



Upgrade

Open in app



Ugonna Ubaka · Follow

Oct 3, 2021 · 8 min read · Listen



Securing your Spring-Boot Microservice with Keycloak and Spring Security

Photo by [Jose Fontano](#) on [Unsplash](#)

[Keycloak](#) is an open-source product developed by the RedHat Community which provides identity and access management solutions to modern applications. It requires little to no code to secure your applications and services.

Let's dive right in

Set Up Keycloak

Head over to the Keycloak's [getting-started-guide](#) , download and set up keycloak on your local.

Getting Started Guide

This section describes how to install and start a Keycloak server in standalone mode, set up the initial admin user...

www.keycloak.org

Create a Realm

A realm manages a set of users, credentials, roles, and groups. The users you create in a realm belong to that realm. So, when they log in to Keycloak, they log in to the specified realm





Upgrade

Open in app

1. Click on the upper left corner and choose **Add realm**.
2. Name it `demo` and click **Create**.

The screenshot shows the 'Add realm' form in the Keycloak Admin Console. The left sidebar has a 'Select realm' dropdown. The main form area has an 'Import' section with a 'Select file' button. Below that is a 'Name' field with the value 'demo'. There is an 'Enabled' toggle switch set to 'ON'. At the bottom are 'Create' and 'Cancel' buttons.

Create Realm

Create a client

A client in keycloak, is a resource server i.e the server that hosts the resources that needs to be secured. In our case, it'll be the Spring Boot app we're going to create shortly.

Let's add a client to our realm.

1. Click the **Clients** menu item.
2. Select **Create** from the upper right corner.
3. Call it `demo-service`.
4. Enter the root/base url of our microservice. Our Spring Boot app will be running on <http://localhost:8081>. So, let's enter <http://localhost:8081>
5. Save the client.
6. Click the **Settings** tab. change the **Access Type** from `public` to `confidential` and switch on **Authorization Enabled**.

The screenshot shows the 'Add Client' form in the Keycloak Admin Console. The left sidebar has a 'Demo' dropdown and a 'Clients' menu item. The main form area has an 'Import' section with a 'Select file' button. Below that is a 'Client ID' field with the value 'demo-service'. There is a 'Client Protocol' dropdown set to 'openid-connect'. Below that is a 'Root URL' field with the value 'http://localhost:8081'. At the bottom are 'Save' and 'Cancel' buttons.

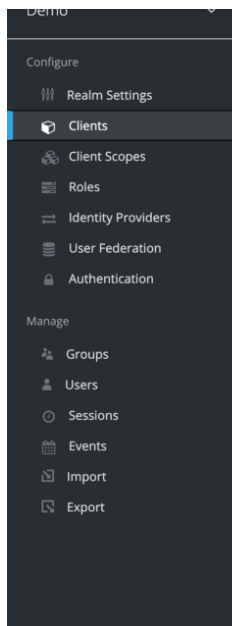
Add Client





Upgrade

Open in app



Demo-service

Settings

Credentials

Keys

Roles

Client Scopes

Mappers

Scope

Authorization

Revocation

Sessions

Offline Access

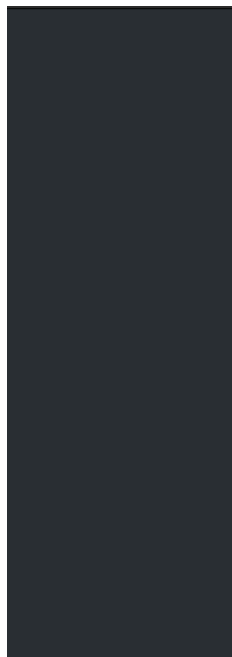
Clustering

Installation

Service Account Roles

Client ID	demo-service
Name	
Description	
Enabled	<input checked="" type="checkbox"/>
Always Display in Console	<input type="checkbox"/>
Consent Required	<input type="checkbox"/>
Login Theme	
Client Protocol	openid-connect
Access Type	confidential
Standard Flow Enabled	<input checked="" type="checkbox"/>
Implicit Flow Enabled	<input type="checkbox"/>
Direct Access Grants	<input checked="" type="checkbox"/>

Settings for Demo-service client



Direct Access Grants Enabled	<input checked="" type="checkbox"/>
Service Accounts Enabled	<input checked="" type="checkbox"/>
OAuth 2.0 Device Authorization Grant Enabled	<input type="checkbox"/>
OIDC CIBA Grant Enabled	<input type="checkbox"/>
Authorization Enabled	<input checked="" type="checkbox"/>
Root URL	http://localhost:8081
* Valid Redirect URIs	http://localhost:8081/*
Base URL	
Admin URL	http://localhost:8081
Web Origins	http://localhost:8081
Backchannel Logout URL	
Backchannel Logout Session Required	<input checked="" type="checkbox"/>
Backchannel Logout Revoke Offline Sessions	<input type="checkbox"/>

Settings for Demo-service client

Create a role

Currently, there are three types of roles in Keycloak:

- realm roles — they're global roles that are specific to the realm.
- client roles — they're specific to each client.
- composite roles — as the name implies, they're roles that have multiple roles associated with them.

