Imagine there are 2 micro-services built using [Spring-Boot](https://spring.io/projects/spring-boot) and secured using [Keycloak](https://www.keycloak.org/" \t "_blank) (OpenID + OAuth 2.0 compliant Authx Server).

1. [**spring-oauth2-employee-service**](https://github.com/bcarun/spring-oauth2-employee-service)**(service which calls another)**
2. [**spring-oauth2-department-service**](https://github.com/bcarun/spring-oauth2-department-service)**(service which is being called)**

In this article let us see how to configure spring-oauth2-employee-service with Service Account enabled and use [Spring OAuth2RestTemplate](https://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/client/OAuth2RestTemplate.html) to access spring-oauth2-department-service API.

*This is a lengthy article as it includes setting up Keycloak for 2 micro-services, coding 2 micro-services and testing oauth service account flow. It might look little complicated, believe me it is not. If you carefully follow each step, you don’t need to search for any other content to learn service account implementation.*

This article assumes you have basic understanding of securing REST API using Spring OAuth and keycloak, configuring clients (micro-services) in Keycloak. If not, please read my previous article stated below,

**[Securing REST API using Keycloak and Spring Oauth2](https://medium.com/@bcarunmail/securing-rest-api-using-keycloak-and-spring-oauth2-6ddf3a1efcc2" \t "_blank)**

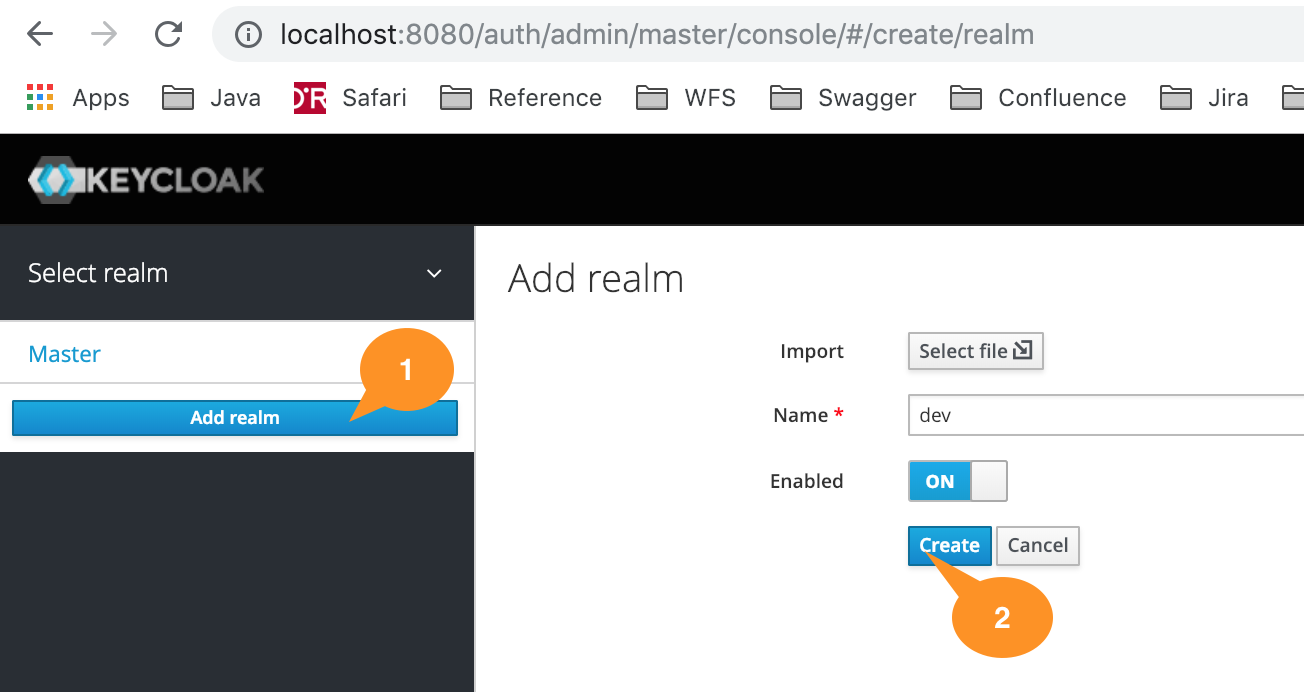
[Keycloak is Open Source Identity and Access Management Server, which is a OAuth2 and OpenID Connect(OIDC) protocol…](https://medium.com/@bcarunmail/securing-rest-api-using-keycloak-and-spring-oauth2-6ddf3a1efcc2" \t "_blank)

[medium.com](https://medium.com/@bcarunmail/securing-rest-api-using-keycloak-and-spring-oauth2-6ddf3a1efcc2" \t "_blank)

**1. Create ‘dev’ Realm In Keycloak**

Login to Keycloak and click on realm → Add realm.

Image for post



Create Realm In Keycloak

***Note:****To**access realm or open-id configuration use below link. Assuming Keycloak runs on Port 8080 and realm name = dev.*

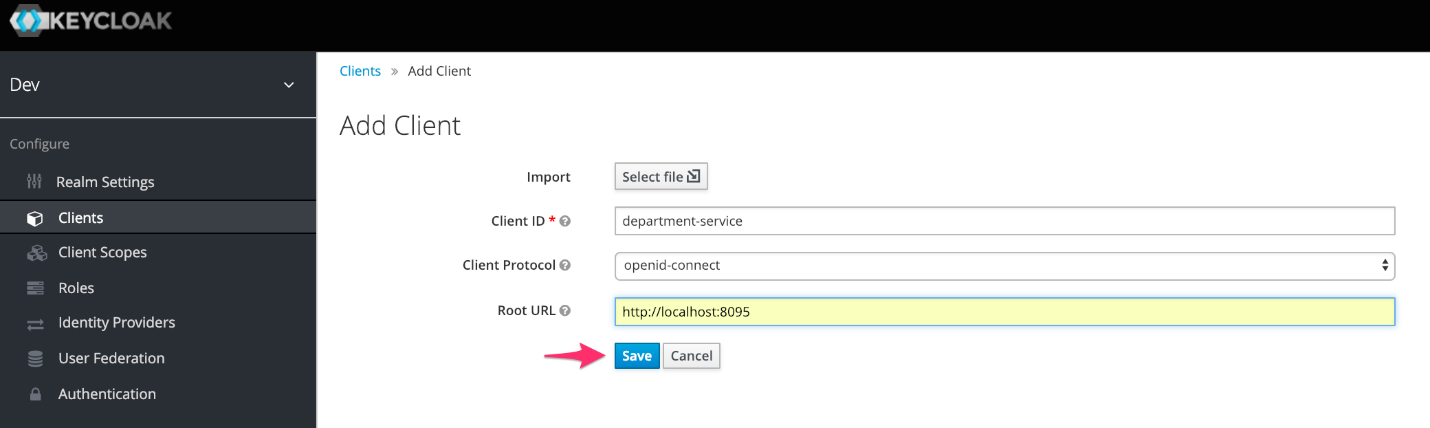
[*http://localhost:8080/auth/realms/dev/.well-known/openid-configuration*](http://localhost:8080/auth/realms/dev/.well-known/openid-configuration)

**2. Create Department Service Client (Callee)**

Click on clients → Create, to create a client named ‘**department-service**’ for securing **spring-oauth2-department-service**(micro-service).

