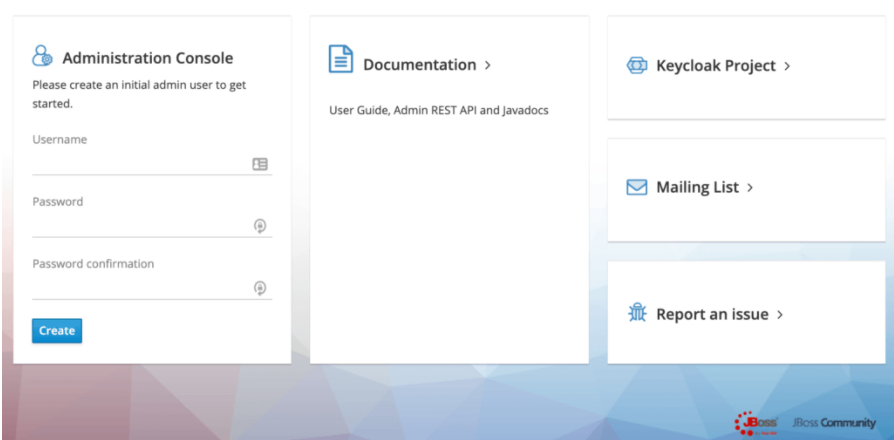
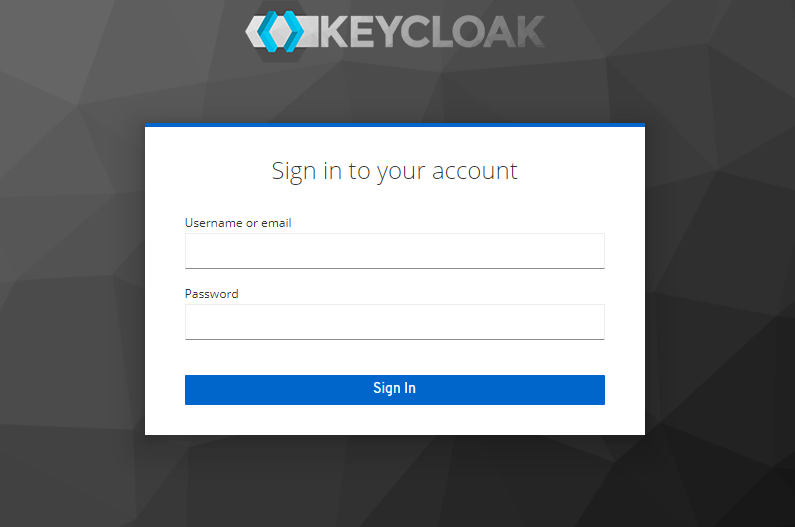
# **Configure Keycloak**

* Now go to the following link <http://localhost:8080> and it will redirect to <http://localhost:8080/auth>.
* Register Yourself in the keycloak.



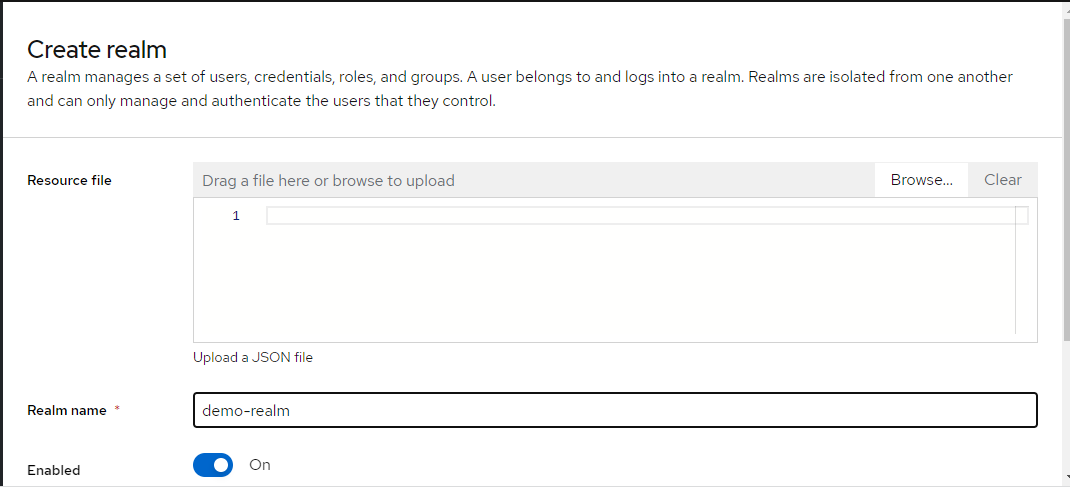
* After entering your credentials you’ll be prompted to admin login page.



* Enter your admin name and password and you’ll be redirected to keycloak dashboard.