In our example, let's create a client role dedicated to our Spring Boot app:

1. Select the `demo-service` client from the **Clients** menu item.
2. Click the **Roles** tab.

3. Click **Add Role**

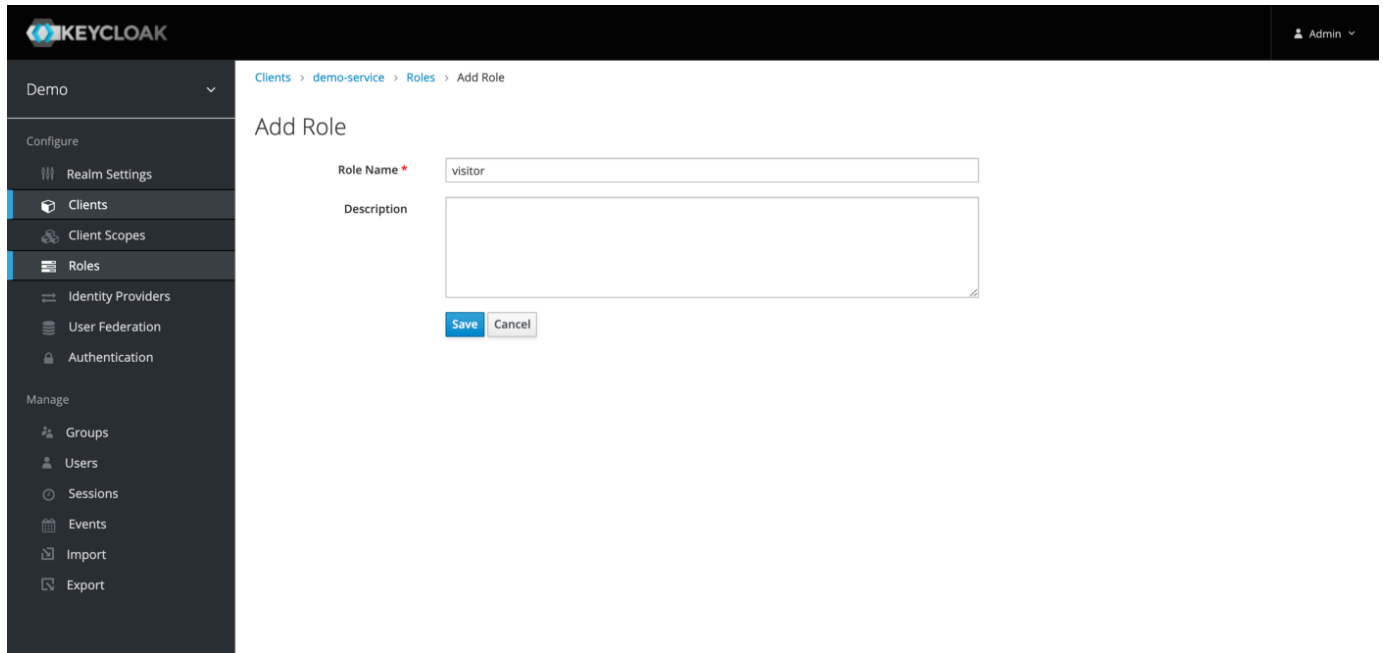




Upgrade

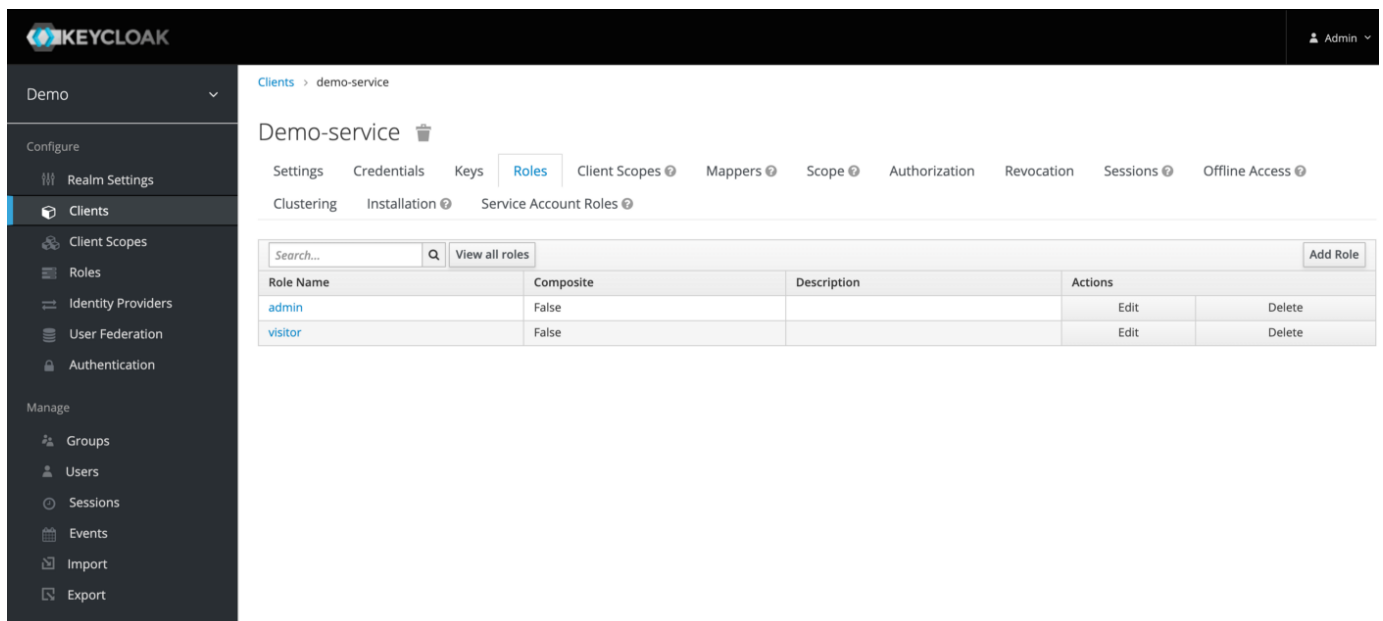
Open in app

5. Create another one called `admin` and save.



Add Role to Demo-service client

To verify that the roles have been successfully created, click the `demo-service` client, then the **Roles** tab and select **View all roles**:



All roles for Demo-service

Create users

Now we need to assign users to the roles we've just created.

1. Click the **Users** menu item.
2. Select **Add User**.
3. Create a user called `user` and save it.
4. Click on the **Credentials** tab for `user` and create a password, To avoid asking the user to change their password after the first login, switch the **Temporary** option off and set the password.





Upgrade

Open in app

Add user

ID

Created At

Username *

Email

First Name

