👻	<	>
Descrip	Filter by clients		Description			
	app-admin		app-admin: Realm level role			
	app-user		app-user: Realm level role			
	default-roles-myrealm		\${role_default-roles}			
	offline_access		\${role_offline-access}			
	uma_authorization		\${role_uma_authorization}			
				1-5 👻	<	>
	Assign Cancel					

i. Select client level role 'user'

Realm roles	> Role details		
app-us	er		
Details	Attributes Users in role Permissions		
Role na	Assign roles to app-user account		×
Descrip	$\mathbf{Y}$ Filter by clients $\mathbf{Q}$ user $\mathbf{X}$	1-4 💌	< >
	Name	Description	
	realm-management manage-users	\${role_manage-users}	
	realm-management query-users	\${role_query-users}	
	✓ myclient-sb user	user : Client level role	
	realm-management view-users	\${role_view-users}	
		1-4 *	< >
	Assign Cancel		

- j. Click on Assign to save the role
- k. Now we can see both roles are composite role as follows:

Realm roles are the roles that you define for use in the current realm. Learn more 🗹					
Q Search role by name → Create role					
Role name	Composite	Description			
app-admin	True	app-admin: Realm level role			
app-user	True	app-user: Realm level role			

#### 2.1.2.6 Adding Role to the User

We'll now add the composite role we created to the user so that role will be a part of the user's authorization parameters i.e. it'll be a part of users Access Tokens.

- 1) Open the Keycloak Admin Console.
- 2) Click on master on top right corner and select myrealm
- 3) Click on 'Users' in the menu bar on the right
- 4) Select 'myuser' the user we previously created
- 5) Click on 'Role mapping' tab
- 6) Click on 'Assign role' button as follows

Users > User details							
myuser							
Details Attributes Crea	dentials Role mapping	Groups Consents	Identity provider links	Sessions			
<b>Q</b> Search by name	Q     Search by name     →     ✓     Hide inherited roles     Assign role     Unassign						
Name		Inherited		Description			
default-roles-myrealm		False		\${role_default-ro	les}		

7) Select 'app-admin' composite role we created in the previous step as follows:

Assign roles to myuser	account				×
▼ Filter by realm roles ▼	${\bf Q}_{\rm c}$ Search by role name	$\rightarrow$	1-4 💌	< 1	>
Name		Description			
app-admin		app-admin: Realm level role			
app-user		app-user: Realm level role			
offline_access		\${role_offline-access}			
uma_authorization		\${role_uma_authorization}			
			1-4 👻	< .	>
Assign Cancel					

- 8) Click on 'Assign' button to assign the role
- 9) Now you can see the role is assigned to the user 'myuser' under the 'Role mapping' tab

Role mapping	Groups Consents	Identity provider links	Sessions		
Q Search by name → ✓ Hide inherited roles Assign role Unassign					
	Inherited	C	Description		
	False	a	pp-user: Realm level role		
	False	\$	{role_default-roles}		
	Role mapping	Role mapping     Groups     Consents       Hide inherited roles     Assign role       Inherited     False       False     False	Role mapping       Groups       Consents       Identity provider links         Hide inherited roles       Assign role       Unassign         Inherited       Inherited       Inherited         False       a       False       s		

10) Now similarly using the same steps as above, we'll add 'app-admin' role to user 'myadminuser'

Users > User details						
myadminuser						
Details Attributes Credentials	Role mapping	Groups	Consents	Identity provider links	Sessions	
Q Search by name → ✓ Hide inherited roles Assign role Unassign						
Name		h	nherited	De	scription	
app-admin		F	alse	app	o-admin: Realm level role	
default-roles-myrealm		F	alse	\${n	ole_default-roles}	

We have now successfully configured Keycloak Users with associated roles.

## 2.2 Testing the Keycloak setup using Postman

I am assuming at this point that you have the 'Postman' app installed on your local machine. If not, please google the step to install Postman based on your device.

We'll connect to Keycloak to fetch 'Access token'.

I am attaching the Postman collection here for the reference. However, we'll create a new connection as follows:

Keyo	łoak-Merce Way / Auth admin			🖺 Save 🗸 🚥	1	P
POS	T  V http://iocalhost:8080/realms/myrealm/protocol/openid-connect/	token			Sen	d ~
Paran	is Authorization Headers (8) <b>Body •</b> Pre-request Script Tests					
le no	ne 🔵 form-data 🐞 x-www-form-urlencoded 🔵 raw 🔵 binary 🔵 Gra	phQL				
	Кеу	Value	Description		000	Bulk Edit
~	client_id	myclient-sb				
~	client_secret	EvP10E952ivNyy6MeCpBWGjJciPEQjQV				
~	username	myadminuser@merce.co				
~	password	mailpass				
~	grant_type	password				
~	scope	openid				

Note the URL: <u>http://localhost:8080/realms/myrealm/protocol/openid-connect/token</u>

Here: localhost : It is the host where Keycloak is running

8080: Port on which Keycloak is listening

'Myrealm' is the realm we created above.