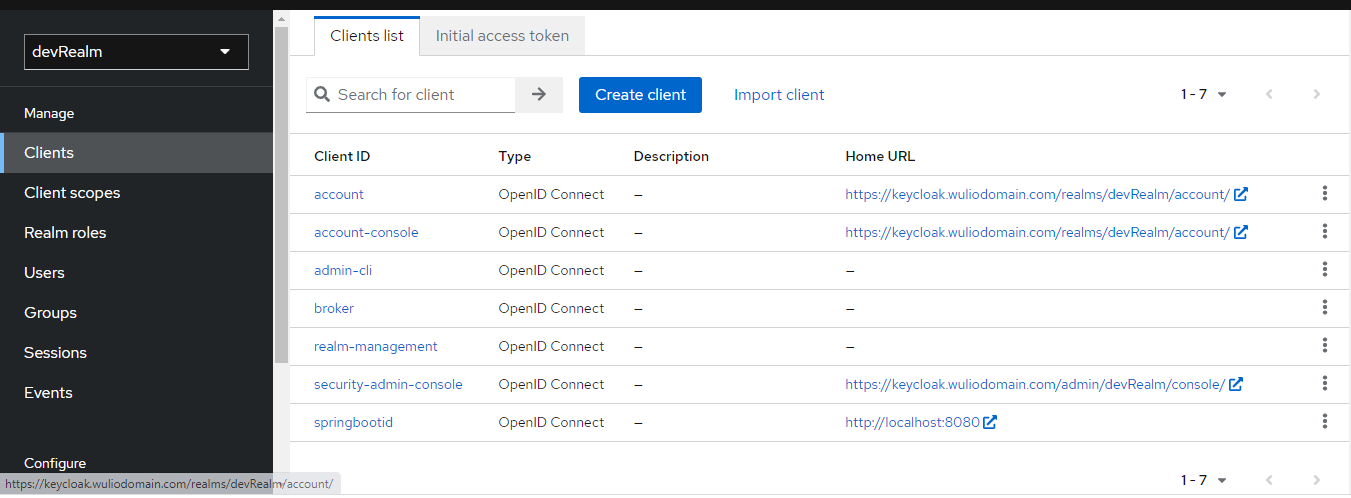
# **Create Realm**

* From the Master drop-down menu, click Add Realm. When you are logged in to the master realm this drop-down menu lists all existing realms.
* Type <realm-name> in the Name field and click Create.



