

Institute of Computer Technology
B. Tech Computer Science and Engineering

Sub: Identity and Access Management (2CSE507)

Name - Nisarg Prajapati

Branch - Cyber Security

Enrollment no. – 23162171019

Semester - 5

Class - A

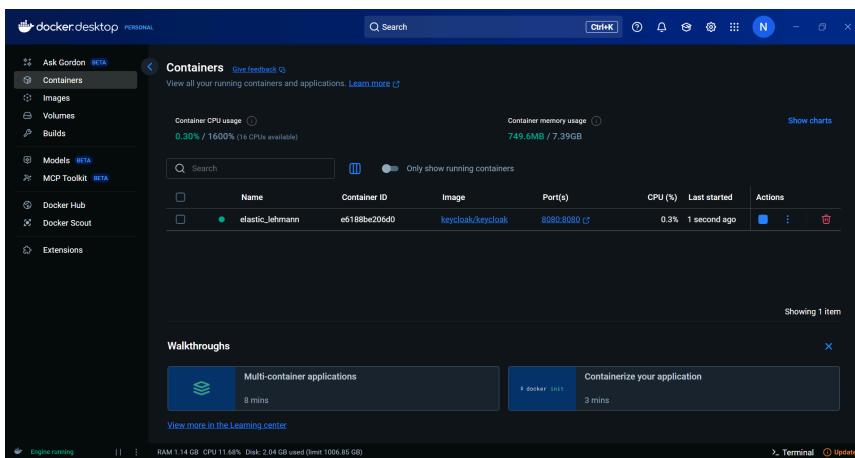
Batch – 52

PRACTICAL NO:- 7

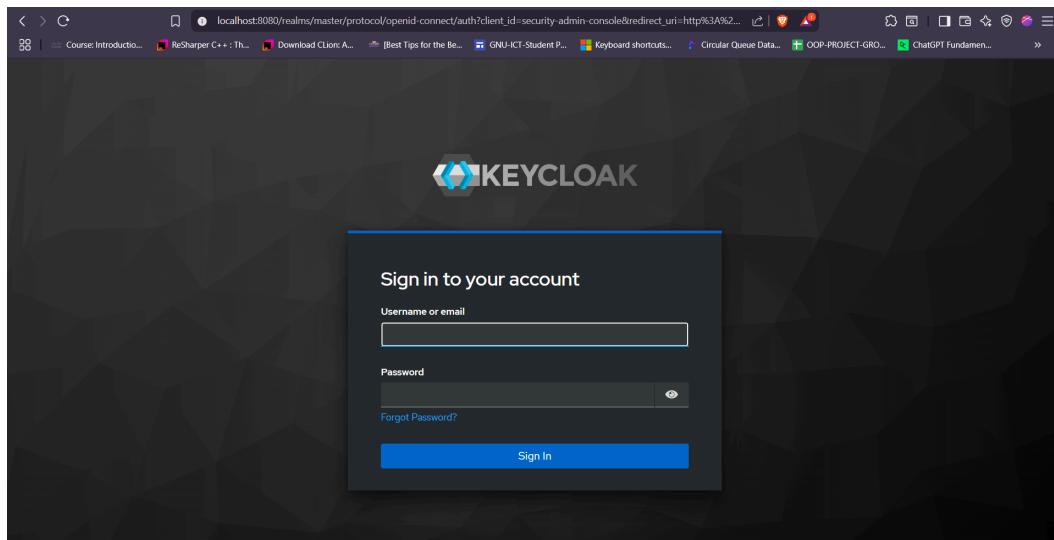
Aim : Demonstration of SSO in Keycloak

Starting the Container:

- Open your Docker Desktop.
- In the containers tab, click the start button of the container you want to start.



- Now open your browser and go to '<http://localhost:8080/>'.