Following are the parameters which we add in the request body:

- client\_id: ID of the keycloak client we created above
- client\_secret : client secret generated by Keycloak [as seen in step 9 of 2.1.2.3]
- username : username of the userid trying to login to keycloak
- password : password of the userid
- grant\_type : this can be 'client\_credentials' or 'password'. We'll use 'password'
- scope : 'openid'

If all configuration is correct, on sending this request, keycloak will respond with 'access\_token' and 'refresh\_token' alongwith expiry and other parameters.

PDST     Intpull/ocahonta/000/wams/myream/protocol/opani-connect/book     Send of an anti-anti-anti-anti-anti-anti-anti-anti-	Keycloak-Merce Way / Auth admin		🖺 Save 🗸 👓 📃					
Authorization       Headers (B)       Body       Perception for task = binary       Control                more              more in more data              maxe-norm-untendededededededededededededededededede	POST v http://localhost:8080/realms/myrealm/protocol/openid-connect/	POST v http://localhost.8080/realms/myrealm/protocol/openid-connect/token						
Norm • Norm-data • xwww-tom-unkneepede • Na	Params Authorization Headers (8) Body Pre-request Script Tests							
Key     Value     Description     ••• Bake Ed       •     clent_Ld     myclent-sb     ••• CP10E9520x4yygMcCpBW0jLcPEQQV     ••• CP10E9520x4yygMcCpBW0jLcPEQQV       •     clent_Ldc     ExP10E9520x4yygMcCpBW0jLcPEQQV     ·•• CP10E9520x4yygMcCpBW0jLcPEQQV       •     usename     myclent-sb       •     password     malpass       •     gent_type     password       •     password     oped       •     oped     oped       •     value     Description       •     password     oped       •     value     Description       •     value     Save Response       •     value     Save Response       <	🔵 none 🔹 form-data 🔵 x-www-form-urlencoded 🌑 raw 🌑 binary 🗨 Gra	aphQL						
cient_d       mylent-sb       cv106920/kjr86/cj8W3/kc/F6/O0V         cient_scoret       Ev106920/kjr86/cj8W3/kc/F6/O0V       c         cient_scoret       myladmiusergenerce.co       malpass         cient_yce       pasword       malpass         cient_yce       opend       c         cient_yce       opend       c         cient_yce       opend       c         cient_yce       opend       c         cient_yce       value       c         Body       Cookies       Headers (11)       Test Results       Save Response         retty       Raw       Preview       Vlaue       Cookies       Save Response       c         cient_yce       Naw       Preview       Vlaue       Save Response       c       c       c         cient_yce       Naw       Preview       Vlaue       Save Response       c       c	Кеу	Value	Description ••• Bulk Edit					
Central cent         EvP10EPS2ivMysMecCpBW03jLoPECQ0V         Central centra central central central central central central central central	Client_Id	myclient-sb						
vername       myadminuser@merce.co         malipass       malipass         vername       malipass         vername       password         malipass       openid         vername       openid         Vername       openid         New       Value         Description         Body       Cookies         Headers (11)       Test Results         Petty       Raw         Petty       Raw         Petty       Raw         Value       Description         Image: Status: 200 OK       Test: 3.98 KB         Save Response       Status: 200 OK         Image: Status: 200 OK       Test: 3.98 KB         Image: Status: 200 OK       Test: 3.9	Client_secret	EvP10E952ivNyy6MeCpBWGjJciPEQjQV						
i alignass        malipass             i grant_type        password             grant_type        password             grant_type        password             grant_type        password             grant_type        password             key        Value        Description             Body         Cookies         Headers (11)         Test Results	✓ username	myadminuser@merce.co						
Image: scope       password       openid         Image: scope       openid       Description         Image: scope       Openid       Openid       Openid         Image: scope       Image: scope       Openid       Openid         Image: scope       Openid       Openid       Openid       Openid         Image: scope       Image: scope       Openid       Openid       Openid         Image: scope       Value       Description       Image: scope       Openid         Image: scope       Image: scope       Scope       Image: scope <th< td=""><td>✓ password</td><th>mailpass</th><td></td></th<>	✓ password	mailpass						