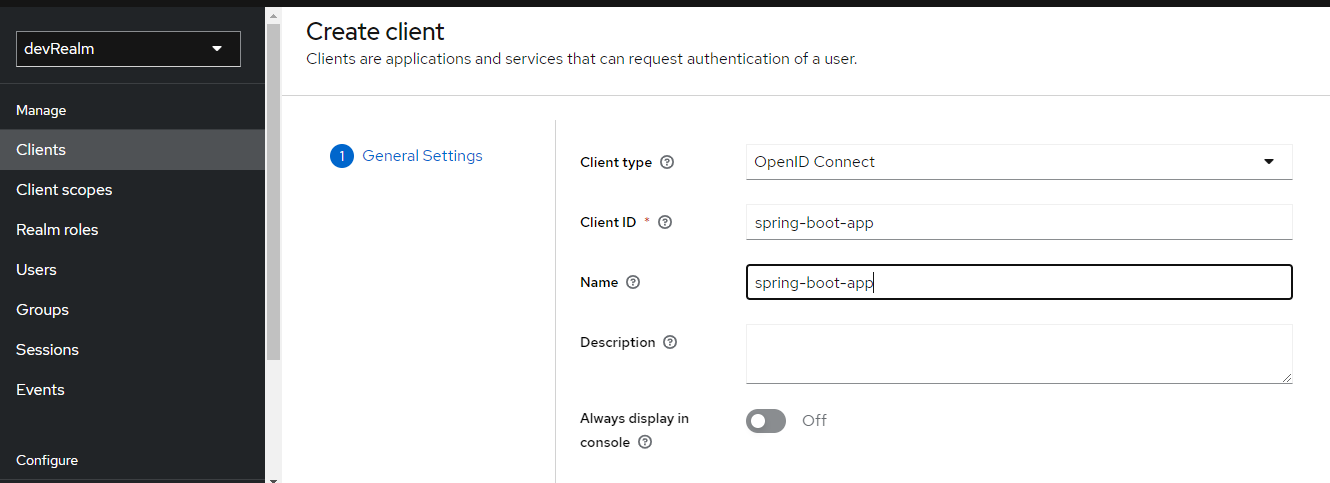
# **Create Client**

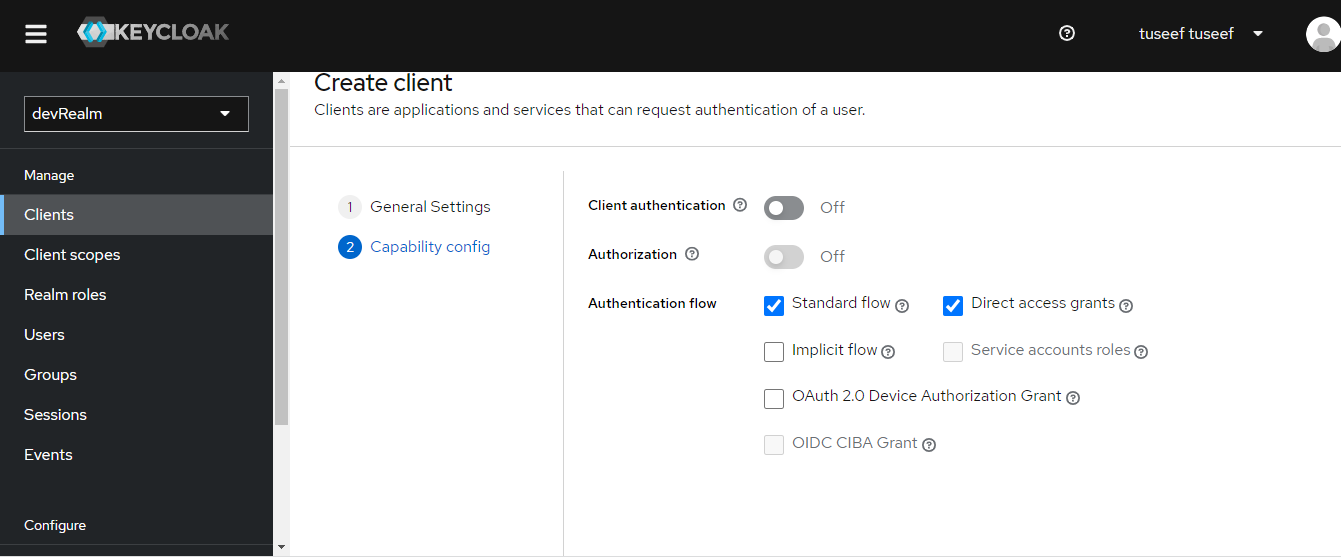
* Click on the Clients menu from the left pane. All the available clients for the selected Realm will get listed here.



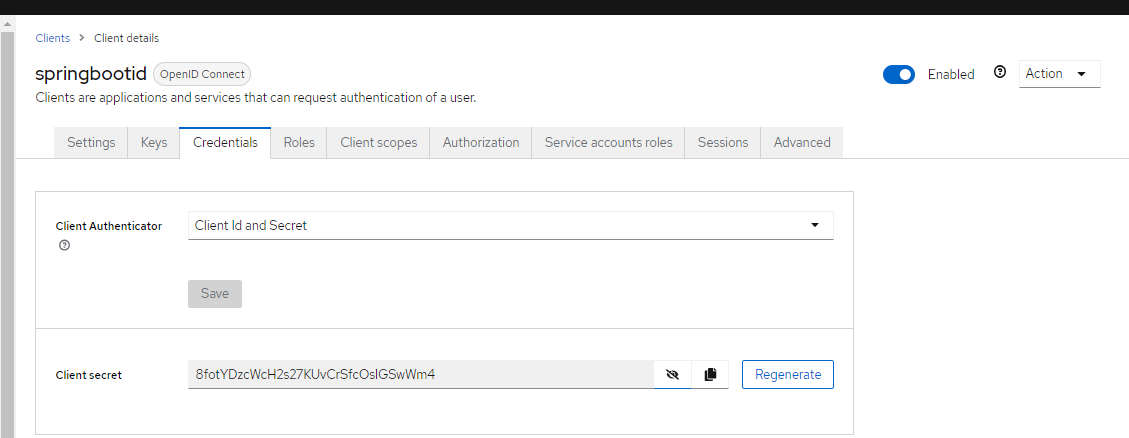
* To create a new client, click Create. You will be prompted for a Client ID, a Client Protocol and a Root URL. A good choice for the client ID is the name of your application (springboot-microservice), the client protocol should be set to openid-connect. After clicking next you will be presented with the client configuration page where you can assign a name and description to the client if desired.

Set the Access Type to confidential, Authorization Enabled to ON, Client Authentication Enabled to ON and click Save. Set root URL as your springboot application URL.