Last Name

User Enabled ☒

Email Verified ☐

Groups
No group selected

Required User Actions

Add User

KEYCLOAK Admin Admin

Users > user

User

Details Attributes **Credentials** Role Mappings Groups Consents Sessions

Manage Credentials

Position	Type	User Label	Data	Actions
Set Password				
Password	<input type="text" value="user"/>			
Password Confirmation	<input type="text" value="user"/>			
Temporary	<input type="checkbox"/>	OFF		
<input type="button" value="Set Password"/>				

create password for user

5. Add another user called `admin` and create `admin` credentials following same steps in 3 & 4

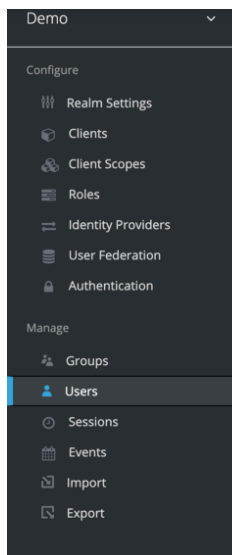
6. To demonstrate how permissions work, let's assign different roles to the users. For `user`, go to the `Role Mappings` tab. From the `Client Roles` drop-down menu, select `demo-service` and assign `visitor`.





Upgrade

Open in app



User

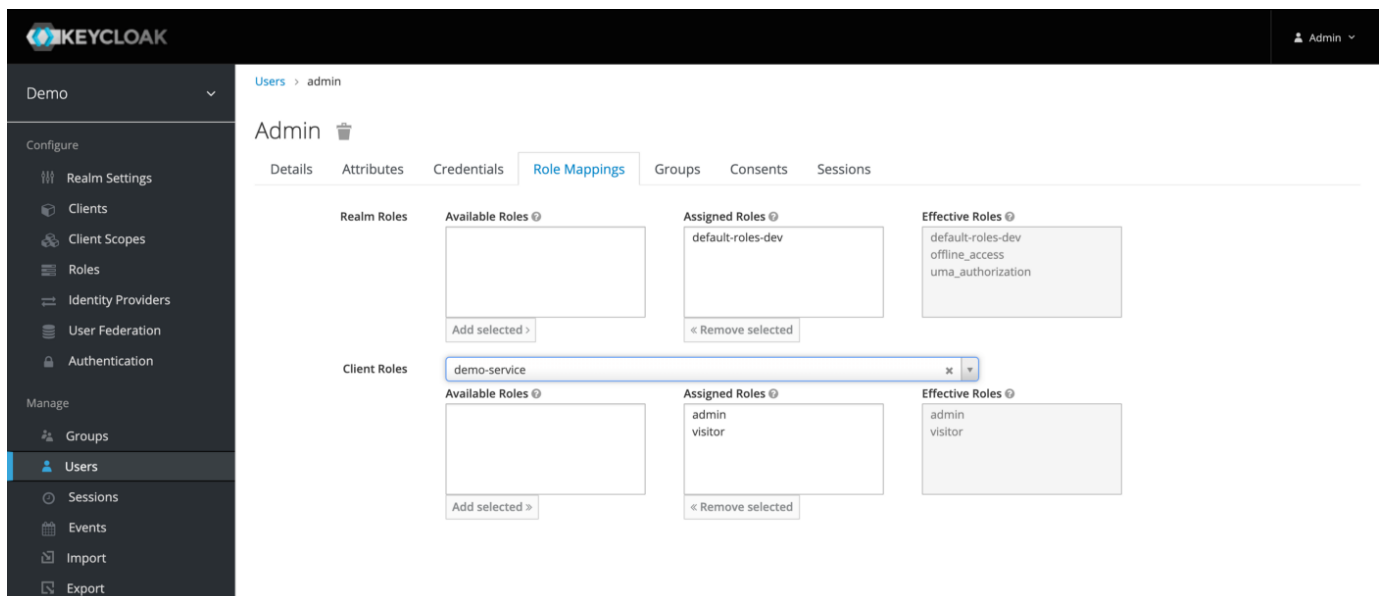
Details Attributes Credentials **Role Mappings** Groups Consents Sessions