copend     openda       Kay     Value       Body     Cookes     Headers (11)     Test Results       Pretty     Raw     Preview     Visualize     Jon       Pretty     Raw     Preview     Visualize     Jon       Image: Cookes     Headers (11)     Test Results     Status: 200 OK     Tme: 64 ms     Size: 3.98 KB     Save Response       Pretty     Raw     Preview     Visualize     Jon     Image: Cookes     Headers (11)     Size: 3.98 KB     Save Response       Pretty     Raw     Preview     Visualize     Jon     Image: Cookes     Headers (11)     Size: 3.98 KB     Save Response       Image: Cookes     Headers (11)     Test Results     Jon     Image: Cookes     Headers (11)     Size: 3.98 KB     Save Response       Image: Cookes     Headers (11)     Test Results     Jon     Image: Cookes     Headers (11)     Pretty     Raw     Preview     Visualize     Jon     Image: Cookes     Headers (11)	✓ grant_type	password						
Kay     Value     Description       Body     Cookles     Headers (11)     Test Results     Save Response v       Pretty     Raw     Preview     Visualize     JSON v     Image: Cookles     Save Response v       1     Image: Cookles     Headers (11)     Test Results     Save Response v     Image: Cookles     Image: Cookles     Save Response v       1     Image: Cookles     Headers (11)     Test Results     JSON v     Image: Cookles     Image: Cookles <td>✓ scope</td> <th>openid</th> <td></td>	✓ scope	openid						
Body       Cookes       Headers (11)       Test Results       Save Response         Pretty       Raw       Preview       Visualize       JSON        Image: Cookes       Save Response       Image: Cookes       Image: Cookes       Save Response       Image: Cookes       Save Response       Image: Cookes								
Pretty       Raw       Preview       Visualize       JSON V       Description       C         1       0       *access_token*: *eyhb6c10135Uz1INIISIn8cc1g01AKIL003VEF1HkVVSm8xbW%md2NVdHR5Ump1YXpzcDRxaUhmVV9VhZER29va2BR1n8.       eyileiAi0j22000007AkHTcsInahndCloHTYAM00A0Tg1NwitanRp1js1HZ/kMmZ201WhF1LaWZ21000Hx7UL004XVJC100KXUL001XVL01Hk1XAXL210L0AHR7CL012V22LV1D*266F025161j1In2F1Yz1m1TAMPXHKM60401YTKH1ku13XWZ1UMF1KH1XU11NU1111C1XXVX40040Hy11ane8c16161151XYZ204BeXL11NiA112XVZ4F04EXL11NL0132WgHXMU2020THk1TiAUXD201Hk1TiAUXD2020THk1TiAUXD201Hk1TiAUXD2020THk1TiAUXD201Hk1TiAUXD202	Body Cookles Headers (11) Test Results							
<pre> 2 3 *access_token*: *eyDbbGc1015ULIIMITETRESCIG01AtSIGUIIwia21kIIAGCJDVEFIH/WVSBbbW9d2NVdHR8UmpjYXpzeDRxaUhmVV9VN0ZEP29vzBBTn0. 2 *access_token*: *eyDbbGc1015ULIIMITETRESCIG01AtSIGUIIwia21kIIAGCJDVEFIH/WVSBbbW9d2NVdHR8UmpjYXpzeDRxaUhmVV9VN0ZEP29vzBBTn0. eyD1eHA10jE2000DTxMTCsIahDdCJMTYTCsIAhDdCJMTYL1kuli32Wg7UL00GUMyIaIneScifiaIIIaFecIfiaIIIS2/zw2GBUNJ1igiaIIC222221vDj9zdF625161j1IAFF1v1mILAM32Hj0ABCMCJ22221VDj9zdF625161j1IAFF1v1mILAM32Hj0ABCMCJ22221VDj9zdF625161j1IAFF1v1mILAM32Hj0ABCMCJ2221VJ120HVMF1zHiNJ22PMBCJ2221VJ120HVMF1zHiNJ22PMBCJ2221VJ120HVMF1zHiNJ22PMBCJ2221VD122221VD122221VD122221VD122221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD12221VD1222AM0VY22UV3BHILG22WV1222AM0VY22UV3BHILG22WV12V22AM0VY22UV3BHILG22WV12V22AM0VY2UV3BHILG22WV12V22AM0VY2UV3BHILG22WV12V22AM0VY2UV3BHILG22WV12V22AM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2ZAM0VY2UV3BHILG22WV12V2AM0VY2UV3BHILG22WV12V2AM0VY2UV3BHILG22WV12V2AM0VY2UV3BHILG2UV1UA3BH0BFB0HLC4AVW14W0ABCB1DV1UD32PMSBH0BBEB0HLC4AVW14W0AG7AM0W70UV2ZAM0VY2UV3BHILG2UV1UX3DH0V7UV4SZAM0VY2UV3BHILG2UV1UX3DH0V7UV4SZAM0VY2UV3BHILG2UV1UX3DH0V7UV4SZAM0VY2UV3BHILG2WV14VIIJ010HV10A3DH30HB0BFB0HLC4AVW14W0AG7AM0V7UV4SZAM0VY2UV3BHILG2WV14VIIJ01XH12FV7UXAG9P3gPX1rw3J1PK7NCXXXJHJH_g*, *epTres_hr: 300, *tefresh_exptres_hr: 1800, *tefresh_exptres_hr: 2000CH2V2F1saBhdC101V2XIIIINECU2C00BGUULThJMUVDG3zKJBUUJ9NCCnP3GEAF5GT6J5MmVKN6EffQ. *y31eHA10f2200Q0T2EXFUND3JVETcv9teXJ1VxtT1ini2XXIJ01AHK1EV2Z2UJUJ11XEUUZ20BMGVJ2ZMUUJ1IaiaNXL7j04HBC2DvL2XV2F2aG9Zd0AM0BguIJ31VMtcv9teXJ1VXtI1ini2XXIJ01AHBC1031V2XV2F36G9Zd0AM0BguIJ31VWtcv9teXJ1VXtI1iniXXKIJ01AHBC103HZ2MMUUJ13L2MNCD4YGZ</pre>	Pretty Raw Preview Visualize JSON ~ 🚍		🖻 Q					
o token_type : bearer ,	Pretty       Raw       Prevew       Visualize       Jon       Image: Control in the control in							

We'll use this 'access\_token' for all consecutive requests to authenticate yourself.