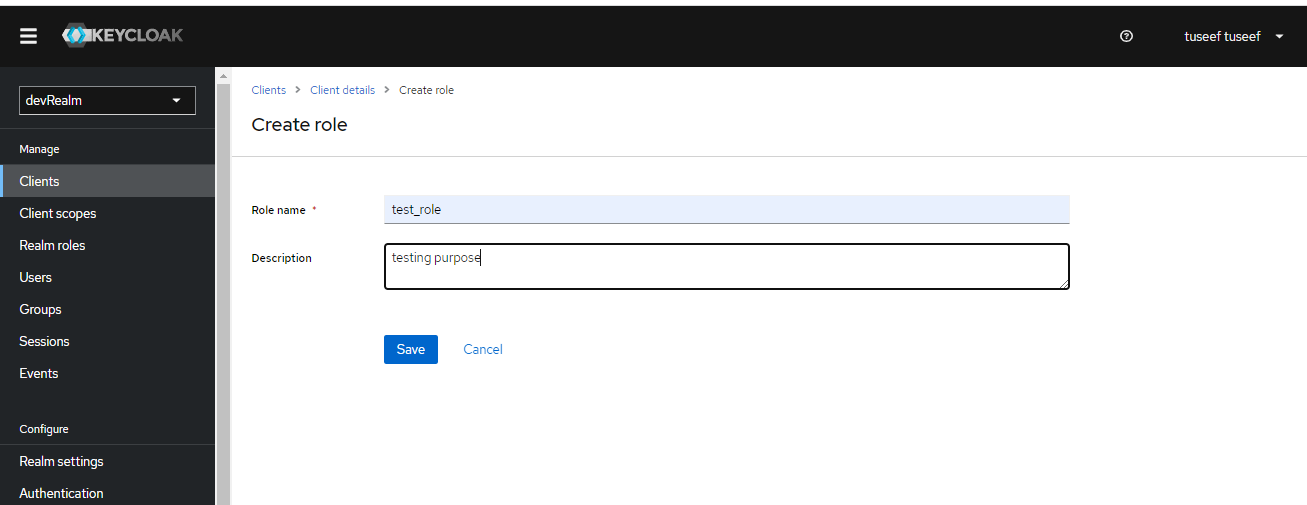


* Credentials tab will show the Client Secret which is required for the Spring Boot Application Keycloak configurations.



**Create Client Roles**

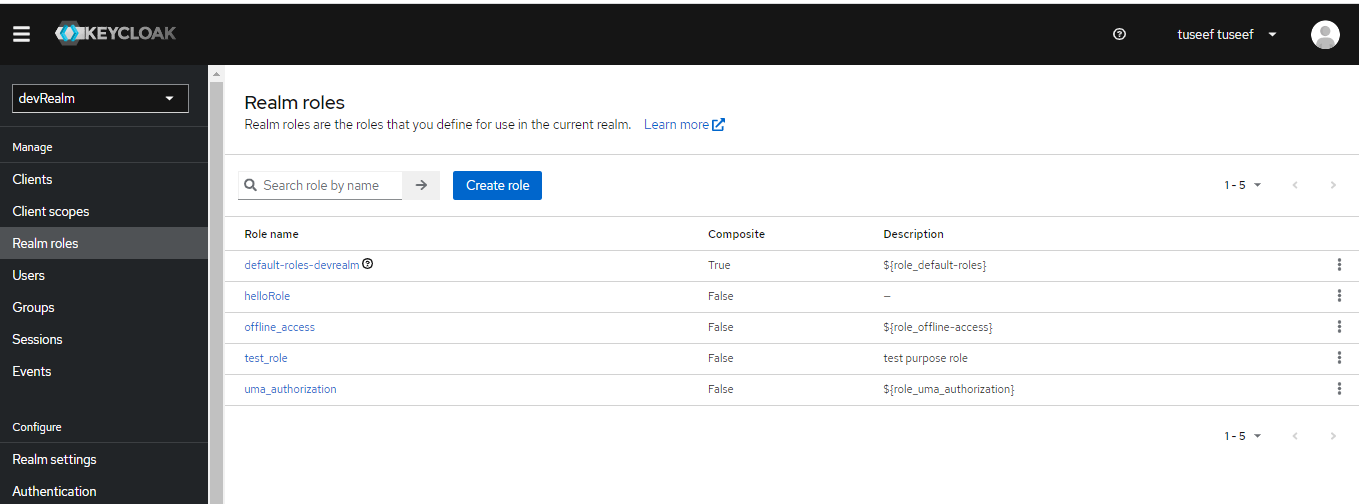
* Go to Client Roles tab to create the springboot-microservice role definitions. Imagine the Application that you are building will have different types of users with different user permissions. Ex: users and administrators.
  + Some APIs would only be accessible to users only.
  + Some APIs would be accessible to administrators only.
  + Some APIs would be accessible to both users and administrators.