Code:

- App1:

- ❖ index.js

```

import express from "express";
import session from "express-session";
import { client, code_verifier, code_challenge } from
"./auth.js";

const PORT = 3001;
const app = express();

app.use(
  session({
    secret: "app1_secret",
    resave: false,
    saveUninitialized: true,
  })
);

// Home route
app.get("/", async (req, res) => {
  if (!req.session.tokenSet) {
    return res.send(`<a href="/login">Login with
Keycloak</a>`);
  }
  res.send(`

<h2>Welcome to App1!</h2>
<p>User: ${req.session.user?.preferred_username}</p>
<a href="/logout">Logout</a>
<br><a href="http://localhost:3002/">Go to App2</a>
`);

});

// Login route
app.get("/login", (req, res) => {
  req.session.code_verifier = code_verifier;
}
)
  
```

```

const authUrl = client.authorizationUrl({
  scope: "openid profile email",
  code_challenge,
  code_challenge_method: "S256",
});
res.redirect(authUrl);
});

// Callback route
app.get("/callback", async (req, res) => {
  try {
    const params = client.callbackParams(req);
    const tokenSet = await client.callback(
      "http://localhost:3001/callback",
      params,
      { code_verifier: req.session.code_verifier }
    );
    req.session.tokenSet = tokenSet;

    const userinfo = await
client.userInfo(tokenSet.access_token);
    req.session.user = userinfo;

    res.redirect("/");
  } catch (err) {
    console.error(err);
    res.status(500).send("Login failed");
  }
});

// Logout route
app.get("/logout", (req, res) => {
  const idToken = req.session.tokenSet?.id_token;
  req.session.destroy(() => {
    let logoutUrl =
`http://localhost:8080/realm/demo-realm/protocol/openid-conne
ct/logout`;
    if (idToken) {
      logoutUrl +=
`?id_token_hint=${idToken}&post_logout_redirect_uri=http://loc
`
  
```

```
alhost:3001/`;
    }
    res.redirect(logoutUrl);
});
};

app.listen(PORT, () =>
  console.log(` App1 running on http://localhost:${PORT}`)
);

```

- auth.js

```
import { createClient } from "../oidc_config/oidc-config.js";

export const { client, code_verifier, code_challenge } =
createClient(
  "app-a",
  "http://localhost:3001/callback"
);
```

- App2

- ❖ index.js

```

import express from "express";
import session from "express-session";
import { client, code_verifier, code_challenge } from
"./auth.js";
import axios from "axios";

const PORT = 3002;
const app = express();

app.use(
  session({
    secret: "app2_secret",
    resave: false,
    saveUninitialized: true,
  })
);

// Home route
app.get("/", async (req, res) => {
  if (!req.session.tokenSet) {
    // Attempt SSO: check if user is logged in Keycloak using
    // iframe-like request
    return res.send(`<a href="/login">Login with
Keycloak</a>`);
  }
  res.send(`

    <h2>Welcome to App2!</h2>
    <p>User: ${req.session.user?.preferred_username}</p>
    <a href="/logout">Logout</a>
    <br><a href="http://localhost:3001/">Go to App1</a>
  `);
});

// Login route
app.get("/login", (req, res) => {

```

```

req.session.code_verifier = code_verifier;
const authUrl = client.authorizationUrl({
  scope: "openid profile email",
  code_challenge,
  code_challenge_method: "S256",
});
res.redirect(authUrl);
});

// Callback route
app.get("/callback", async (req, res) => {
  try {
    const params = client.callbackParams(req);
    const tokenSet = await client.callback(
      "http://localhost:3002/callback",
      params,
      { code_verifier: req.session.code_verifier }
    );
    req.session.tokenSet = tokenSet;

    const userinfo = await
client.userInfo(tokenSet.access_token);
    req.session.user = userinfo;

    res.redirect("/");
  } catch (err) {
    console.error(err);
    res.status(500).send("Login failed");
  }
});

// Logout route
app.get("/logout", (req, res) => {
  const idToken = req.session.tokenSet?.id_token;
  req.session.destroy(() => {
    let logoutUrl =
`http://localhost:8080/realm/demo-realm/protocol/openid-connect/logout`;
    if (idToken) {
      logoutUrl +=

```

```

`?id_token_hint=${idToken}&post_logout_redirect_uri=http://localhost:3002/`;
}

res.redirect(logoutUrl);
});

);

app.listen(PORT, () =>
  console.log(` App2 running on http://localhost:${PORT}`)
);