Pro-tip: You can check contents of this JWT token, using a site like '<u>http://jwt.io</u>' Following is the example:

JSON Web Tokens - jwt.io × +	✓ - □ ;
→ C 🖬 jwt.io	< 😒 🧤 🖬 🕲
	Libraries Introduction Ask Crafted by SauthO of by Okta
<pre>pjYXpzeDRxaUhmVV9VNnZER29vazBRIn0.eyJle HAi0jE20DQ00TAxNTcsImlhdCI6MTY4NDQ40Tg1 NywianRpIjoiM2FkNmIwZDItNWFhZi00NzM3LTh mYzEt0DYyYzcy0DEyMDExIiwiaXNzIjoiaHR0cD ovL2xvY2FsaG9zdDo4MDgwL3J1YWxtcy9teXJ1Y WxtIiwiYXVkIjoiYWNjb3VudCIsInN1YiI6ImVh ZD00HzlZiDtthUV20010111</pre>	<pre>"alg": "RS256", "typ": "JWT", "kid": "ITAb2EUJo1mopwcUudlRjcazsx4qiHfU_U6vDGook0Q" } PAYLOAD: DATA { {     "exp": 1684490157.</pre>
ZDc20TMzLTZiZDAtNDYx0C1iMT1kLWI3ZWQzMTU 40GU4NyIsInR5cCI6IkJ1YXJ1ciIsImF6cCI6Im 15Y2xpZW50LXNiIiwic2Vzc21vb19zdGF0ZSI6I j11NzF1Yz1mLTA0MzMtNGM0MC1iYTEwLWU4MWR1 NTVkMzhiMCIsImFjciI6IjEiLCJyZWFsbV9hY2N 1c3MiOnsicm9sZXMiOlsiZGVmYXVsdC1yb2x1cy 1teXJ1YWxtIiwib2ZmbGluZV9hY2N1c3MiLCJhc	<pre>cxp : 1034490137, "iat": 1084499857, "jti": "3ad6b0d2-5aaf-4737-8fc1-862c72812011", "iss": "http://localhost:8080/realms/myrealm", "aud": "account", "sub": "ead76933-6bd0-4618-b19d-b7ed31588e87", "typ": "Bearer", "azp": "myclient-sb", "session_state": "9e71ec9f-0433-4c40-ba10- e81de55d38b0", "acr": "1",</pre>
fSwicmVzb3VyY2VfYWNjZXNzIjp7Im15Y2xpZW5 0LXNiIjp7InJvbGVzIjpbImFkbWluI119LCJhY2 NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2Nvd W50IiwibWFuYWdlLWFjY291bnQtbGlua3MiLCJ2 aWV3LXByb2ZpbGUiXX19LCJzY29wZSI6Im9wZW5 pZCBwcm9maWx1IGVtYWlsIiwic21kIjoi0WU3MW	<pre>"realm_access": {     "roles": [         "default-roles-myrealm",         "offline_access",         "app-admin",         "uma_authorization"     ]     },     "resource_access": {</pre>
VjOWYtMDQzMy00YzQwLWJhMTAtZTgxZGU1NWQzO GIwIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJu YW11IjoibX1hZG1pbiB1c2VyIiwicHJ1ZmVycmV kX3VzZXJuYW11IjoibX1hZG1pbnVzZXIiLCJnaX	<pre>"myclent-s0": {     "roles": [     "admin"     ] },     "account": {     "account": {     "account": {</pre>

As you can see in the above screenshot, under 'resource\_access'>'myclient-sb'>'roles'>'admin' Here we can see the client name we created in Keycloak, along with the role of the user 'admin' We'll use this role for the authorization part in the steps ahead.

# 2.3 Business application

We'll now create a Spring boot application that will use Spring Security to secure the application via OAuth and it'll work with Keycloak for Authentication and Role level Authorization.