Note that the root URL is [http://localhost:8095](http://localhost:8095/) as the corresponding spring-oauth2-department-service spring-boot application runs on port 8095.

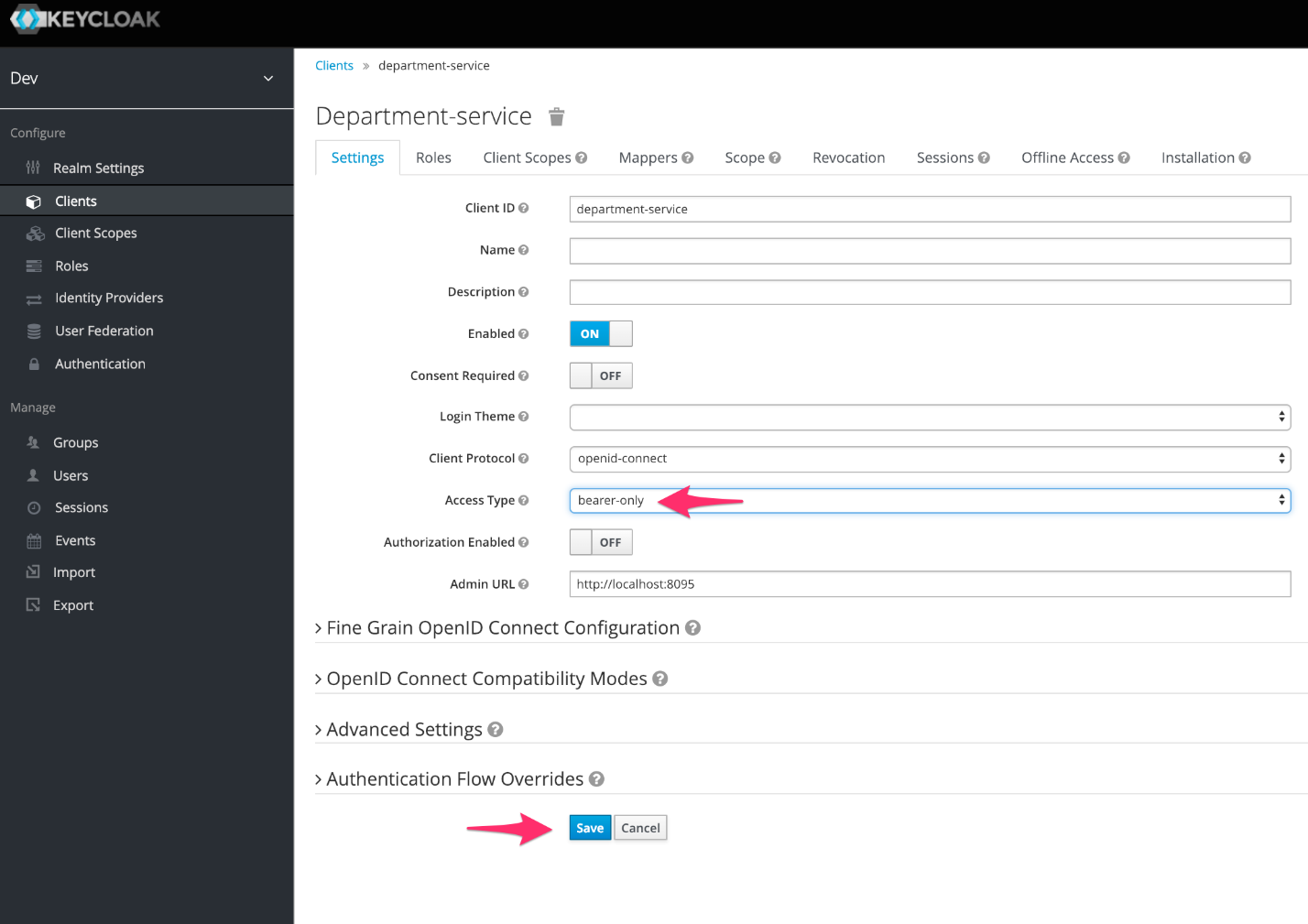
Image for post



Create department-service client in Keycloak

Configure department-service client as shown below and save. **Access Type = Bearer only,** means that this service will be accessed with just a bearer token. It also means that, this service will not call any other secure micro-service.

Image for post

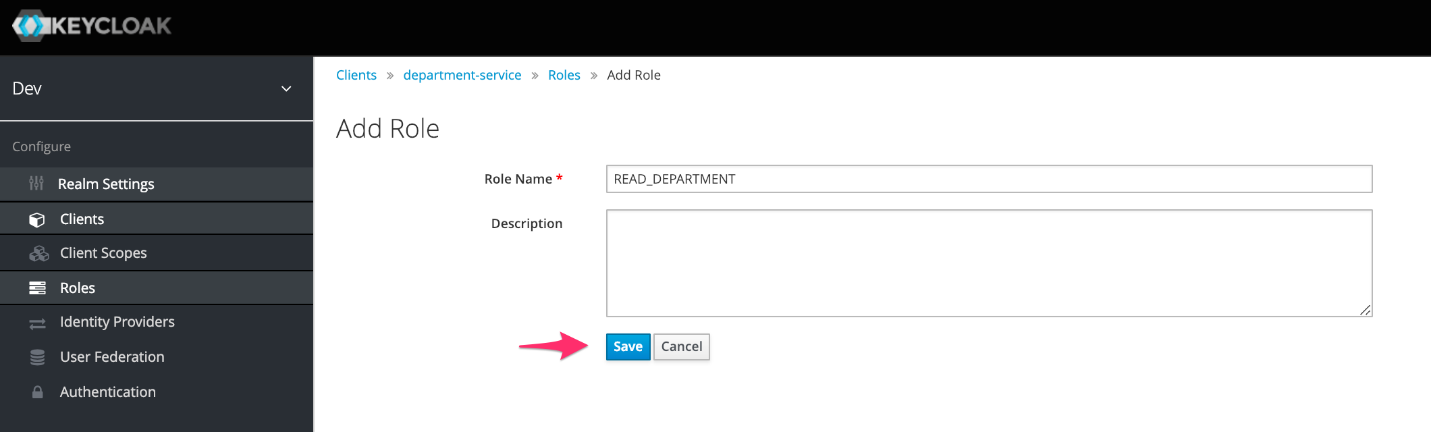


Configuring department-service with access type = bearer only

**3. Create a Role Under Department Service**

Open department-service client and click on Role to create a new **Role named READ\_DEPARTMENT**.

