Fabian Bouché, February 15th 2019

OpenID Connect with Keycloak

Introduction

Keycloak is an Identity and Access Management solution.

In this tutorial, we will use Keycloak to authenticate agents responsible for the human decision of the approval of a credit request.

Besides, Keycloak will be configured to deliver an Access Token to the application that those agents use to perform this task.

Eventually, the API developed in the Activiti tutorial will be enriched so that it checks the validity of the Access Token.

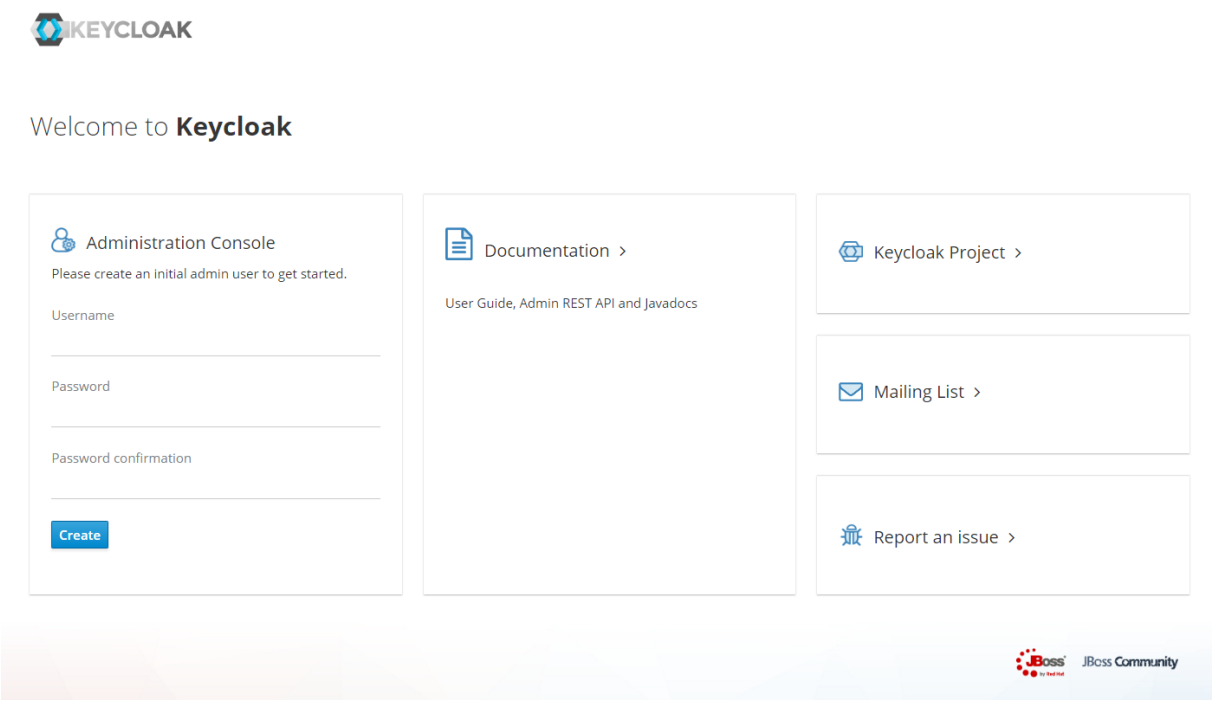
Setup of the environment

The following components have been setup for the POC:

Download Keycloak Server at https://www.keycloak.org/downloads.html

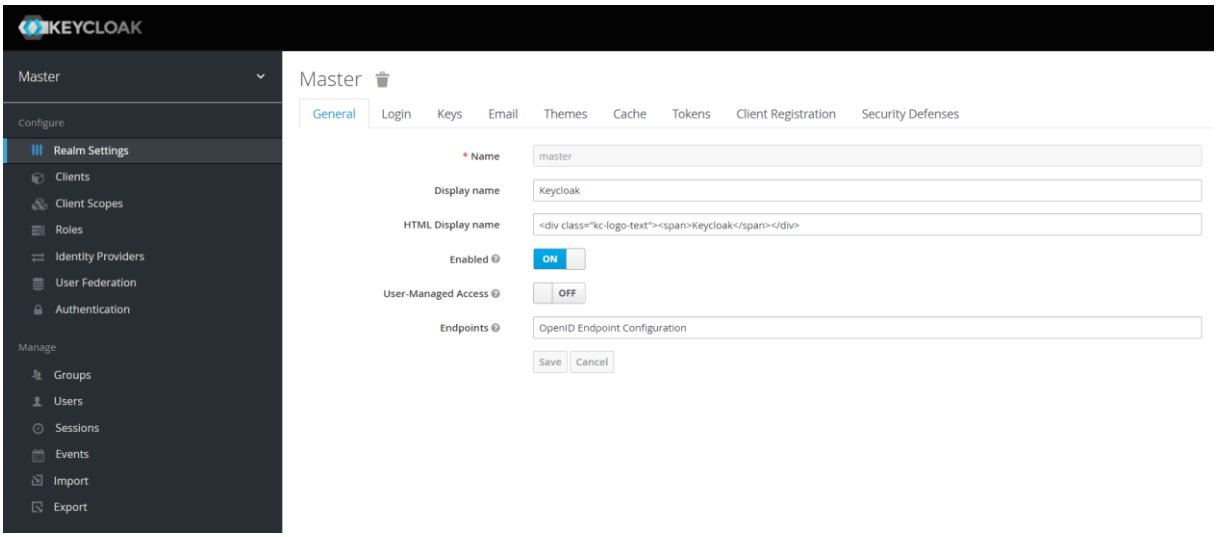
To start the server, execute the standalone script under the bin directory

Connect to http://localhost:8080 :



Create an admin user, choosing a username and a password.

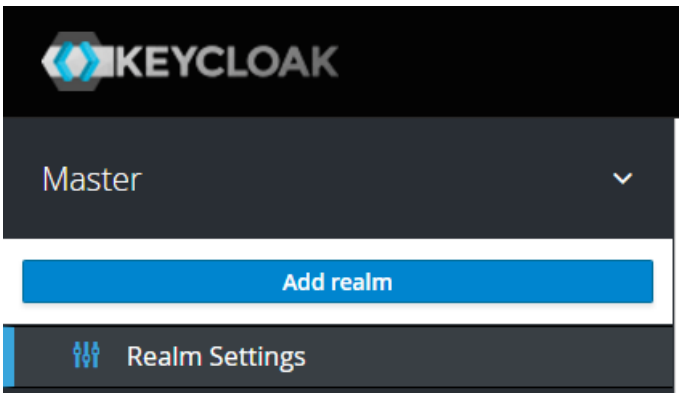
Use that user to connect, you now have access to the administration console:



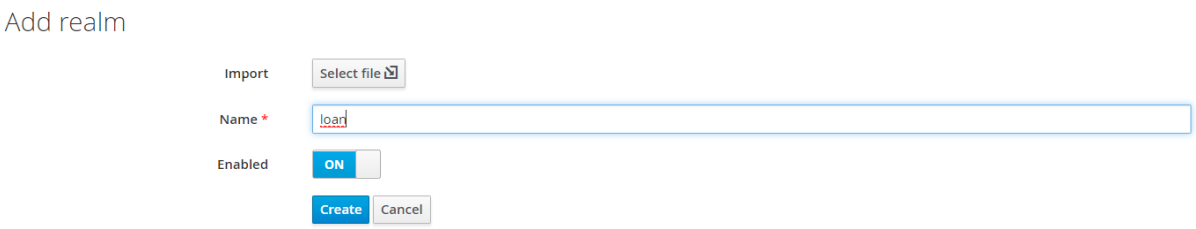
Setting up the Keycloak server

Keycloak is a multi-tenant solution. We will first create a “realm” dedicated to our tutorial:

Click the Add Realm button:

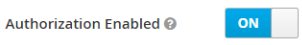


Create a realm called “loan”:

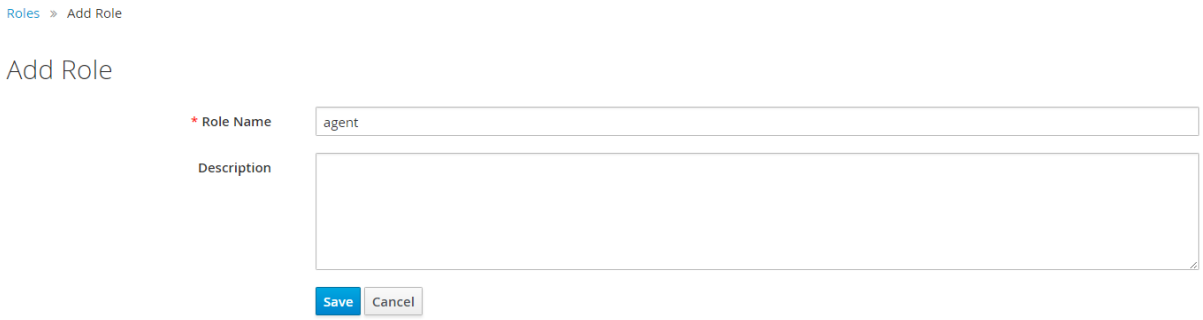


We will register one client application called “lonis”. This is the application internal agents use to interact with the loan subscription API to approve or deny credit requests.