You can find the entire working code on Github link [<u>https://github.com/merce-bhavyag/sb-kc-demo</u>]

Go to <u>https://start.spring.io/</u> and we'll get a new spring boot application. *Note*: You can also use the latest Spring Boot 3.1.0 as well

🔒 sta	rt.spring.io									: < ☆	£
				Meet the Sp	pring team th	is August at SpringOne.					
≡	💋 spri	ng initializr									
	Project Gradle - Groo Maven Spring Boot S.1.1 (SNAPS C.7.13 (SNAP	vy O Gradle - Kotlin (HOT) O 3.1.0 O 3.0.4 SHOT) O 2.7.12	Language Java O Kotin O 8 (SNAPSHOT) 3.0.7	O Groovy		Dependencies OAuth2 Resource Spring Boot integratio Spring Security Highly customizable a	Server SECURITY n for Spring Security's SECURITY uthentication and acc	OAuth2 resource se ess-control framewo	ADD DEPENDENCIES	. CTRL + B	
	Project Metada Group	co.merce				OAuth2 Client Spring Boot integratio	ECURITY n for Spring Security's	OAuth2/OpenID Co	onnect client features.		
	Artifact	sb-kc-demo-app									
	Name	sb-kc-demo-app									
	Description	Demo project for Spring Boot									
	Package name	co.merce.sb-kc-demo-app									
	Packaging	Jar O War									
	Java	O 20 • 17 O 11	O 8								
() Y			GENERATE	CTRL + 🖻	EXPLO	RE CTRL + SPACE	SHARE				

And we'll click on "GENERATE"

This will download a zip archive "sb-kc-demo-app.zip"

We'll now use Either Eclipse or STS(Spring Tool Suite) to use this downloaded application.

For this guide, I'm using STS, but steps should be the same for Eclipse.

Extract the ZIP file to a folder

Open STS, Click on File>'Open Projects from File System' > Browse for the Zip file folder we downloaded from Spring Initilizr

_								
		Import Projects	from File Syste	m or Archive			0	
Ve	Import Projects fron	n File System or Archive					_	
	This wizard analyzes th	he content of your folder or archive fi	ile to find project	s and import the	em in the	IDE.		
9	Import source: //ho	me/bhavyag/Code/sb-kc-demo-app				Directory	Archive	
9	type filter text					Sele	ect All	
	Folder			Import as		Dese	elect All	
ə	✓ sb-kc-demo-app	þ				1 of 1 selected	j dy open proje	ects
Ð	Close newly impor Use <u>installed project c</u> Search for nested Detect and configu	ted projects upon completion <u>configurators</u> to: projects ure project natures						
P	Working sets	vorking sets				(	New	
	Working sets:					~	Select	D
€					<u>Show ol</u>	<u>cher specialized</u>	<u>d import wiza</u>	<u>ırds</u>
rc or	Ø		< Back	Next >		Cancel	Finish	

So now, the application will open in the STS and will look as follows:



We'll create two packages 'config' and 'controller' within the 'co.merce' package We'll create config and controllers files under the respective packages



We'll create a new class 'WebSecurityConfig.java' under the config package.

This 'WebSecurityConfig' file will host all the configuration required for securing the springboot application.

We'll add the following annotations to the class:



@Configuration : Will mark the class as a configuration for spring boot.

@EnableWebSecurity : This will enable Spring Web Security for the application @EnableMethodSecurity(securedEnabled = true, jsr250Enabled = true) : This will enable Method based security annotations and Spring will now look for "@Secured" annotation on methods and will secure the method accordingly.

Here we'll add a bean to manage the HTTP requests the spring boot application will receive.

SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
 http.authorizeHttpRequests(requests -> requests
 /+

```
.anyRequest()
            .authenticated()
http.oauth2ResourceServer(oauth2 -> oauth2
jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthConverter))
);
http.sessionManagement(sessionManagement ->
sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            );
```

Here, if you note, we have our custom 'jwtAuthConverter' class that will extract the role information from the JWT token of the request.

Payload of a Decoded JWT token looks like this: { "exp": 1684504093, "iat": 1684503793, "jti": "47fd2933-be33-4fba-bc98-2a83db11a80a", "iss": "http://localhost:8080/realms/myrealm", "aud": "account",

```
"sub": "ead76933-6bd0-4618-b19d-b7ed31588e87",
 "typ": "Bearer",
 "azp": "myclient-sb",
 "session state": "f1c96b22-70eb-4f4c-9e06-f10d28c2cd94",
 "acr": "1",
 "realm access": {
  "roles": [
   "default-roles-myrealm",
   "offline access",
   "app-admin",
   "uma authorization"
  1
 },
 "<mark>resource_access</mark>": {
  "myclient-sb": {
   "roles": [
     "<mark>admin</mark>"
   ]
  },
  "account": {
   "roles": [
     "manage-account",
     "manage-account-links",
     "view-profile"
   ]
  }
 },
 "scope": "openid profile email",
 "sid": "f1c96b22-70eb-4f4c-9e06-f10d28c2cd94",
 "email verified": false,
 "name": "myadmin user",
 "preferred username": "myadminuser",
 "given name": "myadmin",
 "family name": "user",
 "email": "myadminuser@merce.co"
}
```

From the above token, we need to extract roles of our client "myclient-sb" which falls under "resource\_access"