Image for post



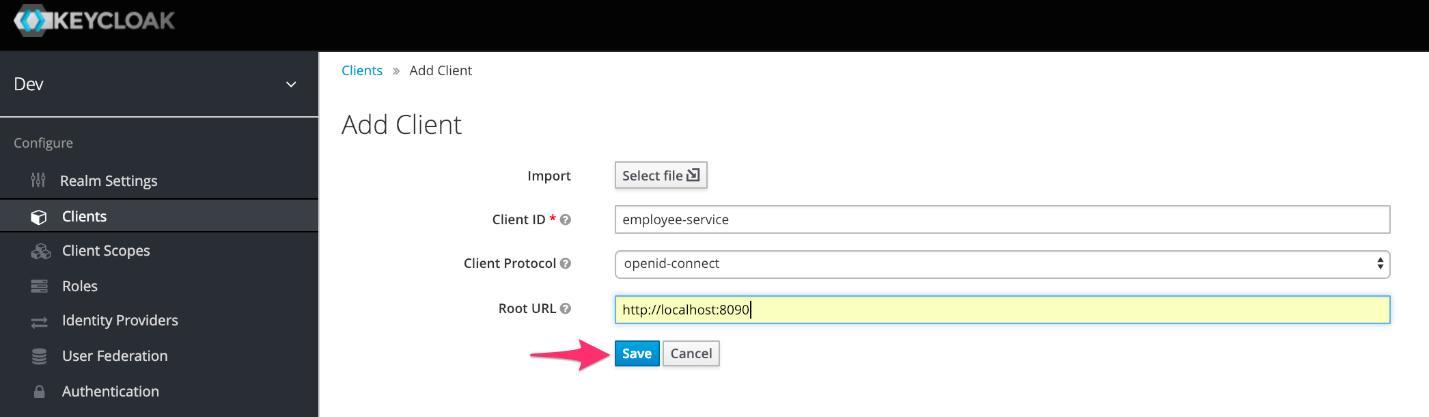
Create a role named READ\_DEPARTMENT Under department-service client

**4. Create Employee Service Client**

Click on clients → Create, to create a client named ‘**employee-service**’ for securing **spring-oauth2-employee-service**(micro-service).

Note that the root URL is [http://localhost:809](http://localhost:8095/)0 as the corresponding spring-oauth2-employee-service spring-boot application runs on port 8090.

Image for post

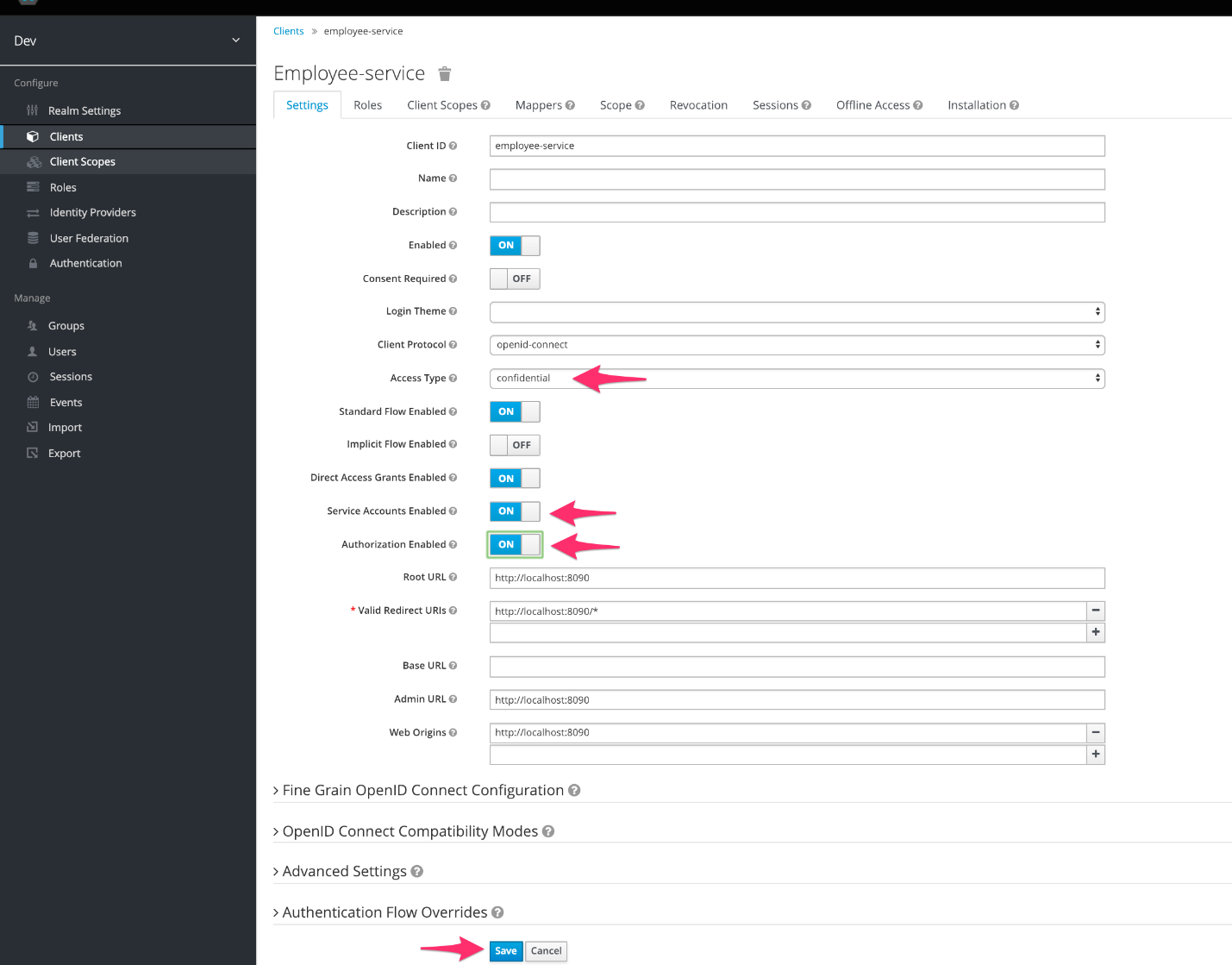


Create employee-service client in Keycloak

Configure **employee-service** client as shown below and save. **Access Type = Confidential,** means that this service will call or access another micro-service by getting an access token from Keycloak using its client credentials. Also, this micro-service may be called from another micro-service or UI.