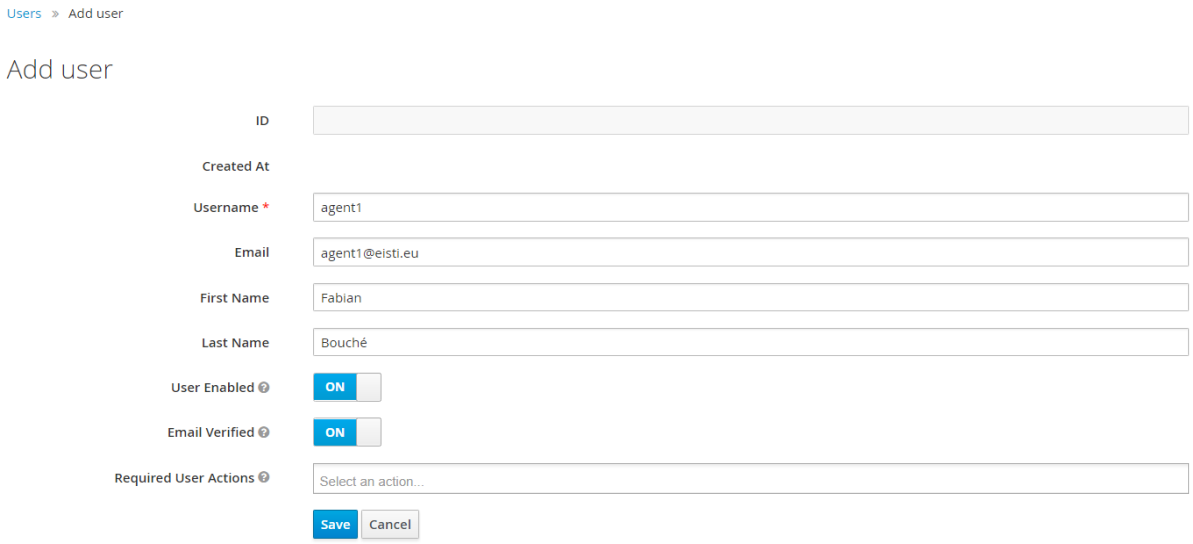
Once it has been created, enable Authorization and save:



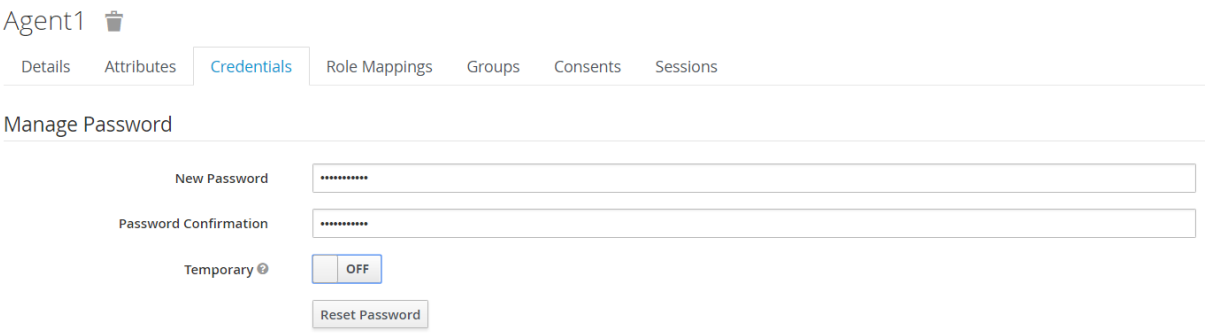
We will now create an “agent” role that all agents will have.