Following is the method within the jwtAuthConverter that extracts roles as per above logic:

```
private Collection<? extends GrantedAuthority> extractResourceRoles(Jwt jwt) {
Map<String, Object> resourceAccess;
Map<String, Object> resource;
Collection<String> resourceRoles;
    resourceAccess = jwt.getClaim("resource_access");
    if (resourceAccess == null
|| (resource = (Map<String, Object>)
resourceAccess.get(properties.getResourceId())) == null
```



```
Of course there are other methods which we'll need in this JwtConverter as follows:
```

```
public AbstractAuthenticationToken convert(Jwt jwt) {
    Collection<GrantedAuthority> a =
    jwtGrantedAuthoritiesConverter.convert(jwt);
    Collection<? extends GrantedAuthority> b = extractResourceRoles(jwt);
    Collection<GrantedAuthority> authorities;
    if(a!=null) {
        authorities =
    Stream.concat(a.stream(),b.stream()).collect(Collectors.toSet());
    }else {
        authorities=b.stream().collect(Collectors.toSet());
    }
    return new JwtAuthenticationToken(jwt, authorities,
    getPrincipalClaimName(jwt));
}
```

And



We'll set the application.yml to set up the configuration as follows:



Ensure you set the correct client secret [as seen in step 9 of <u>2.1.2.3</u>] in "credentials.secret" parameter of application.yml

We'll now create a TestController and secure the methods via the roles as follows:

```
@RestController
@RequestMapping("/test")
public class TestController {
    private static final Logger logger =
LoggerFactory.getLogger(TestController.class);
    //@PreAuthorize("hasRole('ROLE_USER')")
    @Secured("user")
    @GetMapping(value = "/user")
    public ResponseEntity<String> getUser(Principal principal) {
        Logger.info("Hello form method User to user{}
",principal.getName());
        return ResponseEntity.ok("Hello form method User to User
"+principal.getName());
    }
    @Secured("admin")
    @GetMapping(value = "/admin")
    public ResponseEntity<String> getAdmin(Principal principal) {
        Logger.info("Hello from Admin");
        return ResponseEntity.ok("Hello from method Admin To user
"+principal.getName());
    }
    @Secured({ "user", "admin" })
    @GetMapping(value = "/all-user")
    public ResponseEntity<String> getAllUser(Principal principal) {
        Logger.info("Hello from All User");
        return ResponseEntity.ok("Hello from method All User to User
"+principal.getName());
    }
}
```

Now once we have everything set up, we can run the application and test if our authentication and authorization works.

Again, the entire code can be downloaded from the github repository [<u>https://github.com/merce-bhavyag/sb-kc-demo</u>]

## 2.4 Test Authentication and Authorization

We'll use the Postman application to test this as we did in Step 2 before. You can use the postman collection from the link below to test the setup. [https://github.com/merce-bhavyag/sb-kc-demo/blob/main/postman/Keycloak-Merce-Way.postm an collection.json ]

#### This is how the test looks:

Keycloak-Merce Way / Auth admin							
POST · http://localhost:8080/realms/myrealm/protocol/openid-connect/t	r v http://iocaihost:8080/realms/myrealm/protocol/openid-connect/token Send						
Params Authorization Headers (8) Body • Pre-request Script Tests	ns Authorization Headers (8) Body • Pre-request Script Tests • Settings Co						
🔵 none 🌑 form-data 🛛 a-www-form-urlencoded 🌑 raw 🌑 binary 🜑 Gra	phQL						
Кеу	Value	Description	••• Bulk Edit				
✓ client_id	myclient-sb						
Client_secret	EvP10E952ivNyy6MeCpBWGjJciPEQjQV						
✓ username	myadminuser@merce.co						
✓ password	mailpass						
✓ grant_type	password						
Scope	openid						
Key							
Body Cookies Headers (11) Test Results							
Pretty Raw Preview Visualize JSON V			🔳 Q				
<pre>3 *cccss_token*; *cylhbGc1015U:LINITGTR66cCIg01AIS1dUIisis21k1A6fCJ2VEF:MKVVSmBxbWMd2NVdMRdbmcjYV2cd0xdUmbVV9VNZER29vazBRIn0. *clcss_token*; *cylhbGc1015U:LINITGTR66cCIg01AIS1dUIisis21k1A6fCJ2VEF:MKVVSmBxbWMd2NVdMRdbmcjYV2cd0xdUmbV9VNZER29vazBRIn0. *vyleHA10jE2000jMDU300csImlhdClsANTVADUMVTQ4NymianRpIjo1MTkSHj1nYjUtOTdkNC00NzZ1LTKiYZEt001hN2YAMMYjNjYjIisiaXNZIjo1aHR0cDovL2xvY2Fsa69zdDodM0gwL3J1VWxtcy9teXJ1VWxtLy0tDiNS1Mw1Lik Nz*NkYSMyM2IIS1BF5;ciI01jELC3yAFsbV9NY2NL3H0nsicm9ZZMN01SiZ0WFXV8cd1b2Xkcj2teXJ1WkxtIisibZzmbG1UZVNYXDL3HLC3UV2NL2UNJ1g02tbF52bGUX294PV23L0HLC3UV0H01HS1MW3LW2V22V VMKjZNKZ1p7JTMISY2AZ9K6UNIjp7JTabVC4Z9EDVUTSVTDIJT0CC7NVKV0665jJ7TabVC4Z91DF1ThkmF3TjDTB1hbmF3T2D1F1DF1MbmF3T2D1PJTNVK065jJ7TJN0465J37D1mC44T28422001002hbH1LC32WW3LW52E2D0UXX9L02V2V2V VMKjZNKZ1p7JTMISY2AZ9K6UNIjp7JTabVC4Z9EDVUTSVTDIJT0CC7NVKV0665jJ7TabVC4Z9EDVIJSV1C4V1V3L04HULC32WW3LW52E2D0UXX9L02V2V2V2V2V2V2V2V2V2V2V2V2V2V2V2V2V2V2V</pre>							
4 "refresh_expires_in": 1800, 5 "refresh_token": "ev]bbGciOilTHzT1NiTcTnD5cCTdOiAiSIdHTim		0					