Realm Roles	Available Roles	Assigned Roles	Effective Roles
		default-roles-dev	default-roles-dev offline_access uma_authorization
	Add selected >	<< Remove selected	

Client Roles	Available Roles	Assigned Roles	Effective Roles
demo-service	admin	visitor	visitor
	Add selected >	<< Remove selected	

Assign visitor role to user 'user'

For admin, assign both visitor and admin roles.



Assign visitor and admin role to user 'admin'

Create the Spring Boot App

We'll create a very simple app that will display custom text to the user based on their role. In our case, we'll distinguish visitors from admins

I'm using Maven to build this project. Create a New Maven Project in your favorite IDE. I'm Using IntelliJ. Alternatively, go to the [Spring Initializer](#) page to create a template Spring-Boot app.

We'll be using the **Spring-Boot keycloak adapter** dependencies for authorizations with our authorization server already running on <http://localhost:8080>. **lombok** is also used here, to avoid dummy getters and setters and for faster development with less code. Add the following dependencies to your `pom.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```





Upgrade

Open in app

```

 8      <version>2.5.4</version>
 9      <relativePath/>
10  </parent>
11  <groupId>org.pcs</groupId>
12  <artifactId>springboot-keycloak</artifactId>
13  <version>1.0</version>
14  <name>springboot-keycloak</name>
15  <description>Spring Boot for basic integration with Keycloak</description>
16
17  <properties>
18      <java.version>11</java.version>
19  </properties>
20
21  <dependencies>
22      <!--          lombok-->
23      <dependency>
24          <groupId>org.projectlombok</groupId>
25          <artifactId>lombok</artifactId>
26          <optional>true</optional>
27      </dependency>
28      <!--          keycloak adapter-->
29      <dependency>
30          <groupId>org.keycloak</groupId>
31          <artifactId>keycloak-spring-boot-starter</artifactId>
32          <version>15.0.2</version>
33      </dependency>
34      <!--          Spring Security-->
35      <dependency>
36          <groupId>org.springframework.boot</groupId>
37          <artifactId>spring-boot-starter-security</artifactId>
38      </dependency>
39  </dependencies>
40
41  <dependencyManagement>
42      <!--          keycloak adapter-->
43      <dependencies>
44          <dependency>
45              <groupId>org.keycloak.bom</groupId>
46              <artifactId>keycloak-adapter-bom</artifactId>
47              <version>15.0.2</version>
48              <type>pom</type>
49              <scope>import</scope>
50          </dependency>
51      </dependencies>
52  </dependencyManagement>
53
54  <build>
55      <plugins>
56          <plugin>
57              <groupId>org.springframework.boot</groupId>
58              <artifactId>spring-boot-maven-plugin</artifactId>
59          </plugin>
60      </plugins>
61  </build>
62
63 </project>

```

pom.xml hosted with ❤ by GitHub

[view raw](#)



Upgrade

Open in app

Create the main application

Create a new class called `Application.java` and add a **RestTemplate** Bean method, which we will use in making http calls. Create as follows:

```
1  package org.ugonna.springboot.keycloak;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.boot.web.client.RestTemplateBuilder;
6  import org.springframework.context.annotation.Bean;
7  import org.springframework.web.client.RestTemplate;
8
9  @SpringBootApplication
10 public class Application {
11
12     public static void main(String[] args) {
13         SpringApplication.run(Application.class, args);
14     }
15
16     @Bean
17     public RestTemplate restTemplate(RestTemplateBuilder builder) {
18         return builder.build();
19     }
20
21 }
```

Application.java hosted with ❤ by GitHub

[view raw](#)

Set-up Keycloak properties

To integrate Keycloak authentication, we need to define a few settings.