***Note:****Set****Authorization Enabled = On****and then****Service Accounts Enabled = On.***

Image for post

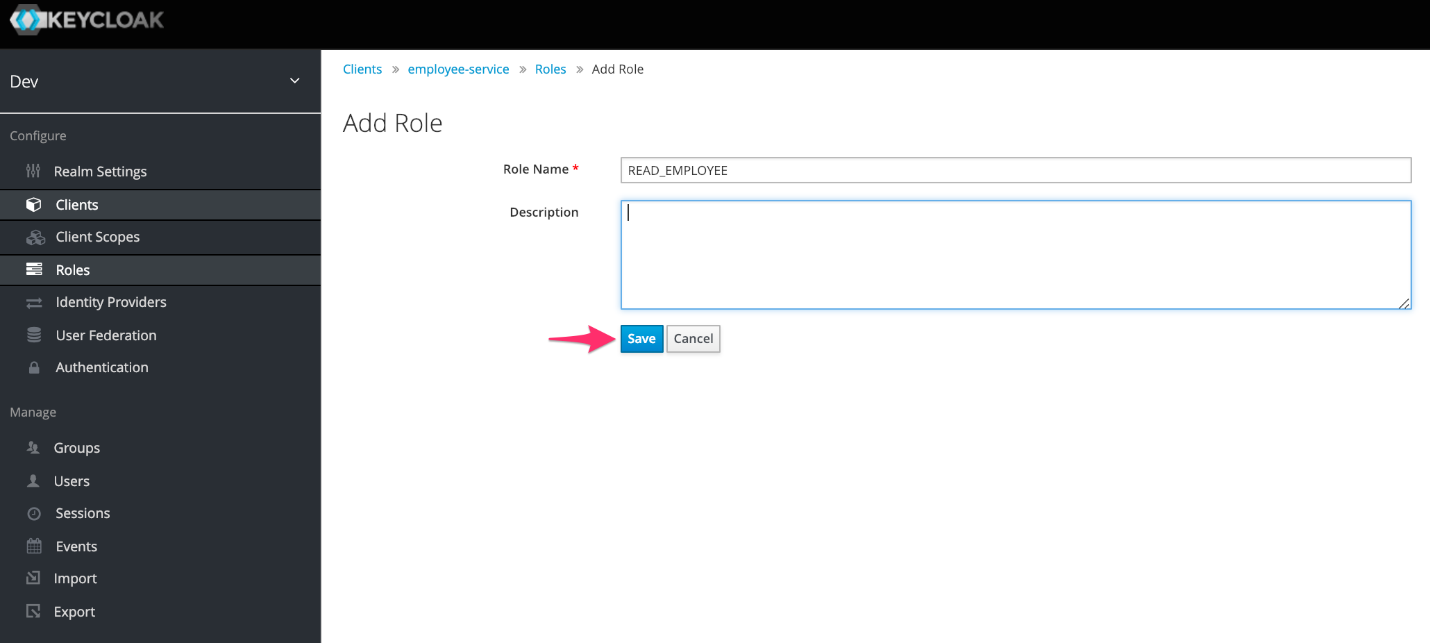


Configuring employee-service with access type = confidential, service account = on

**5. Create Client Role**

Open department-service client and click on Role to create a new **Role named READ\_EMPLOYEE**.

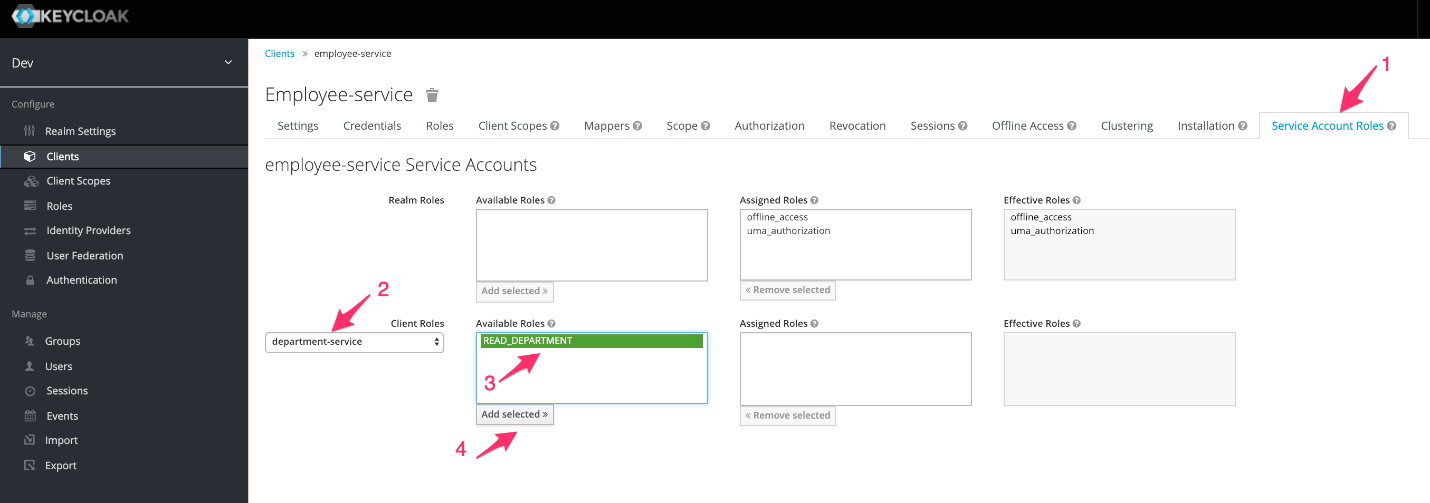
Image for post



Create a role named READ\_EMPLOYEE Under employee-service client

**6. Configure Service Account Role to Access Department Service**

Image for post



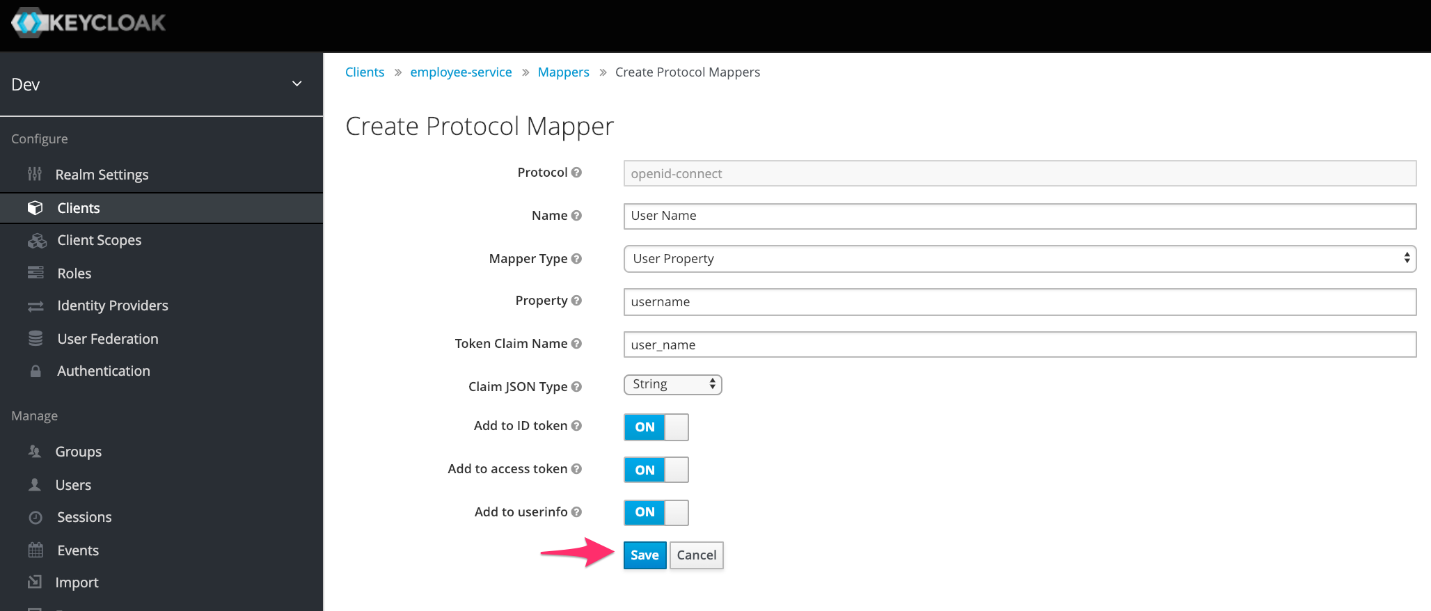
**Note:** User will have access to employee-service, but a micro-service will have access to department-service. Micro-service acting like user is called service account.

**7. Setup Protocol Mapper**

Spring expects JWT or access token to have a claim named user\_name, hence following protocol mapper is added.

This configuration gets the username from user property and sets it to a claim named user\_name.