We'll pass the access\_token to subsequent requests So request to our endpoint "/admin" from myadminuser will work.

Keycloak-Merce Way / admin			🖺 Save 🗸 🚥	1
GET · http://localhost:8090/test/admin				Send ~
Params         Authorization         Headers (7)         Body         Pre-request Script         Tests           Headers         Ø Hide auto-generated headers         Image: Script Script Script         Tests         Script Scrip				
Кеу	Value	Description	••• Bulk Edit	Presets 🗸
Postman-Token 🕢	<calculated is="" request="" sent="" when=""></calculated>			
V Host (3)	<calculated is="" request="" sent="" when=""></calculated>			
User-Agent (1)	PostmanRuntime/7.32.2			
Accept (3)				
Accept-Encoding (1)	gzip, deflate, br			
Connection (3)	keep-alive			
Authorization	bearer eyJhbGciOlJSUzI1NiisinR5cClgOlAlSidUliwia2lkiIA6lCJJVEFiMkVV			
Body Cookles Headers (11) Test Results				
Pretty Raw Preview Visualize Text ~ =				ΓQ
1 Hello from method Admin To user myadminuser				

However, the same request to endpoint "/user" will not work

Keycloak-Merce Way / admin			Save 🗸 👓	
GET ~ http://localhost:8090/test/user				Send ~
Params         Authorization         Headers (7)         Body         Pre-request Script         Tests           Headers         Ø Hide auto-generated headers         Image: Script Script Script         Tests         Script Scrip				
Key	Value	Description	••• Bulk Edit	Presets 🗸
Postman-Token (3)	<calculated is="" request="" sent="" when=""></calculated>			
✓ Host ③	<calculated is="" request="" sent="" when=""></calculated>			
User-Agent (1)	PostmanRuntime/7.32.2			
Accept (1)				
C Accept-Encoding ()	gzip, deflate, br			
Connection (i)	keep-alive			
V Authorization	bearer eyJhbGciOiJSUzI1NiIsInR5cClgOiAiSIdUliwia2lkliA6lCJJVEFiMkVV			
Кеу				
Body Cookies Headers (11) Test Results				
Pretty Raw Preview Visualize Text V				🖻 Q
1				

As we can see in the response as "403 Forbidden"

Similarly, access\_token with user "myuser" will work with endpoint "/user" and will not work with endpoint "/admin"