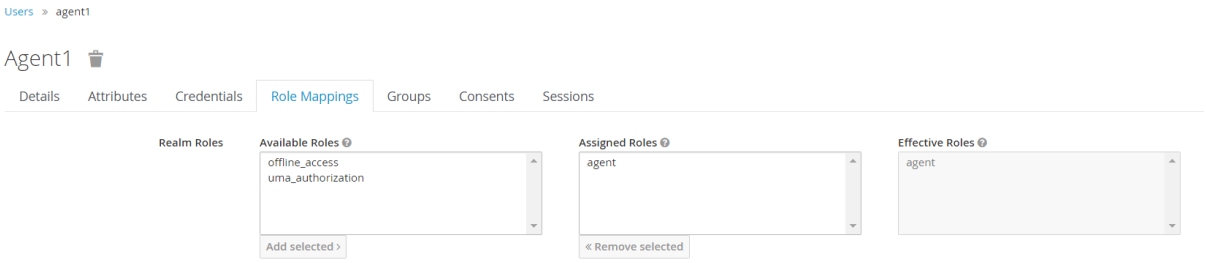
We will now register an agent. Go to Manage ➔ Users and click the “Add User” button.



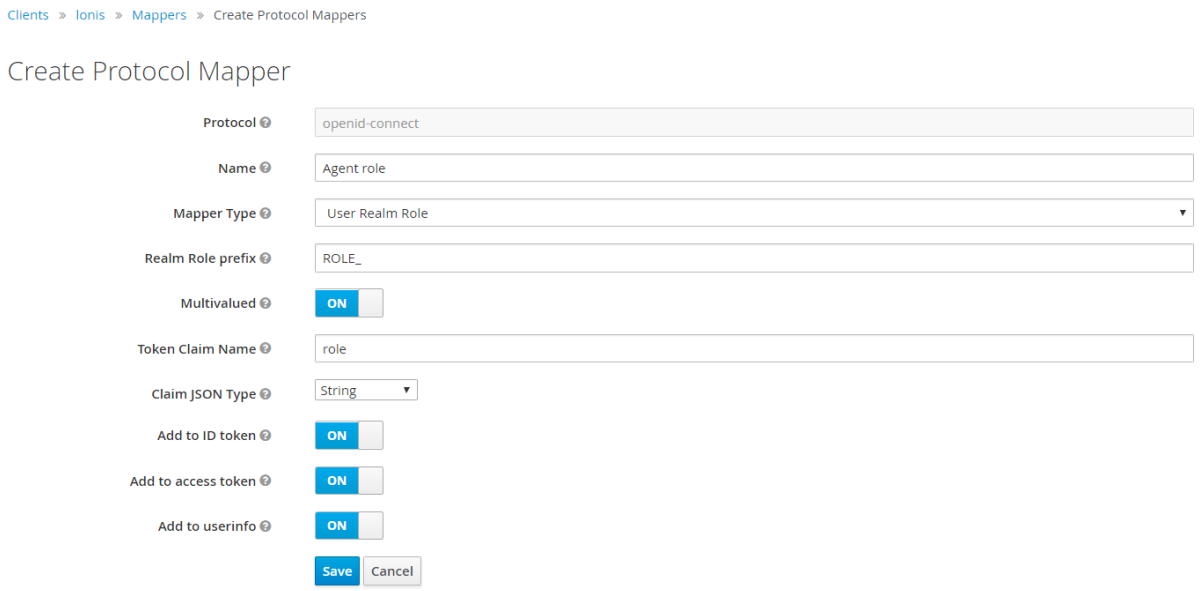
Credentials must be set for that user:



And finally, the agent role must be assigned to him:

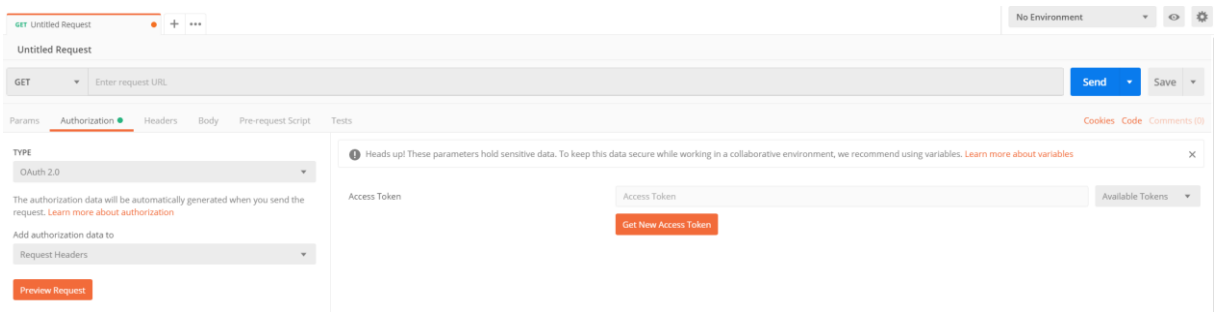


For the roles to appear inside of the Access token, you must define a Role Mapper for the Ionis client:

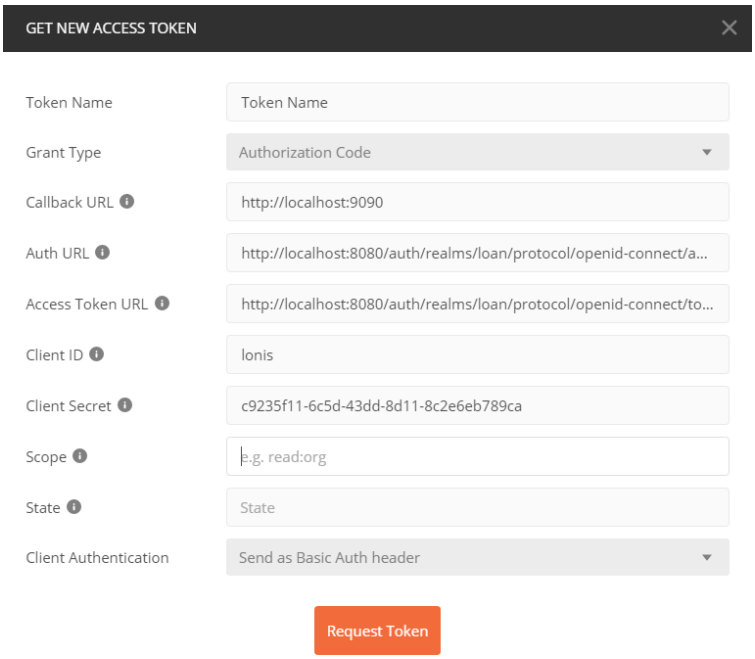


Getting an access token using Postman

Create a request. Under “Authorization”, choose the “OAuth 2.0” type and click “Get New Access Token”:



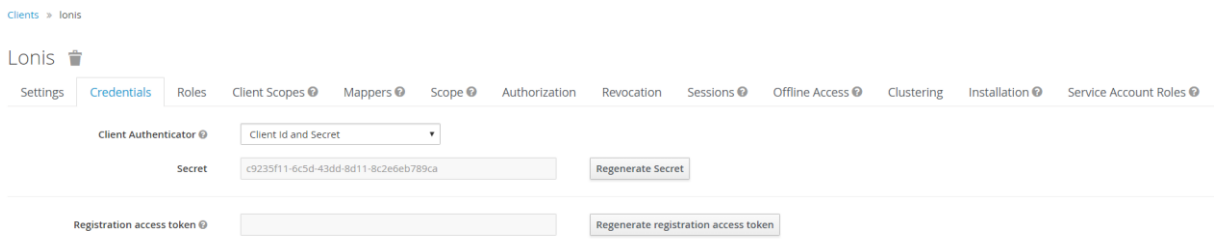
Fill in the following information:



The Auth URL is: http://localhost:8080/auth/realms/loan/protocol/openid-connect/auth

The Access Token URL is: http://localhost:8080/auth/realms/loan/protocol/openid-connect/token

The Client Secret is random and can be gathered from this screen:



Protecting the Springboot API

Add the following dependencies to the pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>com.nimbusds</groupId>
    <artifactId>nimbus-jose-jwt</artifactId>
    <version>7.0</version>
</dependency>
```

- The first one adds Spring security support that will enable us to implement access control on the application's controllers.
- The second one provides utilitary classes to validate and parse JWT Tokens.

Update the listening port in the application.properties file under src/main/resources:

```
server.port = 9090
```

Create a class that implements Web Security Configuration.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

        @Override
        protected void configure(HttpSecurity http) throws Exception {

                http.csrf().disable().authorizeRequests()

                        .antMatchers(
                                "/approve/**",
                                "/deny/**"
                                )
                                .hasRole("agent")

                        .antMatchers(
                                "/",
                                "/js/**",
                                "/error**",
                                "/create-lead",
                                "/submit-customer-information/**",
                                "/submit-credit-proposition/**",
                                "/status/**"
                                )
                                .permitAll()

                        .and()

                        .addFilter(new JWTAuthorizationFilter(authenticationManager()));
        }
}
```

You must make sure that only protected URLs require a given role to be assigned to the authenticated user.