Image for post

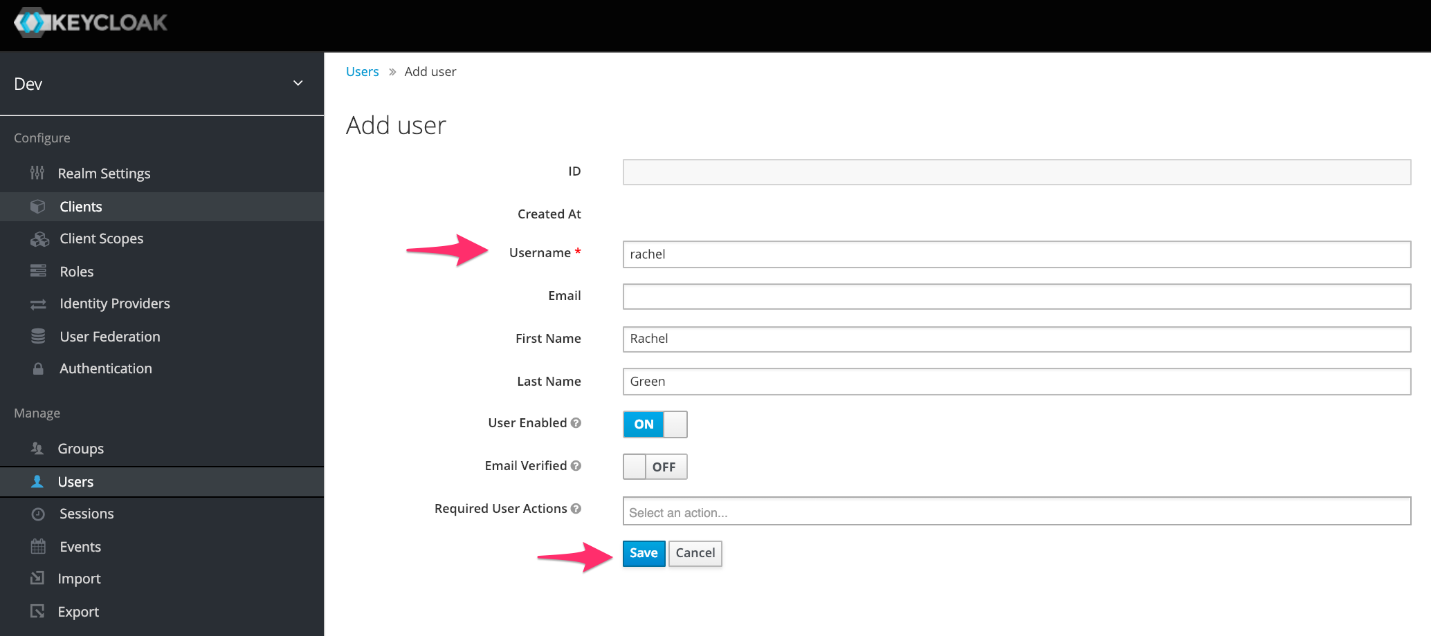


Create user\_name claim

**8. Create a User**

In order to simulate user calling spring-oauth2-employee-service, create a user.