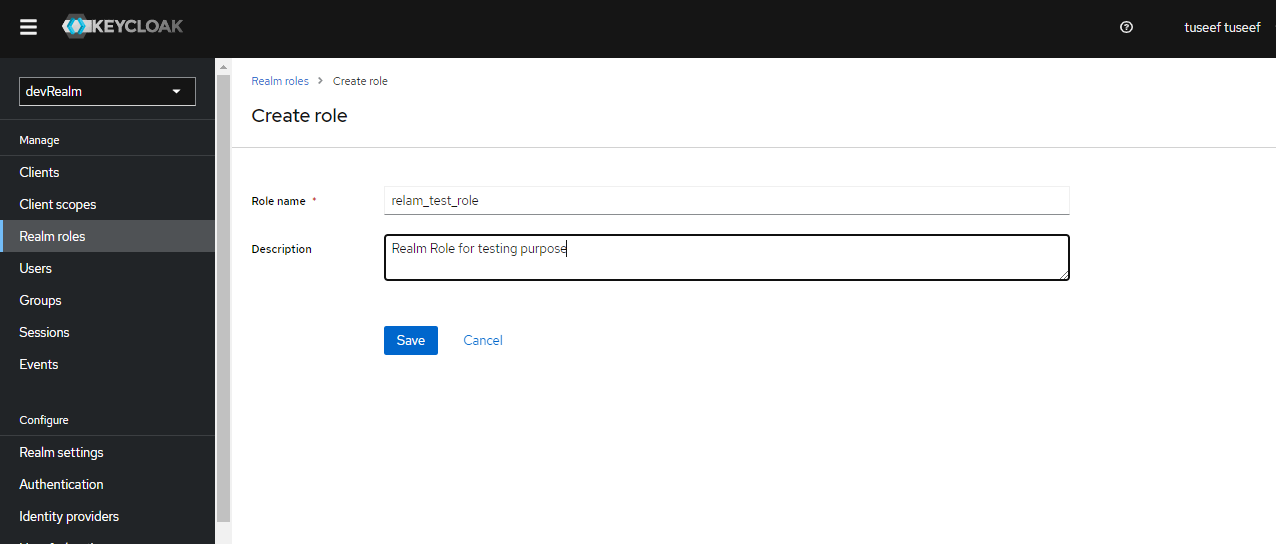
# **Create Realm Roles**

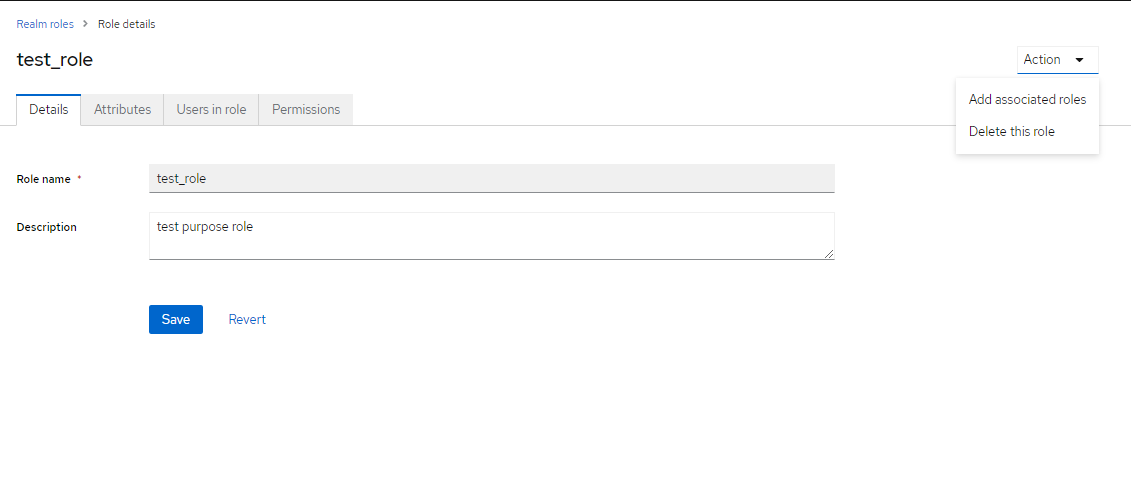
Applications often assign access and permissions to specific roles rather than individual users as dealing with users can be too fine grained and hard to manage.

* Click on the Roles menu from the left pane. All the available roles for the selected Realm will get listed here.