```

❖ auth.js

```

import { createClient } from "../../oidc_config/oidc-config.js";

export const { client, code_verifier, code_challenge } =
createClient(
  "app-b",
  "http://localhost:3002/callback"
);

```

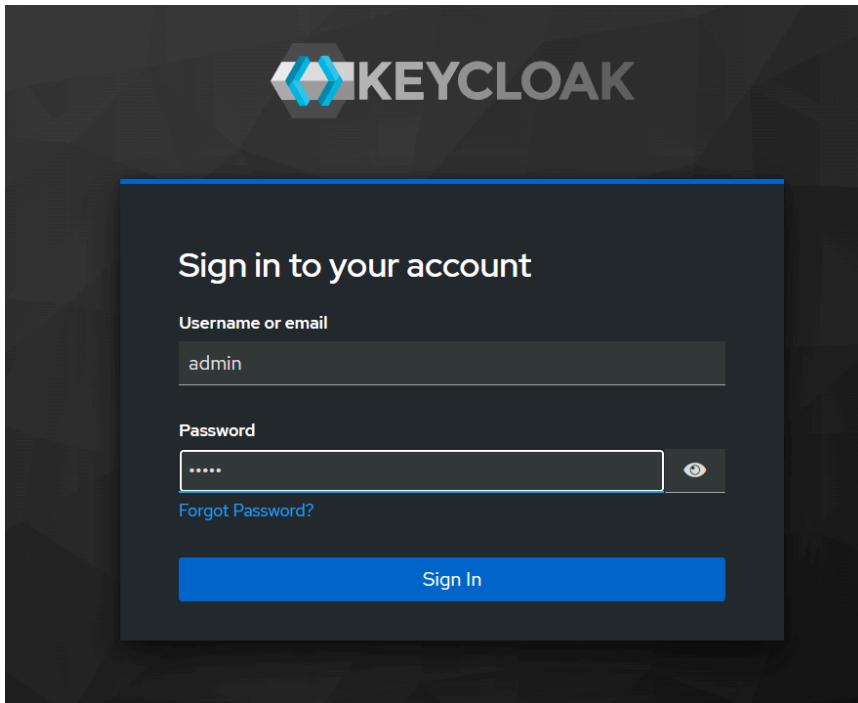
- my-realm.json

```
{
  "realm": "demo-realm",
  "enabled": true,
  "clients": [
    {
      "clientId": "app-a",
      "publicClient": true,
      "protocol": "openid-connect",
      "redirectUris": ["http://localhost:3000/callback"],
      "webOrigins": ["http://localhost:3000"],
      "standardFlowEnabled": true,
      "implicitFlowEnabled": false,
      "directAccessGrantsEnabled": true
    }
  ]
}
```

```
        },
        {
            "clientId": "app-b",
            "publicClient": true,
            "protocol": "openid-connect",
            "redirectUris": ["http://localhost:3001/callback"],
            "webOrigins": ["http://localhost:3001"],
            "standardFlowEnabled": true,
            "implicitFlowEnabled": false,
            "directAccessGrantsEnabled": true
        }
    ],
    "users": [
        {
            "username": "testuser",
            "enabled": true,
            "emailVerified": true,
            "firstName": "Test",
            "lastName": "User",
            "email": "testuser@example.com",
            "credentials": [
                {
                    "type": "password",
                    "value": "Test@123",
                    "temporary": false
                }
            ]
        }
    ]
}
```

Creating a new Realm:

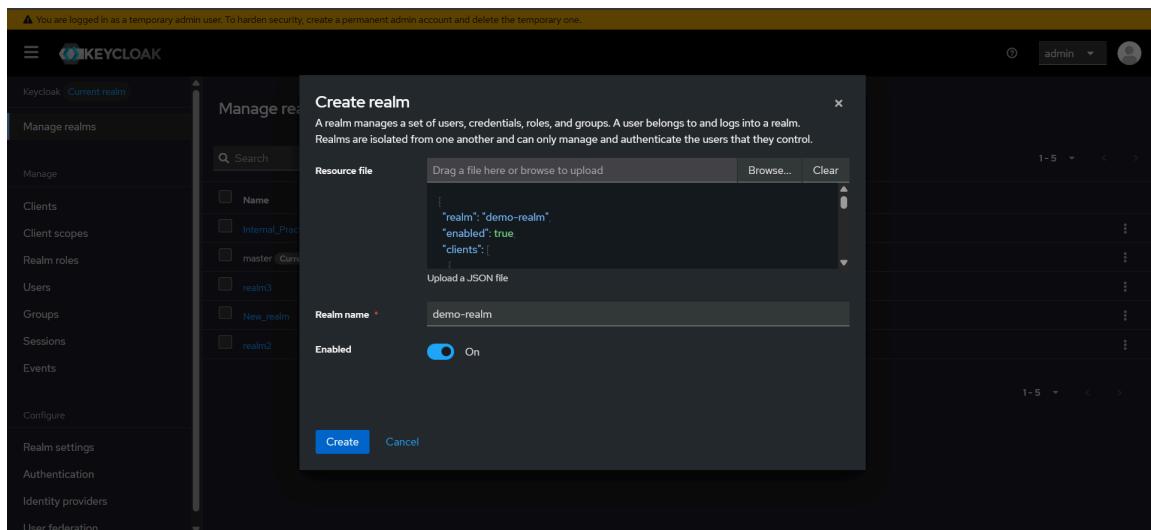
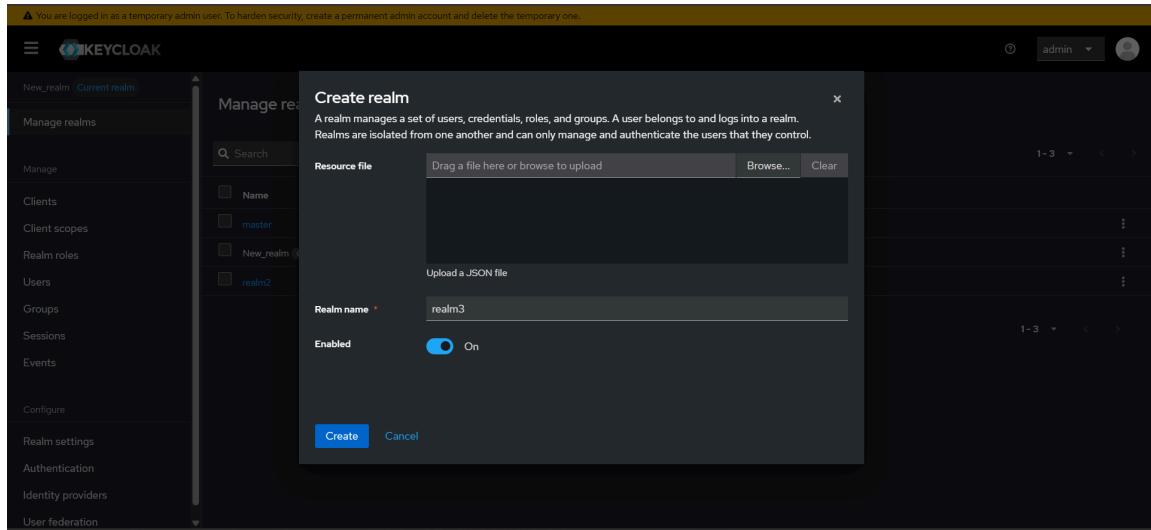
- Login as admin.



- Go to the 'Manage Realm' tab and click on 'Create realm'.

A screenshot of the Keycloak management interface. The left sidebar has a "Manage realms" section selected. The main area is titled "Manage realms" and shows a table of existing realms. The table columns are "Name" and "Display name". The rows listed are: Internal_Practical (Display name -), master (Current realm) (Display name Keycloak), realm3 (Display name -), New_realm (Display name -), and realm2 (Display name -). A "Create realm" button is visible at the top of the table area. A yellow banner at the top of the page says: "⚠ You are logged in as a temporary admin user. To harden security, create a permanent admin account and delete the temporary one." The top right corner shows the user is logged in as "admin".

- Enter the realm name, select the ‘my-realm.json’ in ‘Resource file’ and create the realm.



- To change between realms, in ‘Manage Realm’ tab, click on the realm name.