Image for post



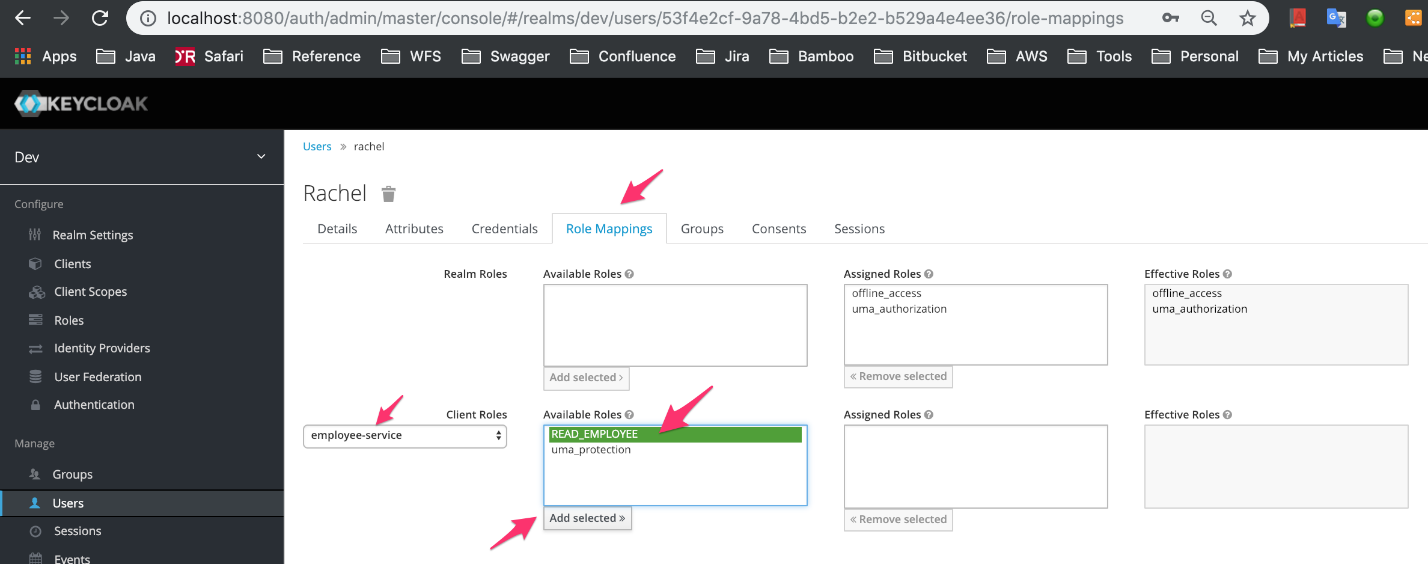
Create User in Keycloak

***Note 1:****Employee Service will use client credentials to call spring-oauth2-department-service.*

***Note 2:****Remember to setup password for user in credentials tab.*

**Assign role to user,**

Image for post

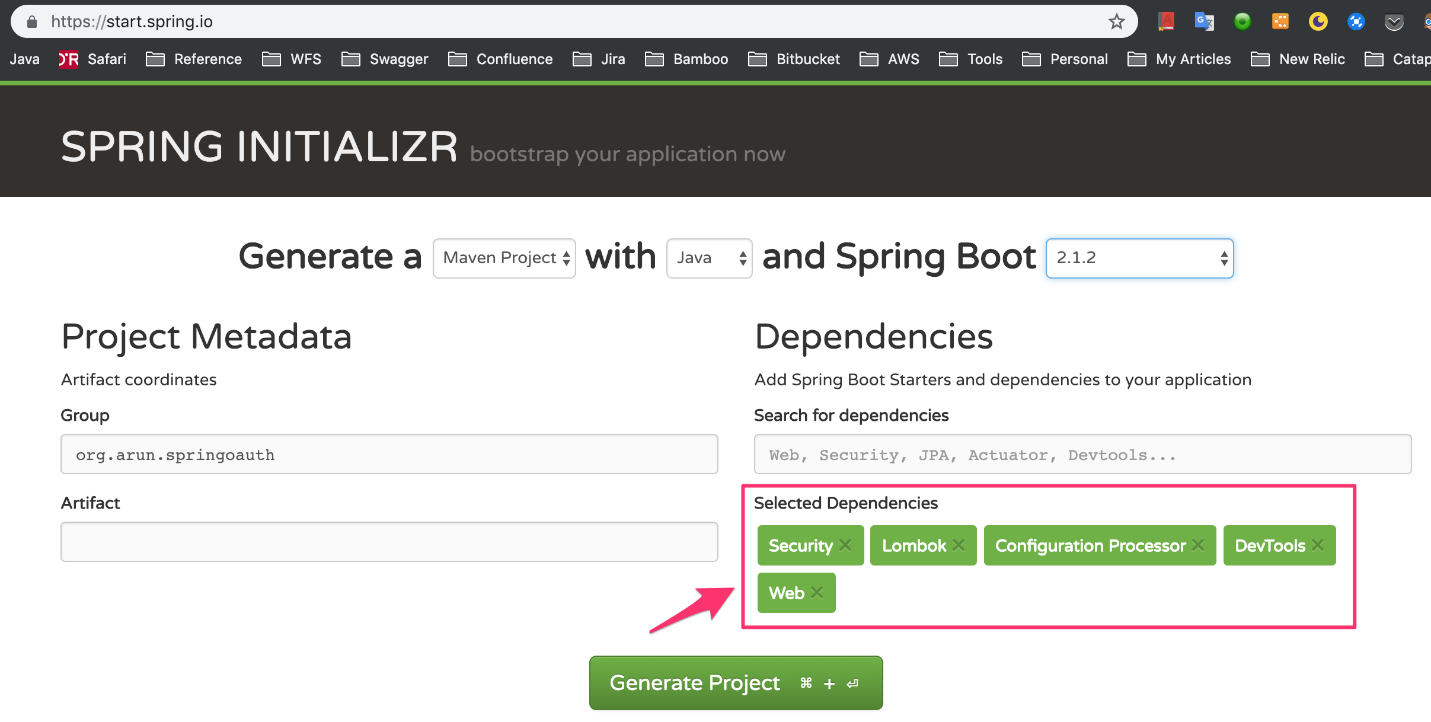


Assign READ\_EMPLOYEE role to user rachel

**9. Create a spring-boot project named spring-oauth2-department-service (The Callee)**

You may create a project using spring initializr or IDE, expectation is to have Web and Security dependencies.

Image for post



spring initializr

***Note:****Lombok, Configuration Processor and DevTools are just helpers. You may use it.*

**Import the project into your IDE and add following dependency for Spring OAuth 2.0 support,**

<!-- Spring OAuth 2.0 Dependency Added Manually -->  
<dependency>  
 <groupId>org.springframework.security.oauth.boot</groupId>  
 <artifactId>spring-security-oauth2-autoconfigure</artifactId>  
 <version>2.0.1.RELEASE</version>  
</dependency>

**You may need to copy following configuration classes from github,**

**Configuration Classes:**

[**JwtAccessTokenCustomizer.java**](https://github.com/bcarun/spring-oauth2-department-service/blob/master/src/main/java/org/arun/springoauth/department/config/JwtAccessTokenCustomizer.java) — Class that takes care extracting user\_name and roles from access token and setting it to spring security context.

[**SecurityConfigurer.java**](https://github.com/bcarun/spring-oauth2-department-service/blob/master/src/main/java/org/arun/springoauth/department/config/SecurityConfigurer.java) — Class that configures OAuth 2.0 Resource Server, Loads CORs configuration, configures HTTP Security and configures OAuth2RestTemplate bean.

[**SecurityContextUtils.java**](https://github.com/bcarun/spring-oauth2-department-service/blob/master/src/main/java/org/arun/springoauth/department/config/SecurityContextUtils.java) — Utility class to get logged in user name. Intended to be used to store created by, last updated by etc.

[**SecurityProperties.java**](https://github.com/bcarun/spring-oauth2-department-service/blob/master/src/main/java/org/arun/springoauth/department/config/SecurityProperties.java) — Configuration properties to enable or disable security, to provide CORs configuration as application properties, to configure OpenID OAuth 2.0 security server URLs.

**10. Create Department Micro-Service Application Class**

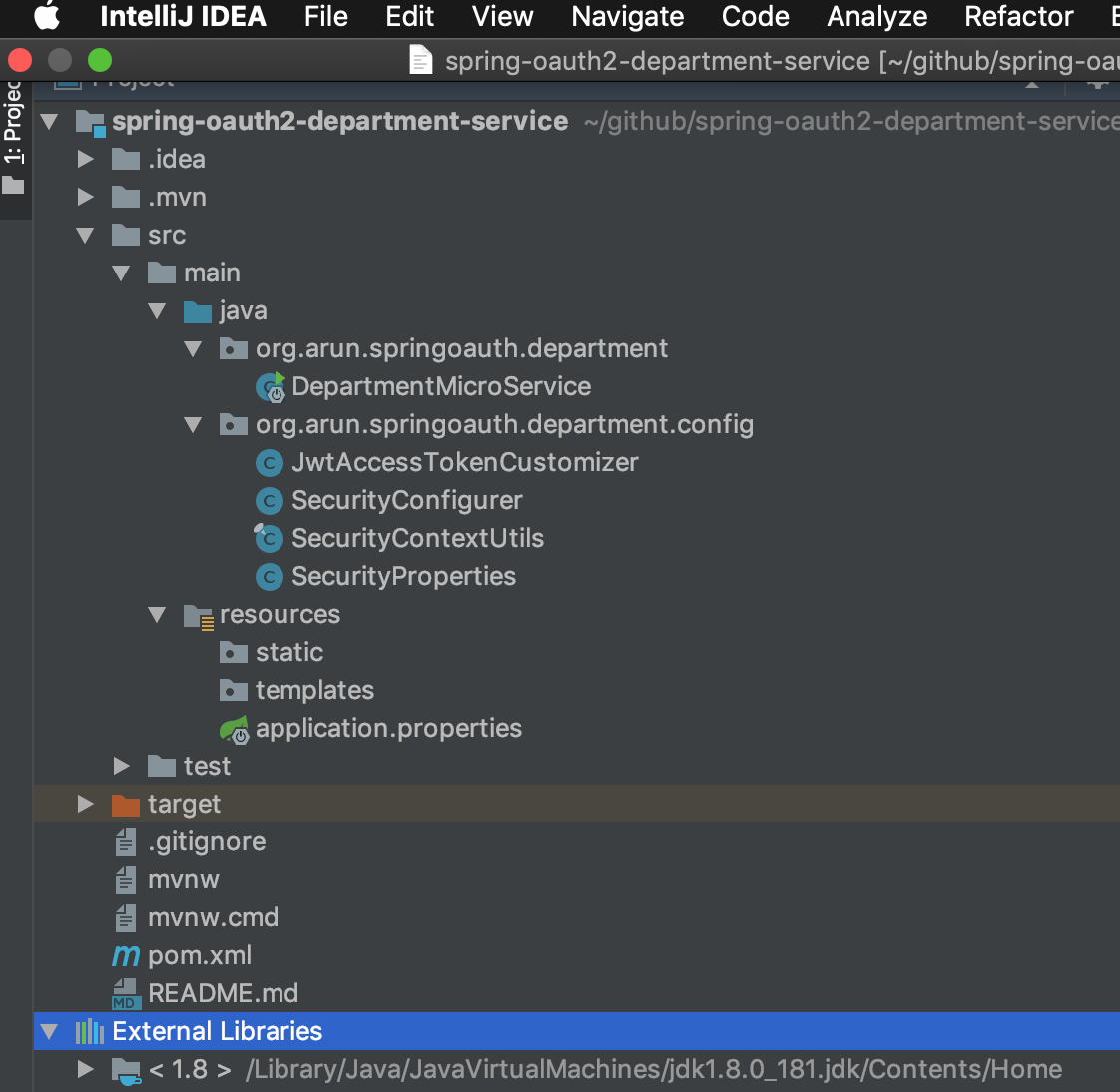
Below code snippet is implementation of DepartmentMicroService.java, a simple REST API secured using **@PreAuthorize** annotation.

