

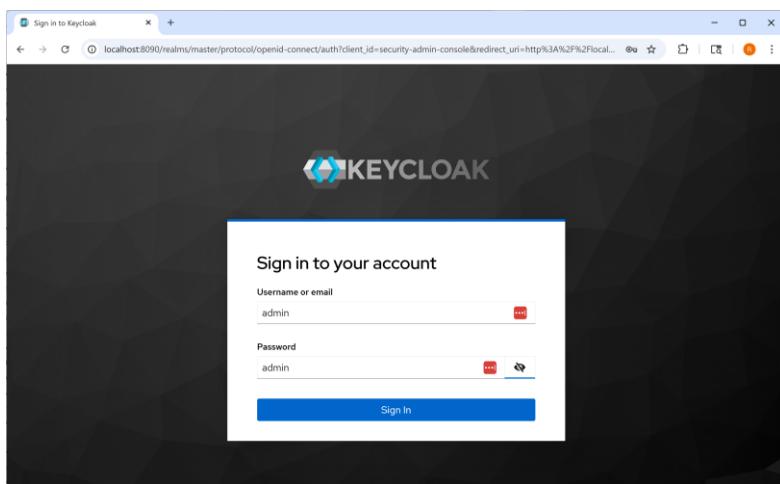
Lab 10

Part 1

First start Keycloak using docker:

```
docker run -d -p 8090:8080 --name keycloak -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:latest start-dev
```

Then open **<http://localhost:8090/>**



Login with Username **admin** and password **admin**.

The screenshot shows the Keycloak administration interface. The top navigation bar includes the Keycloak logo and a 'Current realm' indicator. The left sidebar has a 'Manage realms' section selected, along with other options like 'Clients', 'Client scopes', and 'Realm roles'. The main content area is titled 'Manage realms' and features a search bar, a 'Create realm' button, and a table listing realms. One realm is listed: 'master' (Current realm), with a display name 'Keycloak'.

Name	Display name
master	Keycloak

Select the **Manage realms** from the menu at the left, and click the **Create realm** button.

Create realm

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm.

Realms are isolated from one another and can only manage and authenticate the users that they control.

Resource file Drag a file here or browse to upload

Upload a JSON file

Realm name *

Enabled On

Name the realm **myrealm** and click the **Create** button.

≡ KEYCLOAK

myrealm Current realm

Manage realms

Manage

Clients

Client scopes

Realm roles

Manage realms

Search Create realm Refresh

<input type="checkbox"/> Name	Display name
<input type="checkbox"/> myrealm	Current realm
<input type="checkbox"/> master	Keycloak

The realm **myrealm** is now created is the current active realm

≡ KEYCLOAK

myrealm Current realm

Manage realms

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Clients

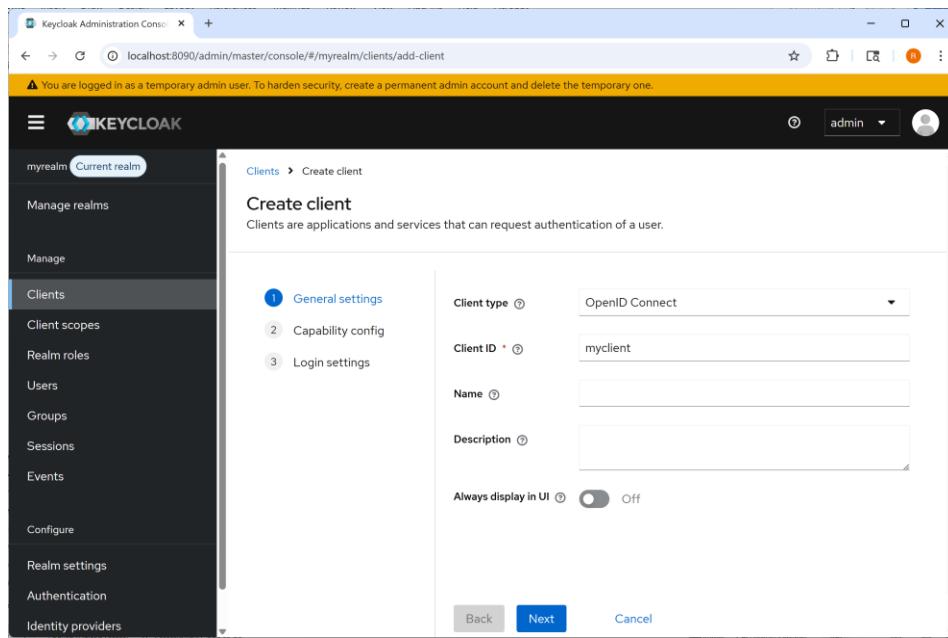
Clients are applications and services that can request authentication of a user.