Create an `application.properties` file under the `resources` folder and define the following properties. The application is explicitly set





Upgrade

Open in app

```

3 keycloak.resource=demo-service
4 keycloak.auth-server-url=http://localhost:8080/auth
5 keycloak.ssl-required=external
6 keycloak.use-resource-role-mappings=true
7
8 #keycloak authentication properties
9 app.keycloak.login.url=http://localhost:8080/auth/realms/demo/protocol/openid-connect/token
10 app.keycloak.grant-type=password
11 app.keycloak.client-id = ${keycloak.resource}
12 app.keycloak.client-secret=53905103-24fd-49ca-b0b7-b08ff0ce3355
13
14 #Define authorization rules
15 #keycloak.security-constraints[0].authRoles[0]=visitor
16 #keycloak.security-constraints[0].securityCollections[0].patterns[0]=/visitor/*
17 #keycloak.security-constraints[0].authRoles[1]=admin
18 #keycloak.security-constraints[0].securityCollections[1].patterns[1]=/admin/*
19
20 server.port=8081

```

application.properties hosted with ❤ by GitHub

view raw

The keycloak authorization properties are specific properties Keycloak expects from our application in order to do authorizations for us. This properties can be found on keycloak's dashboard : Clients > demo-service > installation > keycloak OIDC JSON. The `keycloak.use-resource-role-mappings` property when set to `true` , tells keycloak to use the defined client roles for authorizations, otherwise, keycloak uses the realm roles instead. The authentication properties are required to authenticate users (in-app) within our realm and provide an `AccessToken` upon successful Authentication. We can also decide to set up route rules here, or in the `SecurityConfig.java` class, as we'll see later on. The part is commented out as we will be using the latter.

The screenshot shows the Keycloak Admin Console interface. On the left is a sidebar with navigation options like 'Demo', 'Configure', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', 'Authentication', 'Manage', 'Groups', 'Users', 'Sessions', 'Events', 'Import', and 'Export'. The main panel shows the configuration for the 'demo-service' client. The 'Installation' tab is active, displaying the 'Keycloak OIDC JSON' format. A 'Download' button is present, and the JSON content is shown in a text area.

Keycloak-OIDC-JSON

Create the Loan Service





Upgrade

Open in app

```
1  package org.ugonna.springboot.keycloak.service;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.beans.factory.annotation.Value;
5  import org.springframework.http.*;
6  import org.springframework.stereotype.Service;
7  import org.springframework.util.LinkedMultiValueMap;
8  import org.springframework.util.MultiValueMap;
9  import org.springframework.web.client.RestTemplate;
10 import org.ugonna.springboot.keycloak.dto.LoginRequest;
11 import org.ugonna.springboot.keycloak.dto.LoginResponse;
12
13
14 @Service
15 public class LoginService {
16
17     @Value("${app.keycloak.login.url}")
18     private String loginUrl;
19     @Value("${app.keycloak.client-secret}")
20     private String clientSecret;
21     @Value("${app.keycloak.grant-type}")
22     private String grantType;
23     @Value("${app.keycloak.client-id}")
24     private String clientId;
25
26     @Autowired
27     RestTemplate restTemplate;
28
29     public ResponseEntity<LoginResponse> login (LoginRequest request) {
30
31         HttpHeaders headers = new HttpHeaders();
32         headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
33
34         MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
35         map.add("username", request.getUsername());
36         map.add("password", request.getPassword());
37         map.add("client_id", clientId);
38         map.add("client_secret", clientSecret);
39         map.add("grant_type", grantType);
40
41         HttpEntity<MultiValueMap<String, String>> httpEntity = new HttpEntity<>(map, headers);
42         ResponseEntity<LoginResponse> loginResponse = restTemplate.postForEntity(loginUrl, httpEntity, LoginResponse.class);
43
44         return ResponseEntity.status(200).body(loginResponse.getBody());
45
46     }
47 }
```

LoginService.java hosted with ❤ by GitHub

[view raw](#)



Upgrade

Open in app

Create the Login Request and Response DTOs

Create a new class called `LoginRequest.java` and `LoginResponse.java` as follows :

```
1 package org.ugonna.springboot.keycloak.dto;
2
3 import lombok.Data;
4
5 @Data
6 public class LoginRequest {
7     private String username;
8     private String password;
9 }
```

LoginRequest.java hosted with ❤ by GitHub

[view raw](#)

```
1 package org.ugonna.springboot.keycloak.dto;
2
3 import lombok.Data;
4
5 @Data
6 public class LoginResponse {
7     private String access_token;
8     private String refresh_token;
9     private String expires_in;
10    private String refresh_expires_in;
11    private String token_type;
12 }
```

LoginResponse.java hosted with ❤ by GitHub

[view raw](#)