**Configure application.properties as stated below,**

*Note: Property ‘****security.oauth2.resource.jwt.key-value****’ may need to be replaced with what is there in your Keycloak realm Public key.*

**Finally, project structure looks like the one below,**

Image for post



**11. Create a spring-boot project named spring-oauth2-employee-service (The Caller)**

Follow the same steps done before,

1. Using Spring Initializr to create a spring-boot project with Web and Spring Security dependencies
2. Add spring-security-oauth2-autoconfigure dependency
3. Copy configuration classes JwtTokenCustomizer.java, SecurityConfigurer.java, SecurityProperties.java, SecurityContextUtils.java.

**Note how OAuth2RestTemplate is configured in SecurityConfigurer.java**

**12. Create Employee Micro-Service Application Class**

***Note 1:****See OAuth2RestTemplate is Autowired in DepartmentRestClient class.*

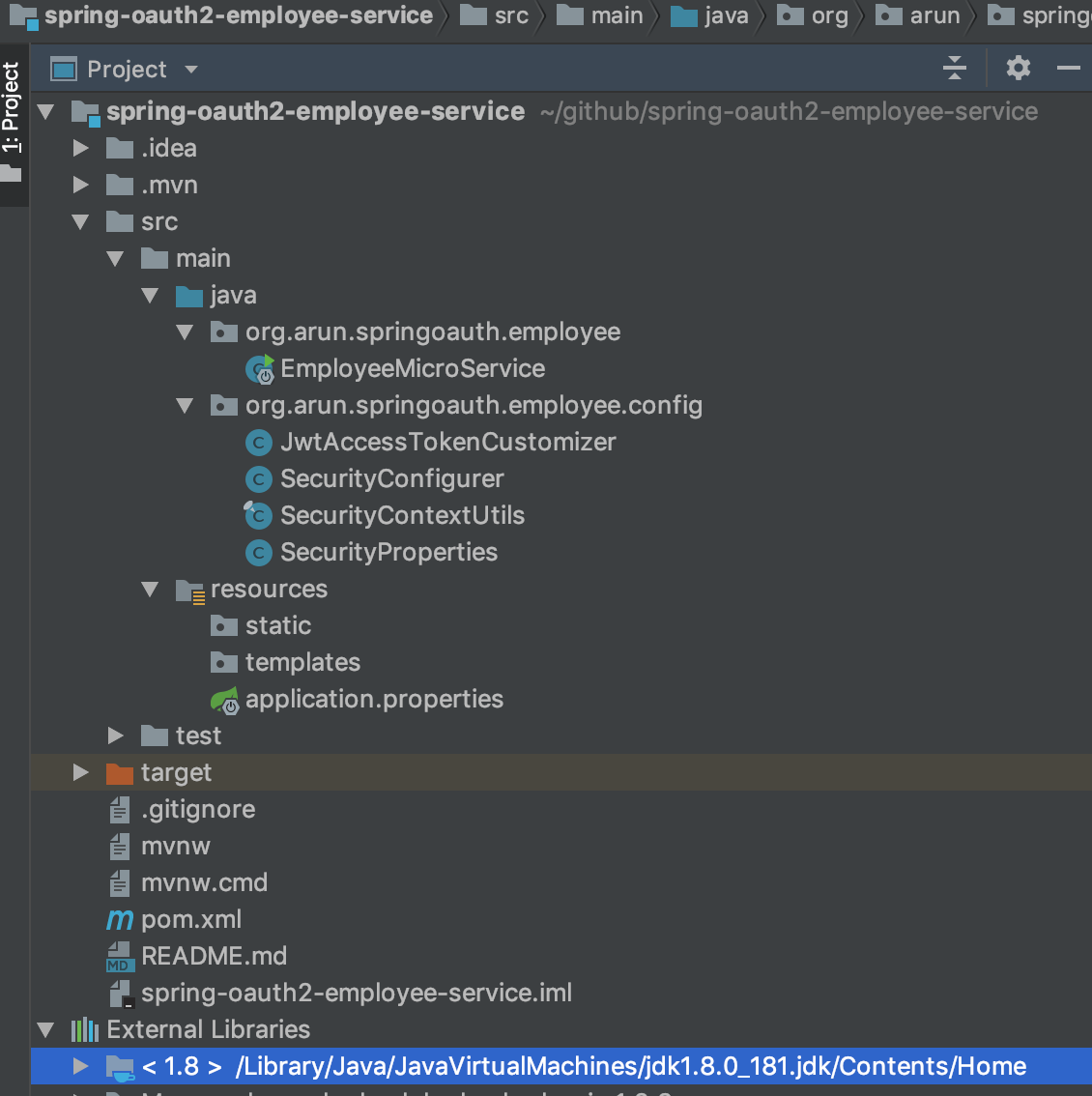
***Note 2:****In order make it easier to read, all the classes are written in same src file, you may write them separately and non-static.*

**Configure application.properties as stated below,**

***Note :****Properties starting****security.oauth2.client.\*****are only required if a micro-service calls another secure micro-service using OAuth2RestTemplate.*