Clients list Initial access token Client registration

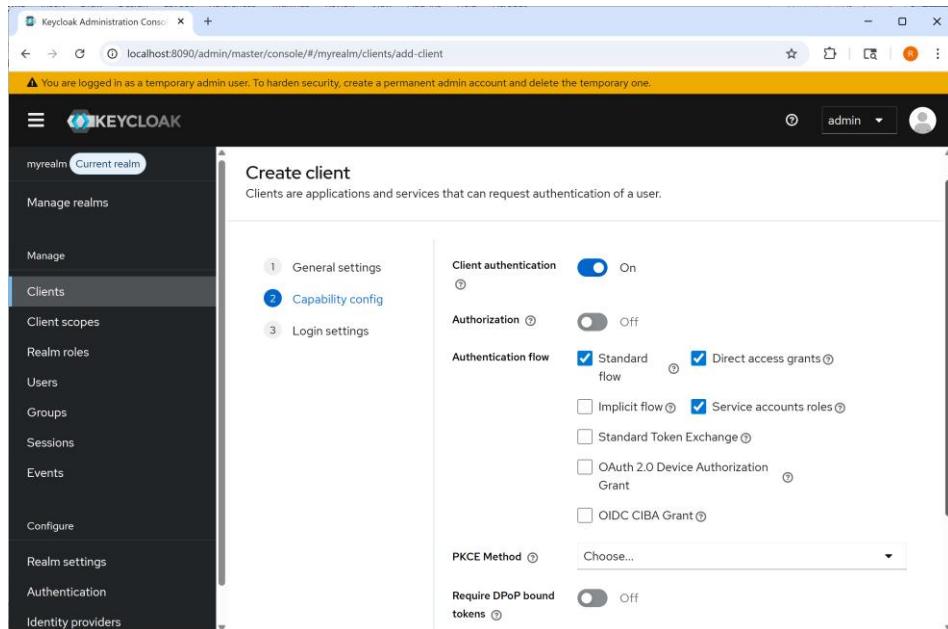
Search for client Create client Import client

Client ID	Name	Type	Description
account	Account	OpenID Connect	–
account-console	Account Console	OpenID Connect	–
admin-cli	Admin CLI	OpenID	–

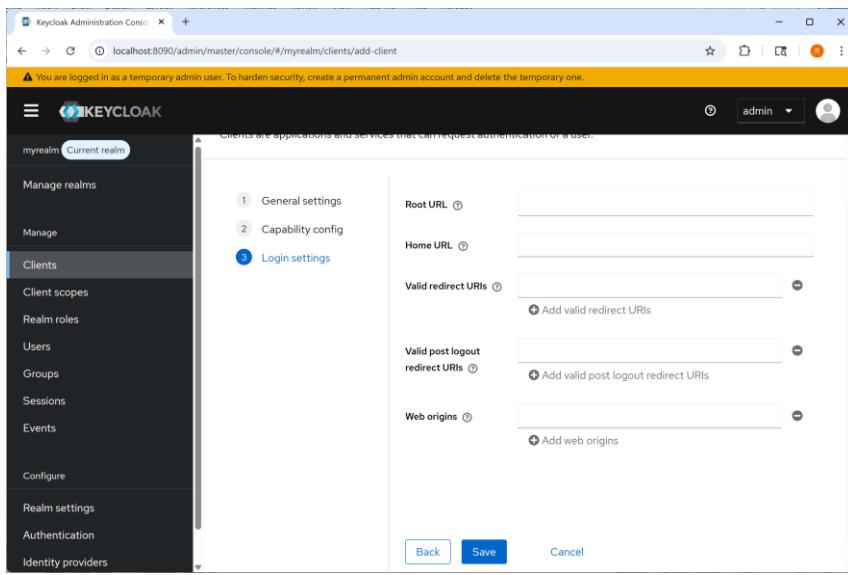
Select the **Clients** from the menu at the left, and click the **Create client** button.