Create the Controllers

Create a new class called `LoginController.java` for our login operation and `HelloController.java` for other operations within our application.

```
1 package org.ugonna.springboot.keycloak.controller;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PostMapping;
9 import org.springframework.web.bind.annotation.RequestBody;
10 import org.springframework.web.bind.annotation.RestController;
11 import org.springframework.web.servlet.HandlerMapping;
```





Upgrade

Open in app

```
17
18 @RestController
19 public class LoginController {
20
21     Logger log = LoggerFactory.getLogger(LoginController.class);
22
23     @Autowired
24     LoginService loginService;
25
26     @PostMapping("login")
27     public ResponseEntity<LoginResponse> login (HttpServletRequest request,
28                                                @RequestBody LoginRequest loginRequest) throws Exception {
29         log.info("Executing login");
30
31         ResponseEntity<LoginResponse> response = null;
32         response = loginService.login(loginRequest);
33
34         return response;
35     }
36 }
```

LoginController.java hosted with ❤ by GitHub

[view raw](#)

```
1 package org.ugonna.springboot.keycloak.controller;
2 import org.springframework.web.bind.annotation.*;
3
4 @RestController
5 public class HelloController {
6
7     @GetMapping("/visitor")
8     public String getVisitor(@RequestHeader String Authorization) {
9         return "Hello visitor";
10    }
11
12    @GetMapping("/admin")
13    public String getAdmin(@RequestHeader String Authorization) {
14        return "Hello admin";
15    }
16
17    @GetMapping("/user")
18    public String getUsers(@RequestHeader String Authorization) {
19        return "Hello user";
20    }
21
22    @GetMapping("/random")
23    public String getRandomUser() {
```





Upgrade

Open in app

We have 5 routes : `/login` , which will be used for authentication, `/visitor` , which will only be accessed by authenticated users with `visitor` role/privilege , `/admin` , which will only be accessed by authenticated users with `admin` role, `/user` , which will be accessed by any authenticated user (regardless of role) and `/random` , which will be accessed by anyone without authentication

Configure Keycloak and Route rules

Create a new class `SecurityConfig.java` , to set-up keycloak's spring security configurations and enable role-based authorizations to routes and other route rules.

```

1  package org.ugonna.springboot.keycloak.config;
2
3  import org.keycloak.adapters.springboot.KeycloakSpringBootConfigResolver;
4  import org.keycloak.adapters.springsecurity.KeycloakConfiguration;
5  import org.keycloak.adapters.springsecurity.authentication.KeycloakAuthenticationProvider;
6  import org.keycloak.adapters.springsecurity.config.KeycloakWebSecurityConfigurerAdapter;
7  import org.keycloak.adapters.springsecurity.filter.KeycloakAuthenticationProcessingFilter;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.context.annotation.Bean;
10 import org.springframework.http.HttpMethod;
11 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
12 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
13 import org.springframework.security.core.authority.mapping.SimpleAuthorityMapper;
14 import org.springframework.security.core.session.SessionRegistryImpl;
15 import org.springframework.security.web.authentication.session.RegisterSessionAuthenticationStrategy;
16 import org.springframework.security.web.authentication.session.SessionAuthenticationStrategy;
17 import org.ugonna.springboot.keycloak.exception.CustomKeycloakAuthenticationHandler;
18 import org.ugonna.springboot.keycloak.exception.RestAccessDeniedHandler;
19
20 // Defines all annotations that are needed to integrate Keycloak in Spring Security
21 @KeycloakConfiguration
22 class SecurityConfig extends KeycloakWebSecurityConfigurerAdapter {
23
24     @Autowired
25     RestAccessDeniedHandler restAccessDeniedHandler;
26
27     @Autowired
28     CustomKeycloakAuthenticationHandler customKeycloakAuthenticationHandler;
29
30     @Override
31     protected void configure(HttpSecurity http) throws Exception {
32         super.configure(http);
33         http.csrf().disable().cors().disable()
34             .authorizeRequests()
35                 .antMatchers("/login", "/random").permitAll()
36                 .antMatchers("/visitor").hasRole("visitor")
37                 .antMatchers("/admin").hasRole("admin")
38                 .anyRequest()
39                 .authenticated();
40
41         //Custom error handler

```





Upgrade

Open in app

```
47     public void configureGlobal( AuthenticationManagerBuilder auth) throws Exception {
48         KeycloakAuthenticationProvider keycloakAuthenticationProvider = keycloakAuthenticationProvider();
49         keycloakAuthenticationProvider.setGrantedAuthoritiesMapper(new SimpleAuthorityMapper());
50         auth.authenticationProvider(keycloakAuthenticationProvider);
51     }
52
53     // Use Spring Boot property files instead of default keycloak.json
54     @Bean
55     public KeycloakSpringBootConfigResolver keycloakConfigResolver() {
56         return new KeycloakSpringBootConfigResolver();
57     }
58
59     // Register authentication strategy for public or confidential applications
60     @Bean
61     @Override
62     protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
63         return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
64     }
65
66     //Keycloak auth exception handler
67     @Bean
68     @Override
69     protected KeycloakAuthenticationProcessingFilter keycloakAuthenticationProcessingFilter() throws Exception {
70         KeycloakAuthenticationProcessingFilter filter = new KeycloakAuthenticationProcessingFilter(this.authenticationManagerBean());
71         filter.setSessionAuthenticationStrategy(this.sessionAuthenticationStrategy());
72         filter.setAuthenticationFailureHandler(customKeycloakAuthenticationHandler);
73         return filter;
74     }
75
76
77 }
```