Eventually, you have to write the JWT Authorization Filter.

```
public class JWTAuthorizationFilter extends BasicAuthenticationFilter {

    private Logger LOG = Logger.getLogger(JWTAuthorizationFilter.class);

    private static final String HEADER_STRING = "Authorization";
    private static final String TOKEN_PREFIX = "Bearer";
    private static final String JWKS_URL =
        "http://localhost:8080/auth/realms/loan/protocol/openid-connect/certs";

    public JWTAuthorizationFilter(AuthenticationManager authManager) {
        super(authManager);
    }

    @Override
    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain chain)
        throws IOException, ServletException {

        // Retrieve Authorization HTTP header
        String header = req.getHeader(HEADER_STRING);

        // Verify whether no access token has been provided
        if (header == null || !header.startsWith(TOKEN_PREFIX)) {
            chain.doFilter(req, res);
            return;
        }

        // Read user authentication information from token
        UsernamePasswordAuthenticationToken authentication = getAuthentication(req);
        SecurityContextHolder.getContext().setAuthentication(authentication);
        chain.doFilter(req, res);
    }

    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

        // Retrieve the Access Token from the HTTP header
        String token = request.getHeader(HEADER_STRING);
        String accessToken = token.replace(TOKEN_PREFIX, "");

        try {

            // Setting up a JWT Token processor
            ConfigurableJWTProcessor jwtProcessor = new DefaultJWTProcessor();

            // This Key source is provided by the Keycloak server
            JWKSource keySource = new RemoteJWKSet(new URL(JWKS_URL));

            // Algorithm used by Keycloak to sign tokens
            JWSAlgorithm expectedJWSAlg = JWSAlgorithm.RS256;

            // Configuration of the JWT Processor to validate tokens against the Keycloak server
            JWSKeySelector keySelector = new JWSVerificationKeySelector(expectedJWSAlg, keySource);
            jwtProcessor.setJWSKeySelector(keySelector);

            // Retrieve claims from the JWT Access Token
            // (but if validation fails, an exception will be raised)
            JWTClaimsSet jwtClaimsSet = jwtProcessor.process(accessToken, null);

            /*
             * Use this JWT parser instead of the one just above if you wish to bypass
             * JWT Access Token signature validation
             */
            //JWTClaimsSet jwtClaimsSet = JWTParser.parse(accessToken).getJWTClaimsSet();

            // Get the user's name from the claims
            String name = jwtClaimsSet.getStringClaim("name");

            // Get the user's roles from the claims (considering they are all prefixed with ROLE_)
            List<String> roleList = jwtClaimsSet.getStringListClaim("role");

            // Create the user's authentication context based on the claims gather in the Access Token
            if (name != null) {
                return createUserAuthenticationContext(name, roleList);
```

```
            }
        } catch (ParseException e) {
            LOG.error("Failed to parse JWT Token", e);
            return null;
        } catch (MalformedURLException e) {
            LOG.error("Malformed JWKS URL", e);
            return null;
        } catch (BadJOSEException e) {
            LOG.error("Failed to validate the JWT Access Token", e);
            return null;
        } catch (JOSEException e) {
            LOG.error("JOSE Exception while validating the JWT Access Token", e);
            return null;
        }
        return null;
    }

    private UsernamePasswordAuthenticationToken createUserAuthenticationContext(
        String name, List<String> roleList) {

        List<GrantedAuthority> list = new ArrayList<GrantedAuthority>();
        if(roleList == null) {
            LOG.info("Found authenticated user: " + name + " with NO ROLES");
            return new UsernamePasswordAuthenticationToken(name, null, null);
        } else {
            LOG.info("Found authenticated user: " + name);
            Iterator<String> roleIterator = roleList.iterator();
            while(roleIterator.hasNext()) {
                String role = roleIterator.next();
                LOG.info("User " + name + " has got role: " + role);
                list.add(new SimpleGrantedAuthority(role));
            }
            return new UsernamePasswordAuthenticationToken(name, null, list);
        }
    }

}
```

Try to make a request to both protected and unprotected URLs with or without an Access Token of an authenticated agent.

Managing several roles

Now, go back to the Activiti BPMN diagram and add a new process rule: loan applications with a duration exceeding 96 months require a validation from a manager.

- Configure this new “manager” Role in Keycloak
- Update the BPMN diagram
- Create two new methods on the API to allow the Manager either to accept or to deny the loan application
- Implement security so that only a manager may perform those actions