Fill in the Client ID **myclient** and click **Next**



Set **Client authentication** to **On** and select **Direct access grants** and **Service accounts roles**. Then click **Next**



Click Save

Clients > Client details

myclient OpenID Connect

Enabled Action

Clients are applications and services that can request authentication of a user.

Settings Keys Credentials Roles Client scopes Service accounts roles Sessions

Client Authenticator: Client Id and Secret

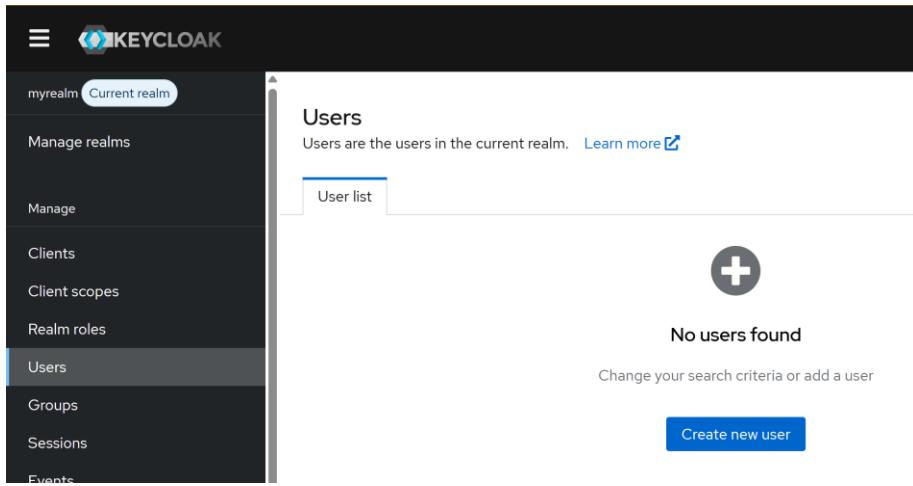
Client Secret: OmaxLeON2d9gwyFlilv52YBxgB5AoXGL

Save Regenerate

Select the **Credentials** tab and copy the **Client Secret**. Save this client secret somewhere (notepad).

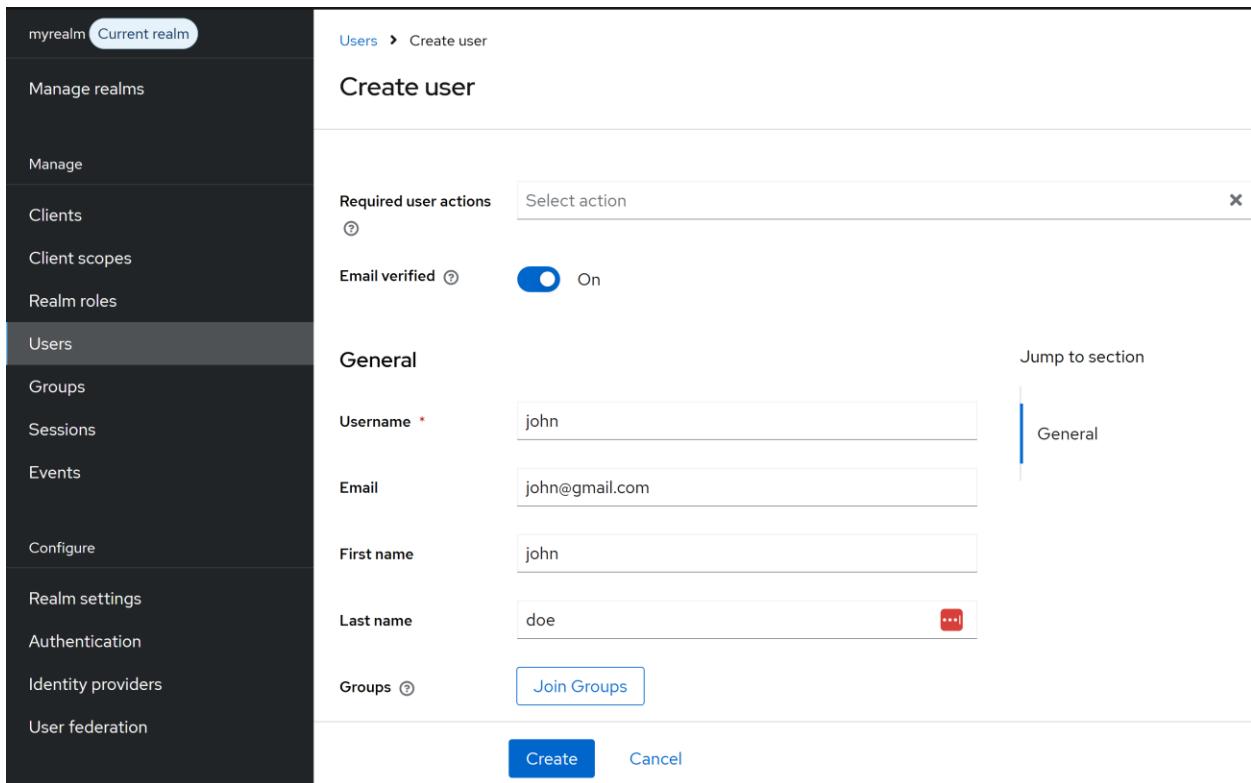
We have now created a client with the Client ID **myclient** and a Client secret.