The screenshot shows the Keycloak administration interface. The left sidebar is titled 'demo-realm Current realm' and includes sections for Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'Manage realms' and displays a table of realms. The columns are 'Name' and 'Display name'. The realms listed are: Internal_Practical (Display name -), master (Display name Keycloak), realm3 (Display name -), New_realm (Display name -), demo-realm (Display name Current realm), and realm2 (Display name -). A success message 'Realm created successfully' is visible in the top right corner.

The screenshot shows the Keycloak administration interface. The left sidebar is identical to the previous screenshot. The main content area has a large Keycloak logo icon and the text 'Welcome to demo-realm'. Below it, a message says 'If you want to leave this page and manage this realm, please click the corresponding menu items in the left navigation bar.' The URL 'localhost:8080/admin/master/console/#/demo-realm' is visible at the bottom of the page.

- Check the created user in ‘Users’ tab

The screenshot shows the Keycloak admin interface with the 'demo-realm' selected. The left sidebar has 'Manage' expanded, with 'Users' selected. The main area is titled 'Users' and shows a table with one row for 'testuser'. The columns are 'Username' (testuser), 'Email' (testuser@example.com), 'Last name' (User), and 'First name' (Test). There are buttons for 'Add user' and 'Delete user' at the top, and a 'Refresh' button.

Username	Email	Last name	First name
testuser	testuser@example.com	User	Test

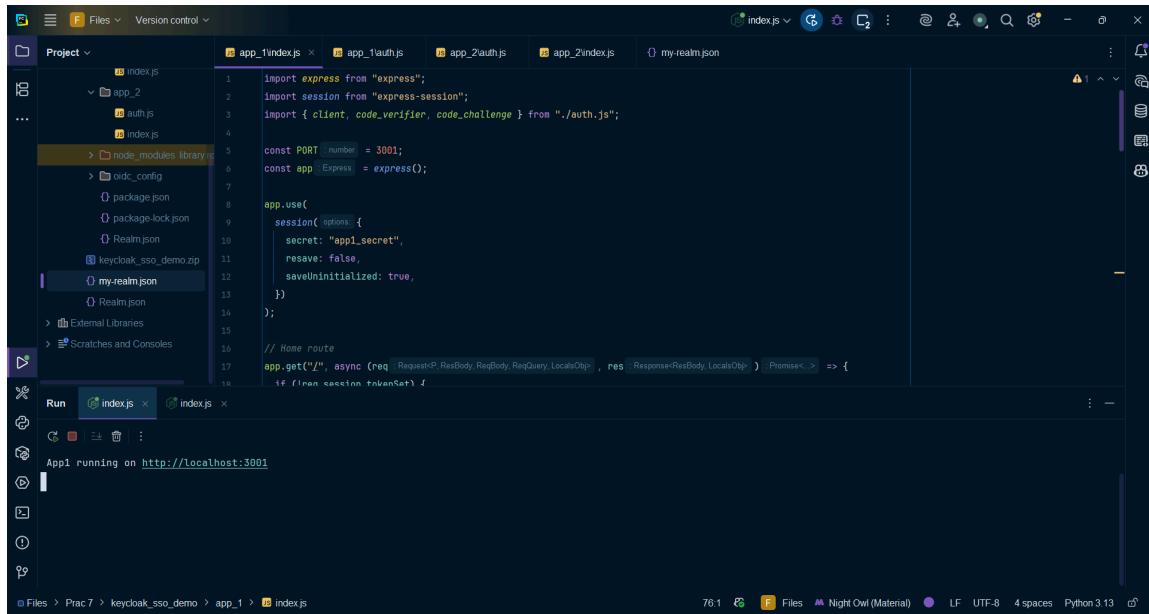
- Check the created clients in ‘Clients’ tab

The screenshot shows the Keycloak admin interface with the 'demo-realm' selected. The left sidebar has 'Manage' expanded, with 'Clients' selected. The main area is titled 'Clients' and shows a table with eight rows of client information. The columns are 'Client ID' (account, account-console, admin-cli, app-a, app-b, broker, realm-management, security-admin-console), 'Name' (Account, Account Console, Admin CLI, App A, App B, Broker, Realm Management, Security Admin Console), 'Type' (OpenID Connect, OpenID Connect), 'Description' (–, –, –, –, –, –, –, –), and 'Home URL' (http://localhost:8080/realm/demo-realm/account/, http://localhost:8080/realm/demo-realm/account/, –, –, –, –, –, http://localhost:8080/admin/demo-realm/console/). There are buttons for 'Create client' and 'Import client' at the top, and a 'Refresh' button.

