DIGITALHOUSE

Keycloak Admin REST Client

Certified Tech Developer
The Ultimate Degree

Índice

01 Configuración

Configuración

Para utilizar este cliente tenemos que agregar la dependencia en el **pom.xml**.

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>21.1.0</version>
</dependency>
```

En **application.yml** configuramos la conexión con el cliente de Keycloak.

```
keycloak:
 realm: <nombre-reino>
 serverUrl: http://localhost:8080
 clientId:<KEYCLOAK-CLIENT-ID>
 clientSecret: <KEYCLOAK-CLIENT-SECRET>
```

Creamos una clase de configuración llamada **KeycloakClientConfiguration** con el fin de leer los atributos del **application.yml** para luego configurar el cliente REST de Keycloak mediante el *builder* **KeycloakBuilder**.

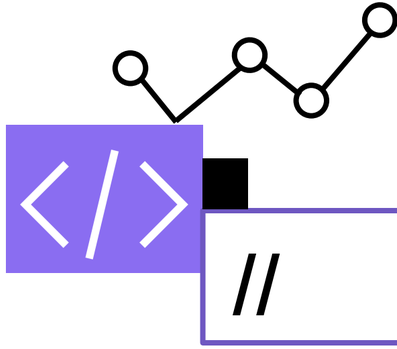
Además, agregamos la anotación **@Bean** para luego inyectar la clase **Keycloak** en donde necesitemos usarla.

```java
@Component
@ConfigurationProperties(prefix = "keycloak")
public class KeycloakClientConfiguration {
private String serverUrl;
private String realm;
private String clientId;
private String clientSecret;

@Bean
public Keycloak getInstance() {
 return KeycloakBuilder.builder()
    .serverUrl(serverUrl)
    .realm(realm)
    .clientId(clientId)
    .clientSecret(clientSecret)
    .grantType(OAuth2Constants.CLIENT_CREDENTIALS)
    .build();
}
}
```

¡Listo! Ya tenemos configurado el cliente REST de Keycloak.

02 Administrar usuarios



Administrar usuarios

Creamos una entidad “usuario” con los datos que queremos leer de la base de datos de Keycloak. Para el ejemplo utilizaremos la siguiente clase:

```
public class User {
private String userId;
private String firstName;
private String lastName;
private String email;
}
```

Creamos un repositorio que será el encargado de comunicarse con Keycloak y creamos, por ejemplo, los métodos para buscar usuarios por nombre de usuario e ID.

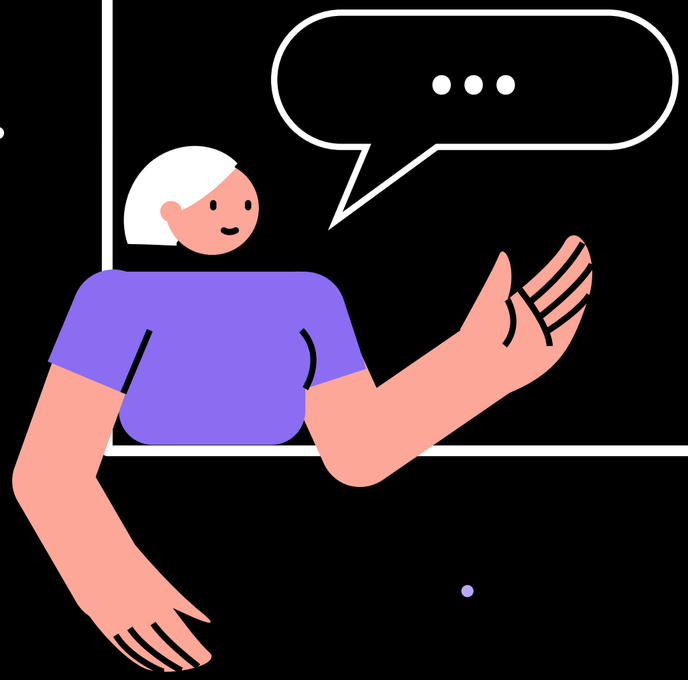
```
@Repository
public class KeycloakRepository {
 @Autowired
 private Keycloak keycloak;
@Value("\${keycloak.realm}")
 private String realm;

public List<User> findByUsername(String username) {
 List<UserRepresentation> userRepresentation =  keycloak
            .realm(realm)
            .users()
            .search(username);
 return userRepresentation.stream().map(user->
fromRepresentation(user)).collect(Collectors.toList());
}

public Optional<User> findById(String id) {
 UserRepresentation userRepresentation = keycloak
            .realm(realm)
            .users()
            .get(id)
            .toRepresentation()
 return Optional.of(fromRepresentation(userRepresentation));
}


private User fromRepresentation(UserRepresentation userRepresentation) {
return
User(userRepresentation.getId(),userRepresentation.getFirstName(),userRepresentation.getLast
Name(),userRepresentation.getEmail())}
```

Podemos ver que las consultas de Keycloak nos retornan un objeto de la clase **UserRepresentation**. Por lo tanto, nosotros creamos un objeto de tipo **User** para utilizar solamente la información que nos interesa.



03

Agregar o modificar atributos del usuario

Agregar o modificar atributos del usuario

Supongamos que agregamos a nuestra clase **User** un atributo para guardar la URL de la foto del usuario.

```
public User update(User user) {
 UserResource userResource = getUserResource(user.getUserId());
 UserRepresentation userRepresentation = userResource.toRepresentation();

Map<String, List<String>> attributes = new HashMap<>();
attributes.put("photoURL", List.of(user.getPhotoURL()));
userRepresentation.setAttributes(attributes);

userResource.update(userRepresentation);
 return fromRepresentation(userRepresentation);}
```

Buscamos el usuario por ID y, una vez lo tenemos, obtenemos el objeto **UserRepresentation**. En dicho objeto, seteamos la lista de atributos. En nuestro caso, será uno solo, llamado “photoURL”. Por último, realizamos la actualización en Keycloak.

DIGITALHOUSE

¡Muchas gracias!