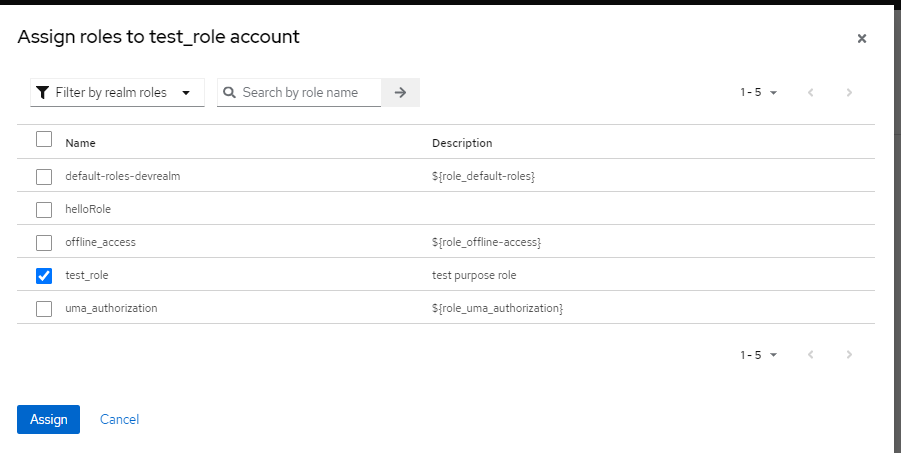


* To create app-user realm role, click Add Role. You will be prompted for a Role Name, and a Description. Provide the details as below and Save.





* After Save, click Action button then click Add Associated roles and Search for springboot-microservice under Client Roles field. Select role of the springboot-microservice and Click Add Selected and click assign button.

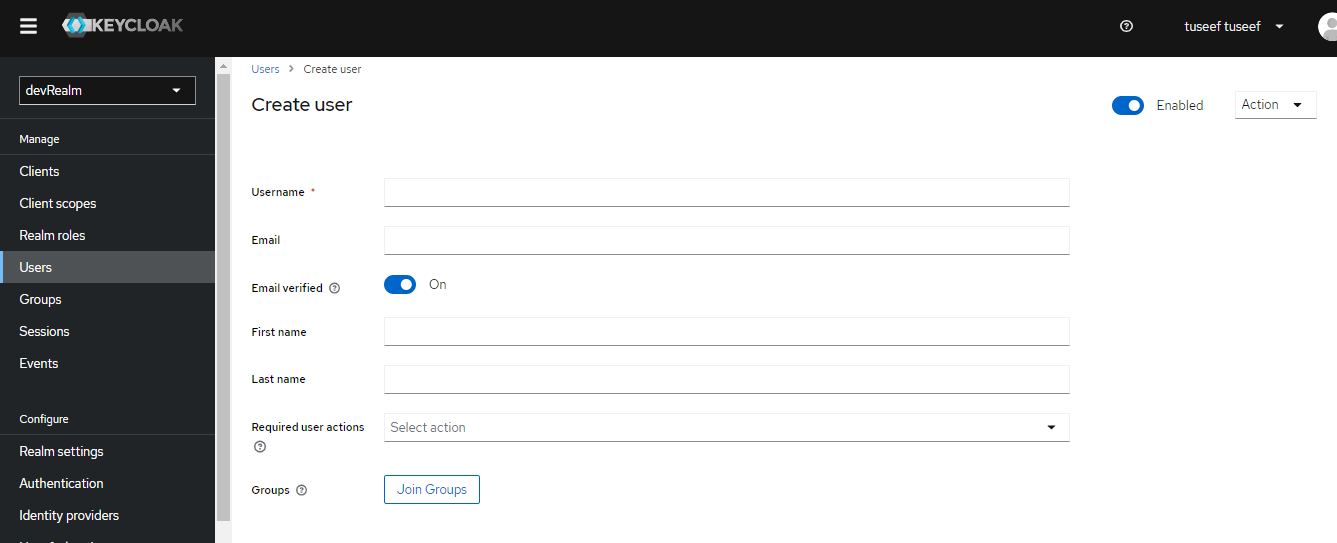


This configuration will assign springboot-microservice client role to the realm role. If you have multiple clients with multiple roles, pick and choose the required roles from each client to create realm roles based on the need.