**Finally, project structure looks like the one below,**

Image for post



**13. Running and Testing**

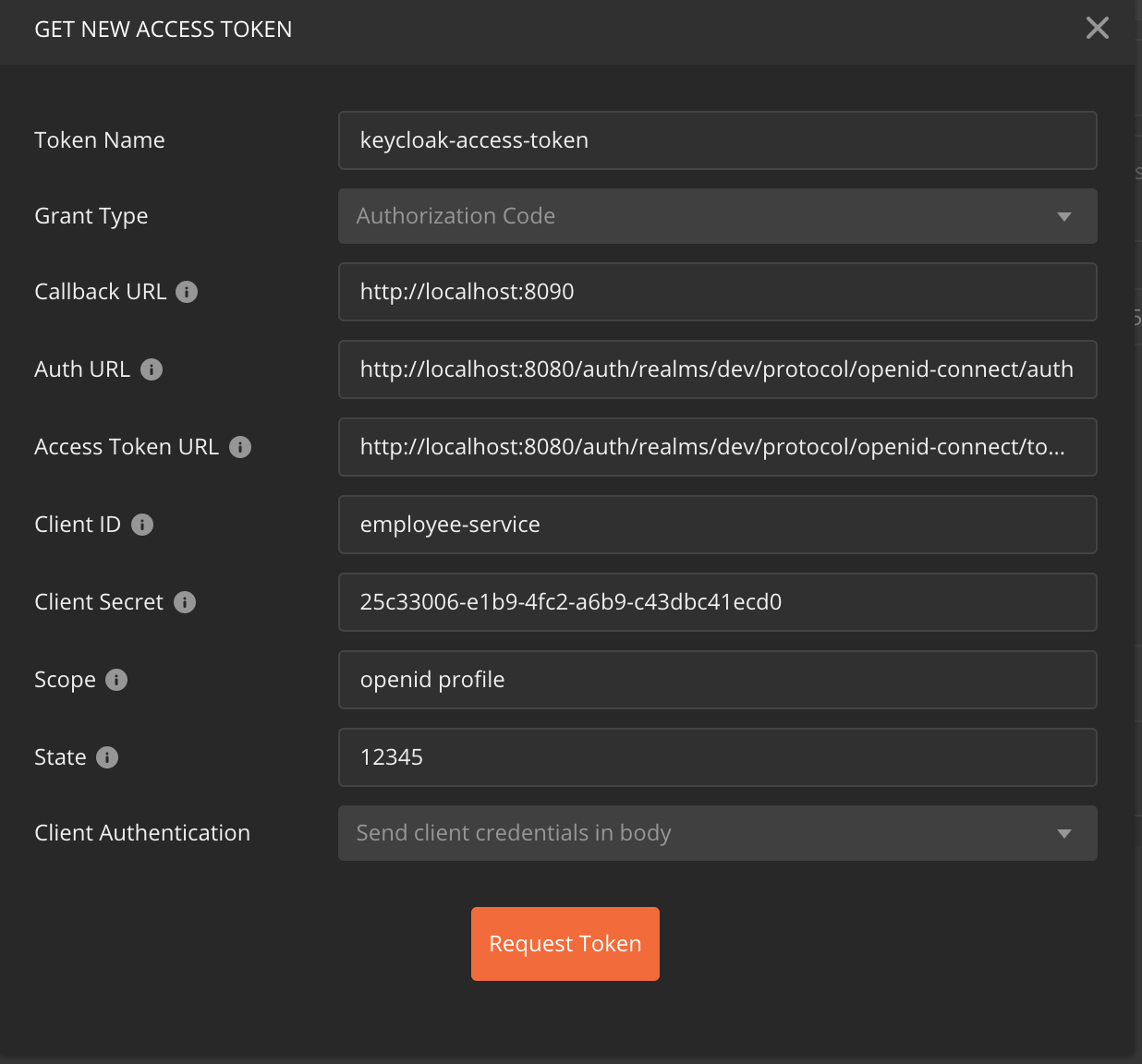
**Now, run auth service and micro-services**

1. Run Keycloak on Port 8080
2. Run spring-oauth2-department-service on Port 8095
3. Run spring-oauth2-employee-service on Port 8090

Note: We will use [Postman](https://www.getpostman.com/) to test REST API.

**Use Postman to get access token**

Image for post



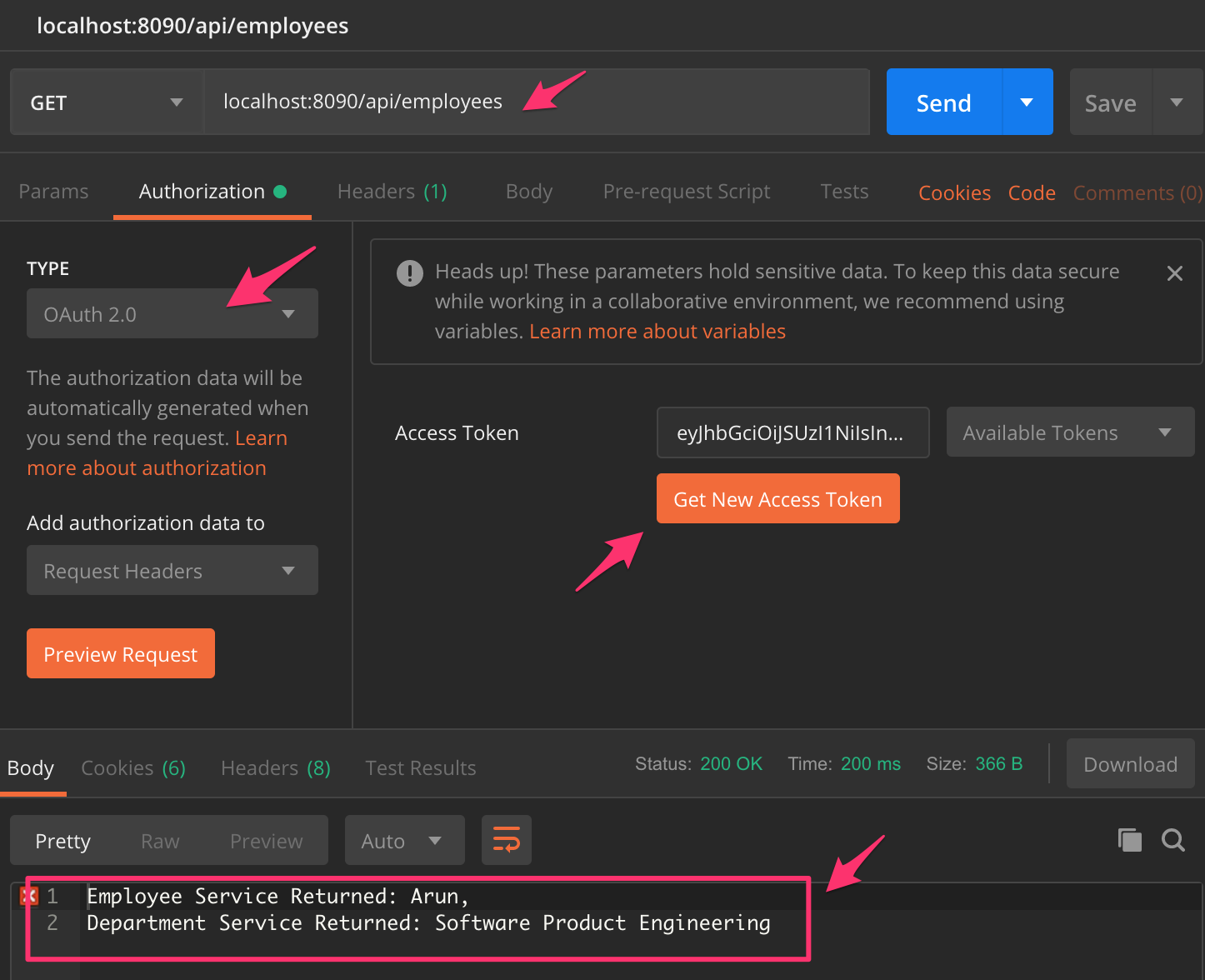
Postman OAuth 2.0 Authorization — Get Access Token configuration

***Note:****If postman prompts for userid/password, you may use the userid and password setup earlier.*

*Note: Once authenticated, you will see access token. You may scroll down and click on ‘Use Token’*

**Testing GET /api/employees Using Postman,**

Image for post



You may inspect the token contents in [JWT](https://jwt.io/). If you watch close, user: rachel had only READ\_EMPLOYEE role, but the call to **GET/api/departments/1** was successful. It is because, before calling GET /api/departments/1 OAuth2RestTemplate got a new access token using Service Account of employee-service.

**Source Code at Github**

[spring-oauth2-employee-service](https://github.com/bcarun/spring-oauth2-employee-service)

[spring-oauth2-department-service](https://github.com/bcarun/spring-oauth2-department-service)