Now we are going to create Users.



The screenshot shows the Keycloak administration interface. The left sidebar has a dark theme with white text. The 'Users' option is selected, indicated by a blue vertical bar. The main area has a light background. At the top, it says 'myrealm Current realm'. Below that is a navigation bar with 'Manage realms', 'Clients', 'Client scopes', 'Realm roles', 'Users' (selected), 'Groups', 'Sessions', and 'Events'. The main content area is titled 'Users' and contains the sub-instruction 'Users are the users in the current realm.' with a 'Learn more' link. A 'User list' tab is active. In the center, there's a large circular button with a plus sign. Below it, the message 'No users found' is displayed, followed by the instruction 'Change your search criteria or add a user'. At the bottom right is a blue 'Create new user' button.

On the menu on the left select **Users** and click the **Create new user** button.



The screenshot shows the 'Create user' form in Keycloak. The left sidebar is identical to the previous screenshot. The main area is titled 'Create user' and shows the path 'Users > Create user'. The 'General' section is expanded. It includes fields for 'Username' (john), 'Email' (john@gmail.com), 'First name' (john), 'Last name' (doe), and 'Groups' (with a 'Join Groups' button). Above these fields is a 'Required user actions' section with a 'Select action' dropdown and an 'Email verified' toggle switch which is set to 'On'. At the bottom are 'Create' and 'Cancel' buttons.

Fill in the **Username**, **Email**, **First name**, **Last name** and select the **Email verified** option.
Then click **Create**

myrealm Current realm

Users > User details

john

Enabled

Action

Details Credentials Role mapping Groups Consents Identity provider links Sessions Events

ID * e1154adc-0b0b-4054-abe2-9f7363d3ab65

Created at * 11/7/2025, 3:32:12 PM

Required user actions Select action

Email verified On

General

Username * john

Email john@gmail.com

Save Revert

User **john** is created.

myrealm Current realm

Users > User details

john

Enabled

Details Credentials Role mapping Groups Consents Identity provider links Sessions

+ No credentials

This user does not have any credentials. You can set password for this user.

Set password

Manage realms

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Select the **Credentials** tab and click the **Set password** button

Set password for john

×

Password * john

Password confirmation * john

Temporary ⓘ Off

Save Cancel

Fill in the password **john** and set **Temporary** to **Off**. Click the **Save** button.

The screenshot shows the Keycloak 'User details' page for a user named 'john'. The 'Credentials' tab is active, displaying a single password entry. The password is 'My password' and was created at '11/7/2025, 3:37:08 PM'. There are buttons for 'Show data' and 'Reset password'.

User john has now password john.

Now we need to create 2 roles: **user** and **admin**.

The screenshot shows the Keycloak 'Client details' page for a client named 'myclient'. The 'Roles' tab is active. It displays a message 'No roles for this client' and a 'Create role' button. The left sidebar shows other management options like 'Clients', 'Client scopes', and 'Realm roles'.

On the menu on the left, select **Clients** and select the roles tab for client **myclient**. click the **Create role** button.

myrealm Current realm

Clients > Client details > Create role

Create role

Role name *	USER
Description	

Save **Cancel**

Make a **USER** role and click **Save**

Clients > Client details

myclient OpenID Connect

Clients are applications and services that can request authentication of a user.