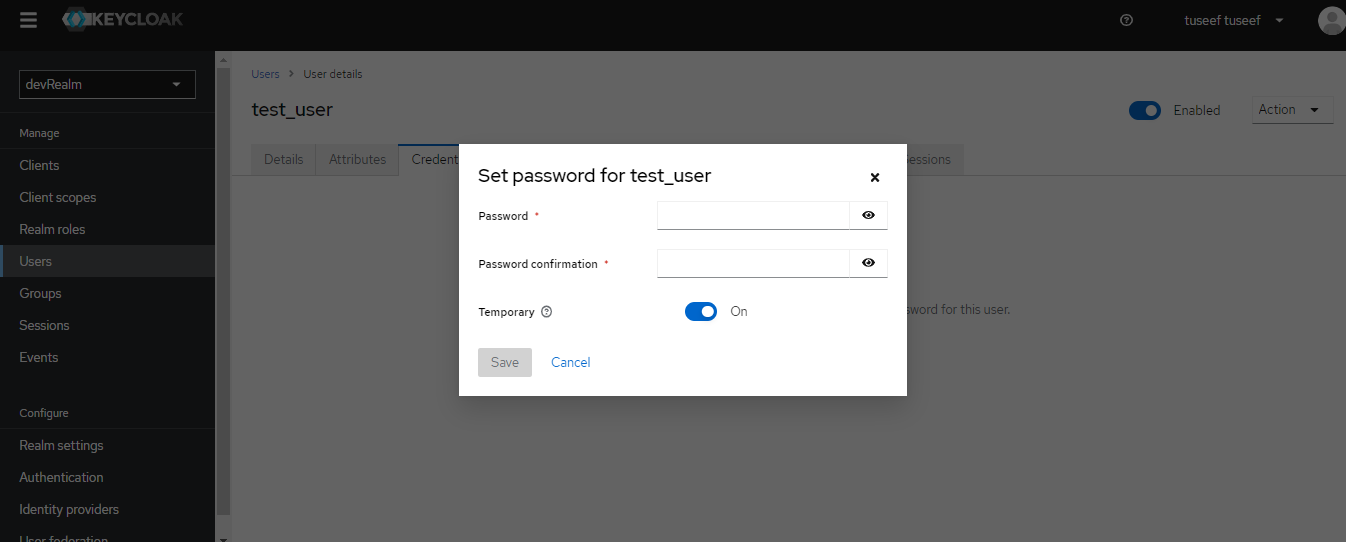
# **Create Users**

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 birth day. They can be assigned group membership and have specific roles assigned to them.

* From the menu, click Users to open the user list page.
* On the right side of the empty user list, click Add User to open the add user page.
* Enter a name in the Username field; this is the only required field. Flip the Email Verified switch from Off to On and click Save to save the data and open the management page for the new user.

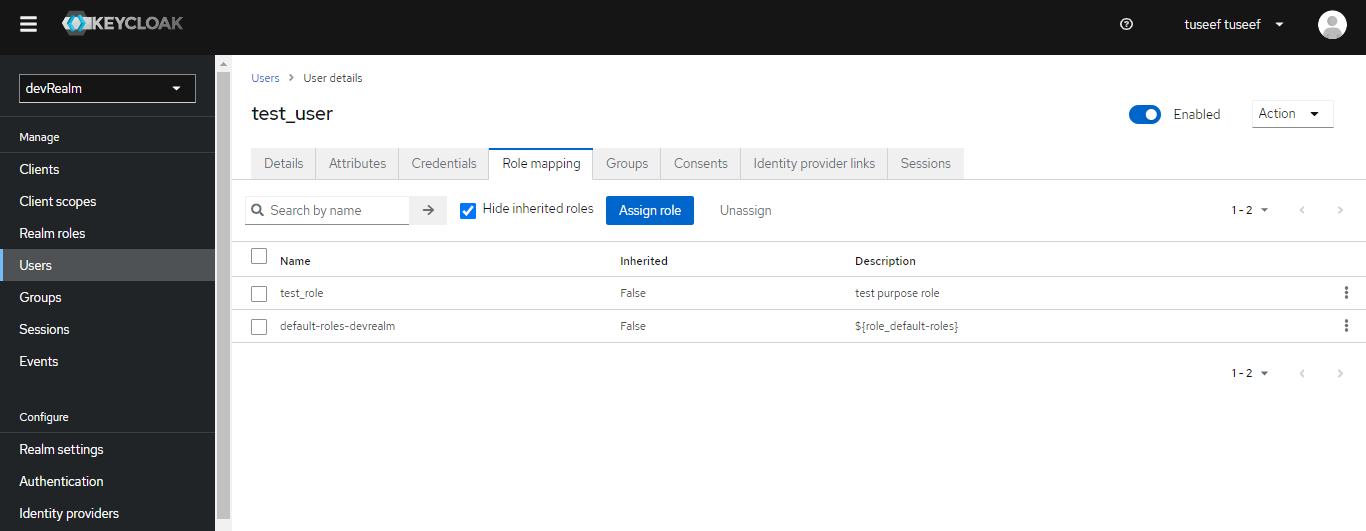


* Click the Credentials tab to set a temporary password for the new user.
* Type a new password and confirm it. Flip the Temporary switch from On to Off and click Reset Password to set the user password to the new one you specified.



* Click the Role Mappings tab to assign realm roles to the user. Realm roles list will be available in Available Roles list. Select one required role and click on the Add Selected > to assign it to the user.

After role assignment, assigned roles will be available under Assigned Roles list.



# **Get Access Token using postman**

Keycloak provides a REST API for generating and refreshing access tokens. We can easily use this API to create our own login page.