SecurityConfig.java hosted with ❤ by GitHub

[view raw](#)



Upgrade

Open in app

We set up the route rules in the `configure` method. Our `/login` and `/random` routes will be accessed by any unauthenticated user, `/visitor` will be accessed by authenticated users with `visitor` role, `/admin` will be accessed by authenticated users with `admin` role. Finally every other route(s) within our application will require the user to just be authenticated (without necessarily having any roles assigned to them). In this case, `/user` route

The `configureGlobal` method tells spring security to use keycloak as the auth provider

The `keycloakConfigResolver` tells keycloak to look into our `application.properties` file for keycloak configurations (prefixed with `keycloak`).

We can handle Authentication and Authorizaton exceptions, and return custom error messages to our users via the `RestAccessDeniedHandler` and `CustomKeycloakAuthenticationHandler` classes (link to complete code will be shared at the end).

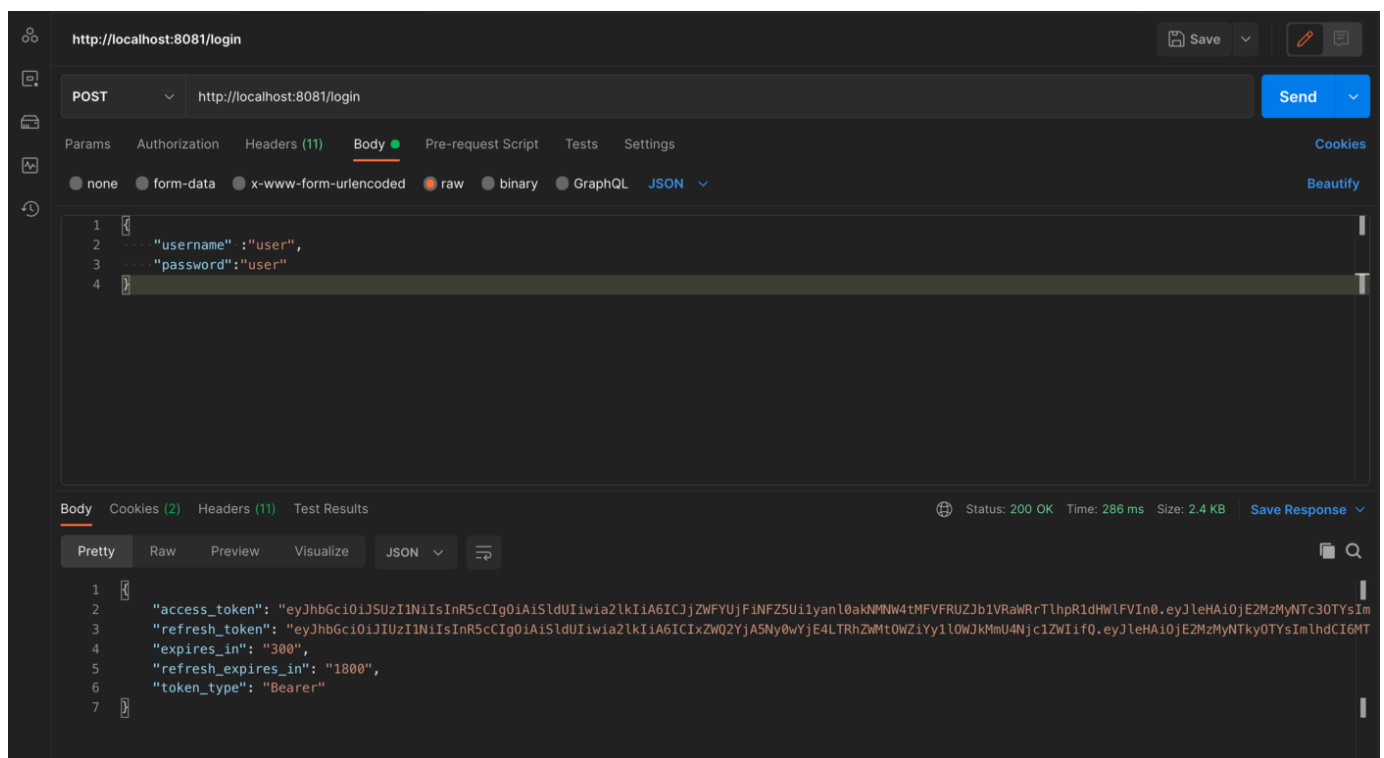
Test the Application

Start the app by executing this command in your Terminal:

```
mvn spring-boot:run
```

Or by clicking on the play button at the top right of intelliJ or from the main `Application.java` class

Open up postman, and make the call to `/login` with the credentials for **user**. retrieve the `access_token`, which will be used to try and access routes within our application.