Client ID	Name	Type	Description	Home URL
account	Account	OpenID Connect	–	http://localhost:8080/realm/demo-realm/account/
account-console	Account Console	OpenID Connect	–	http://localhost:8080/realm/demo-realm/account/
admin-cli	Admin CLI	OpenID Connect	–	–
app-a	App A	OpenID Connect	–	–
app-b	App B	OpenID Connect	–	–
broker	Broker	OpenID Connect	–	–
realm-management	Realm Management	OpenID Connect	–	–
security-admin-console	Security Admin Console	OpenID Connect	–	http://localhost:8080/admin/demo-realm/console/

Starting the login apps:

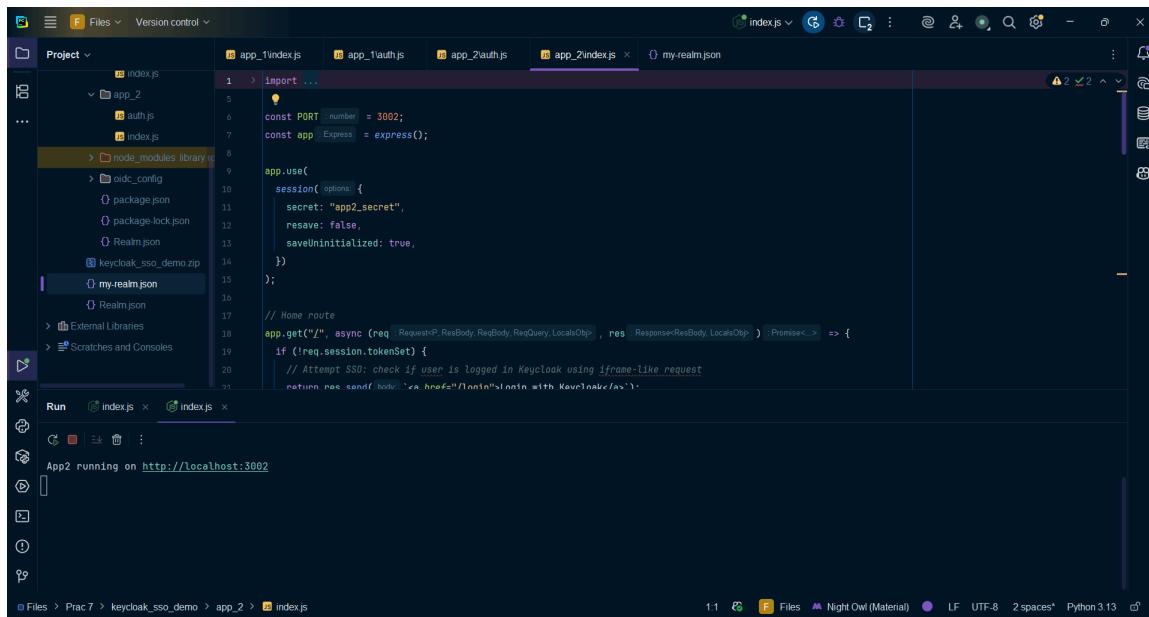
- Run the index.js files from both app1 and app2.



```

1 import express from "express";
2 import session from "express-session";
3 import { client_code_verifier, code_challenge } from "./auth.js";
4
5 const PORT = 3001;
6 const app = express();
7
8 app.use(
9   session({
10     secret: "app1_secret",
11     resave: false,
12     saveUninitialized: true,
13   })
14 );
15
16 // Home route
17 app.get("/", async (req, res) => {
18   if (!req.session.tokenSet) {
19     // Attempt SSO: check if user is logged in Keycloak using iframe-like request
20     return res.send(`<a href="#">Login</a> with Keycloak!`);
21   }
22 });
23
24 app.listen(PORT, () => {
25   console.log(`App1 running on http://localhost:${PORT}`);
26 });