Role name	Composite	Description
ADMIN	False	-
USER	False	-

In the same way create an **ADMIN** role.

Now go back to the Users on the left menu and go to user john.

myrealm Current realm

Users > User details

john

Name	Inherited	Description
default-roles-myrealm	False	role_default-roles

Select the **Role mapping** tab and click the **Assign role** pull down list.

The screenshot shows the Keycloak Administration Console interface. On the left, there's a sidebar with various management options like 'Manage realms', 'Clients', 'Client scopes', 'Realm roles', and 'Users'. The 'Users' option is selected. In the main area, a user named 'john' is viewed. The 'Role mapping' tab is active. A search bar at the top has 'Search by name' and a 'Hide inherited roles' checkbox. Below that is a table with columns 'Name', 'Inherited', and 'Assignment'. One row is visible: 'default-roles-myrealm' (Inherited: False, Assignment: 'role_default-roles'). At the bottom of the table are '1-1' and navigation arrows. To the right of the table is a dropdown menu labeled 'Assign role' with 'Client roles' and 'Realm roles' options. There are also 'Unassign' and 'Refresh' buttons.

Select **Client roles**.

This screenshot shows a modal dialog titled 'Assign Client roles to john'. It contains a table with columns 'Name', 'Client ID', and 'Description'. The 'Name' column has checkboxes. The 'USER' role is checked and highlighted with a blue border. Other roles listed include 'create-client', 'impersonation', 'manage-authorization', 'manage-clients', 'manage-events', 'manage-identity-providers', 'manage-realm', 'manage-users', and 'query-clients'. Each row also includes a 'Description' column. At the bottom of the dialog are 'Assign' and 'Cancel' buttons.

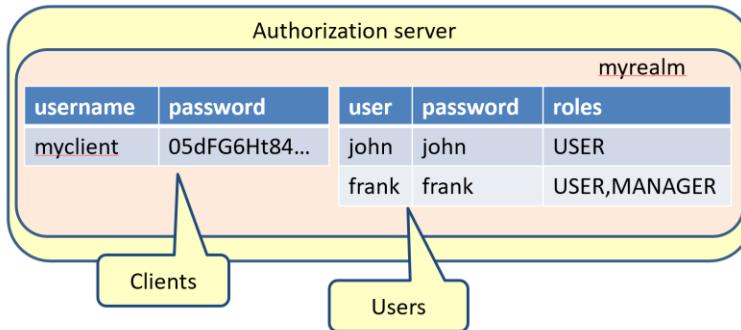
Name	Client ID	Description
<input checked="" type="checkbox"/> USER	myclient	-
<input type="checkbox"/> create-client	realm-management	role_create-client
<input type="checkbox"/> impersonation	realm-management	role_impersonation
<input type="checkbox"/> manage-authorization	realm-management	role_manage_authorization
<input type="checkbox"/> manage-clients	realm-management	role_manage_clients
<input type="checkbox"/> manage-events	realm-management	role_manage_events
<input type="checkbox"/> manage-identity-providers	realm-management	role_manage_identity_providers
<input type="checkbox"/> manage-realm	realm-management	role_manage_realm
<input type="checkbox"/> manage-users	realm-management	role_manage_users
<input type="checkbox"/> query-clients	realm-management	role_query_clients

Select the **USER** role and click **Assign**.

Name	Inherited	Description
myclient - USER	False	-
myclient - ADMIN	False	-
default-roles-myrealm	False	role_default-roles

In the same way create user **frank** with password **frank** and roles **USER** and **ADMIN**

We now have the following configuration in KeyCloak:



Now we are going to retrieve the token for users john and frank.

Open **Postman** and create a **POST** request for the URL

http://localhost:8090/realms/myrealm/protocol/openid-connect/token

In the **Auth** tab, select **Basic Auth** and fill in the **username** and **secret** of the **Client** configured in KeyCloak.