Keycloak-Merce Way / Auth User		🖺 Save 🗸 🚥	Ø	Ē		
POST v http://localhost.8080/realms/myrealm/protocol/openid-connect/token						
Params Authorization Headers (8) Body Pre-request Script Test						
none form-data x-www-form-urlencoded raw binary G	raphQL					
Key	Value	Description	000 <b>E</b>	Bulk Edit		
✓ client_id	myclient-sb					
✓ username	myuser@merce.co					
✓ password	malipass					
grant_type	password					
✓ scope	openid					
Client_secret	EvP10E952ivNyy6MeCpBWGjJciPEQjQV					
Key						
Body Cookies Headers (11) Test Results						
Pretty Raw Preview Visualize JSON V 🚍				<b>a</b>		
<pre> 2 3 *access_token*: "eyJhbGci013SUz11N1i6InRScCIg0IAISIdUIIwia2IkIIA6IC33VEFIMkVVSm8xbW9md2NVdMR3UmpjYXpzeDRxaUhmVV9WnZER29vazBRIn0. ey31eHA10jE200Q1MY0MtyMsInIndcI6HTV4MDUmTc2PywatanPbj201kEtM0BmybEtM03 [Myb0WmR11119MtZa22jQ2MJU3YzhIIwiaXNzj0iaH80cDovL2xvY2Fsa69zdDo4MDgmL3]1Wxtcy9teX3]YWxtCy9teX3]YWxtLy1kIja1 WikjB3vudcIsInIba10mtCiF1Um12XLiIIC1LTATUKUMUSAUMUSIIIFENGI61613/JV3LiIsIaF8cCIE6131242/2022/043 OGYyZ0gBHTVMFSII#j=iIaI3jEiLG3yZ#sbV9My2NiSHI0nsizme2xXH01siZexVVVsdEJV2NiSiIIIV2X1sIIaF8cCIE6131242/2022/043 OGYyZ0gBHTVMFSII#j=iIaI3jEiLG3yZ#sbV9My2NiSH0nsizme2xXH01siZexVVVsdEJV2NiSiIIIV2X1sIIaF8cCIE613242/2022/043 OGYyZ0gBHTVMFSII#j=iIaI3jEiLG3yZ#sbV9My2NiSH0nsizme2x2H01siZexVVVsdEJV9X1y2Liv3IYHKT1wi5DZm6IQL2V9MY2NiSHIIIISIIIIIISII#ecCII20xJ1YU5V05000432/09h YXNiSHNonsiXJ19601LoftC2110nisime3xXH01siSH0NiSH001siZ01cley3y2Liv3IYHKT1wi5DZm6IQL2V9MY2NiSHNMFSII#FYX08609xXphdiivHIISFmerCII20xyII19(C3yZNvdX3)ZV9h YXNiSHNonsiXJ19601LoftC2110nisime3xXH01siSH0NiSH001siZ01cley3y2Liv3IYHKF1wi5DZm6IQL2V9MY2NiSHNMFSII#FYX08601Wyb0EtSiInH01yHxm9makH1119501siC402XH02U0404000 ANQcHJV2NisZ5B1WHF9bC1sInNp2CIG1jkxHjRMKZUMLuzJMertM0gyMCIIZ0cLTRDc6Y70Z04FWHK5II#FYX08601Wyb0EtSiInH01yHxm9makH1119501siC402XH01yH01J1010+YXZXIILC30xH02XJ2V9h YXNiSHNonsiNgHXMSIIIJ9501siAUIC2HWXFXxaHeHFPA5B12KYZWM62UFHXM5III99WHxZSXisHWKF5II#FXXHI0HXm01J131teXzXIILC30xH02XJ2V9h YZNISHNFXXISHWHHy1W1J02XH1LG30xH01QH2XH1C30xH2XH7XH32ZIAYTF3U5ZZIAYTFXX0FXF5H2V91SK1f433S15LVVSysjkWerM75Mdopl5q1HWJFVUMUFXQ782_V2XII LC3naXZID4VHXFXX9TyU2AmARCPMyHiva3gBXAN_vxpY5eQgs866-UVnVYR802CI0MgY7YECt1j0MR9YfmgJmmQuSmcc660hcSUncgYd3HMRAtW1p7EvMN3aLdbQW7VQzZtqNcrumeCsBRxAu2UrCgm*, 3 *vextmes_b0, 3 *vextmes_f04, 3 *vextm</pre>						

We'll use this access\_token to our request to endpoint "/user"

Keycloak-Merce Way / User			Save 🗸 👓	
GET v http://localhost:8090/test/user				Send v
Params Authorization Headers (7) Body Pre-request Script Tests Headers Ø Hilde auto-generated headers				
Key	Value	Description	••• Buik Edit	Presets $\vee$
Postman-Token 💿	<calculated is="" request="" sent="" when=""></calculated>			
✓ Host ③	<calculated is="" request="" sent="" when=""></calculated>			
User-Agent ③	PostmanRuntime/7.32.2			
Accept ()				
Accept-Encoding (i)	gzip, deflate, br			
Connection (1)	keep-alive			
Authorization	bearer eyJhbGciOiJSUzi1NiIsinR5cClgOiAiSidUliwia2lkliA6iCJJVEFIMkVV			
Key				
Body Cookles Headers (11) Test Results				
Pretty Raw Preview Visualize Text ~ 🚍				<b>E</b> Q
1 Hello form method User to User myuser				

We'll get Forbidden error for endpoint "/admin"

Keycloak-Merce Way / User		円 s	ave 🗸 👓	1
GET ~ http://localhost:8090/test/admin				Send 🗸
Params Authorization Headers (7) Body Pre-request Script Tests Headers Ø Hilde auto-generated headers				
Key	Value	Description	••• Bulk Edit	Presets ~
Postman-Token 🗓	<calculated is="" request="" sent="" when=""></calculated>			
✓ Host <sup>①</sup>	<calculated is="" request="" sent="" when=""></calculated>			
User-Agent ()	PostmanRuntime/7.32.2			
Accept ()				
Accept-Encoding ()	gzip, deflate, br			
Connection ()	keep-alive			
✓ Authorization	bearer eyJhbGclOiJSUzl1NilsInR5cClgOiAlSIdUliwia2lkliA6lCJJVEFIMkVV			
Кеу				
Body Cookies Headers (11) Test Results				
Pretty Raw Preview Visualize Text ~ 🚍				🖻 Q
1				

So, we have now secured our endpoints with correct roles along with authentication based on Oauth2 framework using Keycloak.