`access_token` : is a jwt used as a Bearer token when accessing protected routes.

`refresh_token` : is also a jwt used to generate new token credentials without having the user login again (atleast till the refresh token expires).

`expires_in` : gives the expiry time of the `access_token` in seconds. This can be configured either at the realm level or client level or both, on the Keycloak dashboard. However, configuration at the client level takes precedence if set. For Realm level, From the



Upgrade

Open in app

Let's try accessing the `/visitor` route, setting the Authorization header to `Bearer ${access_token}`

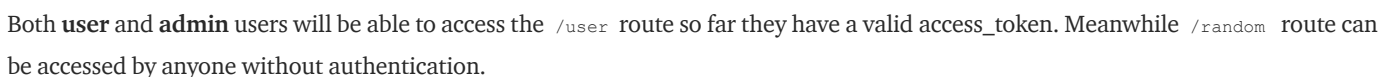
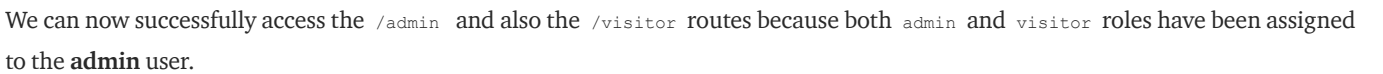
The screenshot shows a Postman interface for a GET request to `http://localhost:8081/visitor`. The 'Headers' tab is selected, showing various headers including 'Authorization' with a Bearer token. The 'Body' tab shows the response: 'Hello visitor'. The status is '200 OK'.

Accessing the `/admin` route with same `access_token` for `user` throws a 403 error because the user doesn't have `admin` privileges.

The screenshot shows a Postman interface for a GET request to `http://localhost:8081/admin`. The 'Body' tab is selected, showing the response in JSON format: `{ "timestamp": "2021-10-03T11:06:09.652+00:00", "status": 403, "error": "Forbidden", "path": "/admin" }`. The status is '403 Forbidden'.

Now let's login as `admin` and retrieve the admin `access_token`







Upgrade

Open in app

GET <http://localhost:8081/random> Send

Params Authorization **Headers (9)** Body Pre-request Script Tests Settings Cookies

Header	Value
Cache-Control	no-cache
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.28.2
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUiIiwia2kiOiAiA6ICJZ...

Body Cookies (2) Headers (11) Test Results Status: 200 OK Time: 20 ms Size: 363 B Save Response

Pretty Raw Preview Visualize Text 1 Hello random user

We can see that our app works as expected.

SideNote : to use `refresh_token` to generate new token credentials, here's how it's done. You can decide to integrate a second login endpoint in the `LoginController.java` class say `/loginRefresh` and add the corresponding method in the `LoginService.java` class. However, I am calling the token endpoint directly from our authorization server to demonstrate how it's done.

<http://localhost:8080/auth/realms/demo/protocol/openid-connect/token> Save Send

POST <http://localhost:8080/auth/realms/demo/protocol/openid-connect/token> Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
grant_type	refresh_token	
client_secret	53905103-24fd-49ca-b0b7-b08ff0ce3355	
client_id	demo-service	
refresh_token	eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUiIiwia2kiOiAiA6ICJZ...	

Body Cookies (5) Headers (11) Test Results Status: 200 OK Time: 88 ms Size: 2.52 KB Save Response

Pretty Raw Preview Visualize JSON 1

```

1 {
2   "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUiIiwia2kiOiAiA6ICJZWFYUjF1NFZSU11yanl0akNWNW4tMFVFRUJb1VRaWRrTlhpR1dHwLFVIn0.eyJleHAiOjE2MzMyNjI0NjUsIm
3   "expires_in": 300,
4   "refresh_expires_in": 1800,
5   "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUiIiwia2kiOiAiA6ICJZWFYUjF1NFZSU11yanl0akNWNW4tMFVFRUJb1VRaWRrTlhpR1dHwLFVIn0.eyJleHAiOjE2MzMyNjI0NjUsIm
6   "token_type": "Bearer",
7   "not-before-policy": 0,
8   "session_state": "23b7d7f6-b4d7-4e61-b569-585c6b191456",
9   "scope": "profile email"
10 }
```

login/authenticate via refresh_token

Conclusion

Congratulations, for making it to the end of this tutorial :). Now you should have a basic understanding of keycloak concepts and Spring Security and how to use it in securing your spring-boot microservice.



[Upgrade](#)[Open in app](#)

References

- <https://betterprogramming.pub/how-to-authenticate-your-spring-boot-application-with-keycloak-1e9ccb5f2478>
- https://www.keycloak.org/docs/latest/getting_started/