First, we need to acquire an access token from Keycloak by sending a POST request to this URL:

http://localhost:8080/realms/<realm-name>/protocol/openid-connect/tokenCopy

The request should have this body in a *x-www-form-urlencoded* format:

client\_id:<your\_client\_id>

username:<your\_username>

password:<your\_password>

grant\_type:passwordCopy

In response, we'll get an *access\_token* and a *refresh\_token*.

{

    "access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJyMFBsZXpLc2x1b2NhVzA2RF9jVFdlNDZjaU9rN0haVFFhVFBocmZEUUt3In0..ZNAn\_nmdjBSX7X0J7v7KFtNzo9y2AuXnWoZhPJ4hDm5MyW1Ae\_AjcNYJ0P4w4bG9v5o-wUczk4nv4-vz4ZDa6XiVLTJ5-JOxwBbRzS7-B7fHivw65UGEpu1IGkcFdu0KAFgbVnAoaEvBE7xF2E-m\_Kbwtk6G\_d6Vf97Q6cou4Utm\_\_LgBerNOBR3Mk-zNuJXkaeWMW6PdXqRxNwHBfIlcBywDi6Zn8gSh4A6Z3-xV6RAZa79LWnC4QvF4-8rYp5Gd5PC6H8RoYObpG-Dqu0Z1lbAPnYsTpkcrsipWTNm0KwM2kGhblAFx2O0-Je\_KYed7Zmu1bChOKkMWK9GvhcbiA",

    "expires\_in": 300,

    "refresh\_expires\_in": 1800,

    "refresh\_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIyOTUxZTZiNS0xMzgzLTQxNGItOTkyYS04NzZjMjRkZmI2ZDcifQ..5mmHWD3cqh4y8KYMVXPP0qew3fhx0O9I2pjEM5B3CMk",

    "token\_type": "Bearer",

    "not-before-policy": 0,

    "session\_state": "6830e489-7342-429b-a93b-b57cde96483e",

    "scope": "profile email"

}

# **Configure Keycloak in project**

# **Adding dependencies**

* Add following dependencies in build.gradle file.

implementation 'org.springframework.boot:spring-boot-starter-security'  
implementation 'org.keycloak:keycloak-spring-boot-starter:20.0.2'  
implementation 'org.keycloak:keycloak-spring-security-adapter:20.0.2'

* Open **application.properties** file and add following properties in it with your credentials.

keycloak.realm = devRealm //(Your realm name)

keycloak.auth-server-url = https://keycloak.wuliodomain.com/

keycloak.ssl-required = external

keycloak.resource = springbootid //(Your service/client name)

keycloak.credentials.secret = 8fotYDzcWcH2s27KUvCrSfcOslGSwWm4 //(Your client secret)

keycloak.use-resource-role-mappings = true

spring.main.allow-circular-references=true // This dependency belongs to spring boot objects references

* Now create a class **SecurityConfig** in **src/main/java/package-name/config**, and add these configuration methods in it. These all methods are belong to Keycloak Security.

@Configuration

@EnableWebSecurity

public class SecurityConfig extends KeycloakWebSecurityConfigurerAdapter {

@Autowired  
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
 KeycloakAuthenticationProvider keycloakAuthenticationProvider = keycloakAuthenticationProvider();  
 keycloakAuthenticationProvider.setGrantedAuthoritiesMapper(new SimpleAuthorityMapper());  
 auth.authenticationProvider(keycloakAuthenticationProvider);  
}

@Bean  
public KeycloakSpringBootConfigResolver KeycloakConfigResolver() {  
 return new KeycloakSpringBootConfigResolver();  
}  
  
@Bean  
@Override  
protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {  
 return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());  
}  
  
@Override  
protected void configure(HttpSecurity http) throws Exception {  
  
 super.configure(http);  
 http.cors().and().csrf().disable().authorizeRequests().anyRequest().authenticated();  
  
}

// This Bean is related to CORS Configuration  
@Bean  
CorsConfigurationSource corsConfigurationSource() {  
 CorsConfiguration configuration = new CorsConfiguration();  
 configuration.setAllowCredentials(true);  
 configuration.setAllowedHeaders(Arrays.*asList*("\*"));  
 configuration.setAllowedOrigins(Arrays.*asList*("\*"));  
 configuration.setAllowedMethods(Arrays.*asList*("\*"));  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 source.registerCorsConfiguration("/\*\*", configuration);  
 return source;  
}

# **Securing Rest APIs**

@GetMapping(value = "/")  
public void index(HttpServletResponse response) throws IOException {  
 response.sendRedirect("/welcome");  
}  
  
@GetMapping("/welcome")  
public String welcome() {  
 return "Greetings from Spring Boot!";  
}

In above code “/” endpint is unsecured and it is redirecting to “/welcome” endpoint which is secured. Add more end points which all will be secured.