The screenshot shows the Postman interface with a POST request to `http://localhost:8090/realm/myrealm/protocol/openid-connect/token`. The Body tab is selected and set to `x-www-form-urlencoded`. The data is filled in as follows:

Key	Value
<input checked="" type="checkbox"/> grant_type	password
<input checked="" type="checkbox"/> username	frank
<input checked="" type="checkbox"/> password	frank
<input checked="" type="checkbox"/> scope	openid roles
<input type="checkbox"/>	
<input type="checkbox"/>	

Below the table, the response status is `200 OK` with a duration of `102 ms` and a size of `3.53 KB`. The response body is displayed as JSON:

```

1 {
2   "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSlDOIiwia2lkIiA6ICJyeHBmMDJqMWF0UlgsS012SHdLTnZXVFoxcmdzOHdDY1h5RENka11DamtfIn0.eyJleHAiOjE3NjI1NTcxODksImhdCI6MTc2MjU1Njg4OSwanRpIjoib25ydHJv0djzZWI2ZWVmLT0ZWltMjZlyi0xZjI0LWiwZGM0YTJmYmVl0CisInlzcIyI6Imh0dHA6Ly9sb2Nhbgvc3Q6ODA5MC9yZWFsbXvbXlyZWFsbSIsImF1ZC6ImFjY291bnQ1LCJzdWlIi0ixNDhiYTY3Mi0xOWIzLTQ4MjAtYjAwZS05MTVmYzQ4MjB1M2MiLCJ0eXAi0iJCZWFyZXIlCJhenAi0iJteWNsaWVu dCIsInNpZC6i1jMwZmRhY2E4LWRiZD6tjA3YY00MDViLWM5ZmU5YmQ4MWVMSIsImFjciI6Ij EilCJhbGxvd2VklW9yaWdpbnMiolsilyoiXSwicmVhbG1fYWnjZXNzIjp7InJvbGVzIjpbImR1 ZmF1bhQtcm9sZXMtbXlyZWFsbSIsIm9mZmpbmVfYWNjZXNzIiwidW1hX2F1dgHvcml6YXRpB2 4ixX0sInJlc291cmN1X2fjY2VzcylIeoy3teWNsaWVudC16eyJyb2xlcyI6NyJBRE1JT1IsI1VT RVIiXX0sImFjY291bnQionsicm9sZXMi0lsibWFuYwd1LWFjY291bnQ1LCJtYW5hZ2UtYWNjb3 VudC1saW5rcyIsInZpZXctcHJvZmlsZSJdfX0sInNjb3BlIjoib3BlbmkIGVtYWlsIHByb2Zp bGUilCJ1bWFpbF92ZXJpZml1ZC16dHJ1ZSwibmFtZSI6ImZyW5rIGJyb3duIiwiCHJZmVycmVkX3VzxJuYW1ljoiznJhbmsiLCJnaX2lb19uYW1lIjoiznJhbmsiLCJmYW1pbHfbmFtZS16"

```

In the **Body** tab, select **x-www-form-urlencoded** and fill in:

- `grant_type = client-credentials`
- `username = frank`
- `password = frank.`

The click the **Send** button, and see that we receive an access token back from the authorization server.

Copy the access token and go to jwt.io

Get up-to-speed with JSON Web Tokens. Get the [JWT Handbook](#) for free ↗

[JWT Debugger](#) [About JWT](#) [See JWT libraries](#)

store or transmit [Debugger](#) [Web](#) [Introduction](#) [of the](#) [Libraries](#) [Ask](#)

English

JWT Decoder JWT Encoder

Paste a JWT below that you'd like to decode, validate, and verify.

Enable auto-focus

ENCODED VALUE

```
eyJhbGciOiJSUzI1NiIsInR5cCigOia1sIdUiwia2lkIia6IC3yeHbmDqMWF
OU1gxS0125Hd1TnZVfocmdzOHdDV1h5RENa1lDamtFTn0.eyJleHA1oJE3Nj
I1N1cx0OKs1m1hdC16MtC2MjU1Njg40SwianRpjoib25ydJv0jdjZW12ZWmI
TV02WtMjZ1Y10xZj18LNIwZGM0Y73mVnLOCIsIm1zcyl61mh0dHA6Ly9sb2Nn
bGhv3c3Q6ODA5MC9yZWFsbXnb1yJyZWFsbs1s1mF1ZC161mFjY291bnQ1LC1zdwI
i011xNdh1iyTY3M10x0W0IzLTQ4MjAtyJaNz50SM7VmYzQ4MjB1M2M1LC30eXA1o1
JCZuFyZX1iLCJhenAi0iJteNsawVudIsInpZC161mNjZmRyE4LWR1ZD6tY
jA3Yy0M0DV0ILWMSZnUSYmQ4MWkMSisImjciI61jEi1Cjhbxvd2vLw0yaikdp
bnM1o1s1yoIxSwicmhbG1FYWnjZxNz1jptInVbGVz1jb1mR1ZmF1bHQtcm9
sZ0XtbXlyZWFsb5IsIm9mZmxbmVYFNWjZKNz1i1wd1hX2f1Ghvcml6YXRpB2
41X0sIn1l291cm1x2FjY2VzcI6eyJteNsawVudC16eyJyb2x1cyI6WyJBR
E1J7i1s1lVTRV1XX0s0ImfjY291bnQ1onsicm9sZM10s1wfWuYd1lWFjY291
bnQ1LCjtYw5hZ22ltYwNb1jb3VudC1aw5rcyIsInz2p2XcthvZns1Z53dfX0sInN
jb3811joiib38lbm1kIGVtYwlsIHByb2ZpbGU1c1jbwPfb922XjpZm1lZC16dH
j1Z5w1bmftZS161m2yW5fIGJyb3du1wi1chH12mVycmVXK3Vz2XjuVw11IjoiZ
nJhms1lCJnaXZ1b19uYm11jjo1znjhbm1lCmYw1pBHlfmftZS161mjb3du
I1w1Zw1haw1o1jmcfua0BnbwFpbC5jb20ifQ.RklrqyMj1FC5d_8Q0FNYxTN
8yAVvUm8MbSr0VC5co0wKVBPUxmZHRB1A1_Firhd5TBSoDgThLckwNlwH6c-
SQ0yLqtnt0Y4Htyj1-F5vOC4SmnGIKEFOv85VnPQqCRUjs5jir-
```

Enable auto-focus

DECODED HEADER

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "rxfp02j1aNRX1K1vHwKnvWT1rgs8wCbXyDCdkYCjkE"
}
```

DECODED PAYLOAD

```
{
  "exp": 1762557189,
  "iat": 1762556889,
  "jti": "onrtro:7ceb6eef-64eb-26eb-1f24-b0dc4a2fbee8",
  "iss": "http://localhost:8090/realms/myrealm",
  "aud": "account",
  "sub": "148ba672-19b3-4820-b00e-915fc4820e3c",
  "typ": "Bearer",
  "azp": "myclient",
  "sid": "30fdaca8-dbd1-b07c-405b-c9fe9bd81ed1",
  "default": "Debugger",
  "realm": "Introduction",
  "libraries": "Libraries",
  "ask": "Ask"
}
```

```

    "offline_access",
    "uma_authorization"
  ],
  "resource_access": {
    "myclient": {
      "roles": [
        "ADMIN",
        "USER"
      ],
      "account": {
        "roles": [
          "manage-account",
          "manage-account-links",
          "view-profile"
        ]
      }
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    },
    "scope": "openid email profile",
    "email_verified": true,
    "name": "frank brown",
    "preferred_username": "frank",
    "given_name": "frank",
    "family_name": "brown",
    "email": "frank@gmail.com"
  }
}
```

Notice that the token contains the **roles** for user **frank**.

Now check the token for user **john**

```
  "resource_access": {
    "myclient": {
      "roles": [
        "USER"
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid email profile",
  "email_verified": true,
  "name": "john doe",
  "preferred_username": "john",
  "given_name": "john",
  "family_name": "doe",
  "email": "john@gmail.com"
```

Our KeyCloak authorization server is now working.

Now you are ready to run the given code for this lecture. Run the given applications and check if they work correctly.

Part 2

Now change the in-memory users so that we have 3 users:

One with the role customer, one with the roles customer and employee, and one with the roles customer, employee and manager.

Then create 3 microservices A, B and C.

C contains salary data

B contains employee contact data (phone)

A is our actual company service that is used by customers, employees and managers

In A you can call productdata that is accessible by all customers, employees and managers

In A you can call employee contact data that is accessible only by employees and managers. A will call B to get the actual employee contact data.

In A you can call salary data that is accessible only by managers. A will call C to get the actual salary data.

Check that everyone can call product data

Check that only employees and managers can call employee data

Check that only managers can call salary data