```



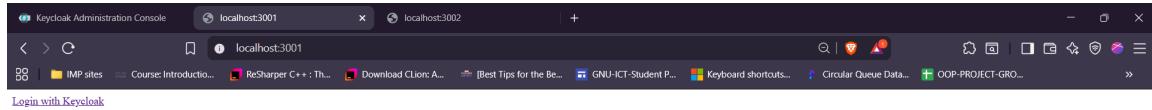
```

1 > import ...
5 const PORT = 3002;
6 const app = express();
7
8 app.use(
9   session({
10     secret: "app2_secret",
11     resave: false,
12     saveUninitialized: true,
13   })
14 );
15
16 // Home route
17 app.get("/", async (req, res) => {
18   if (!req.session.tokenSet) {
19     // Attempt SSO: check if user is logged in Keycloak using iframe-like request
20     return res.send(`<a href="#">Login</a> with Keycloak!`);
21   }
22 });
23
24 app.listen(PORT, () => {
25   console.log(`App2 running on http://localhost:${PORT}`);
26 });

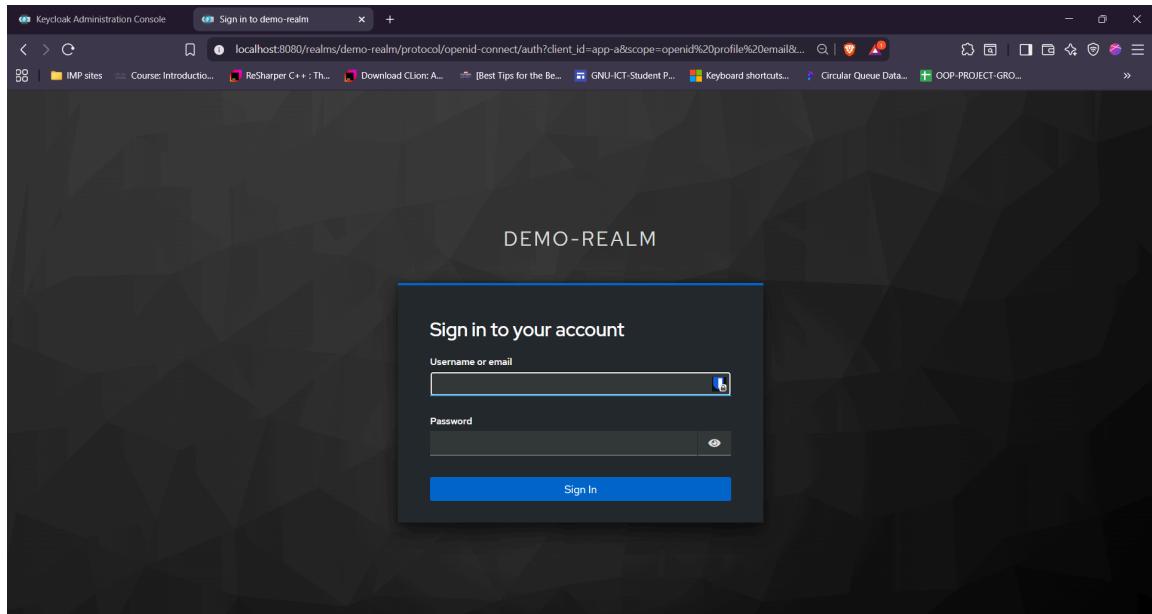
```

Checking SSO:

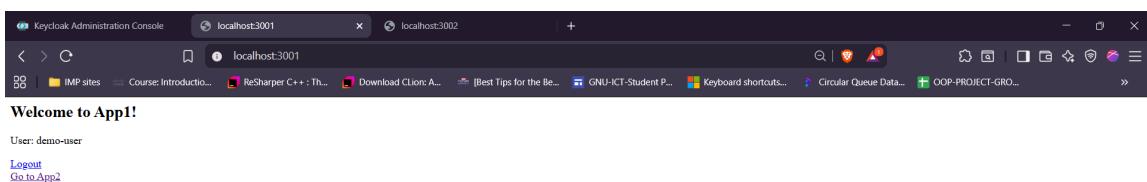
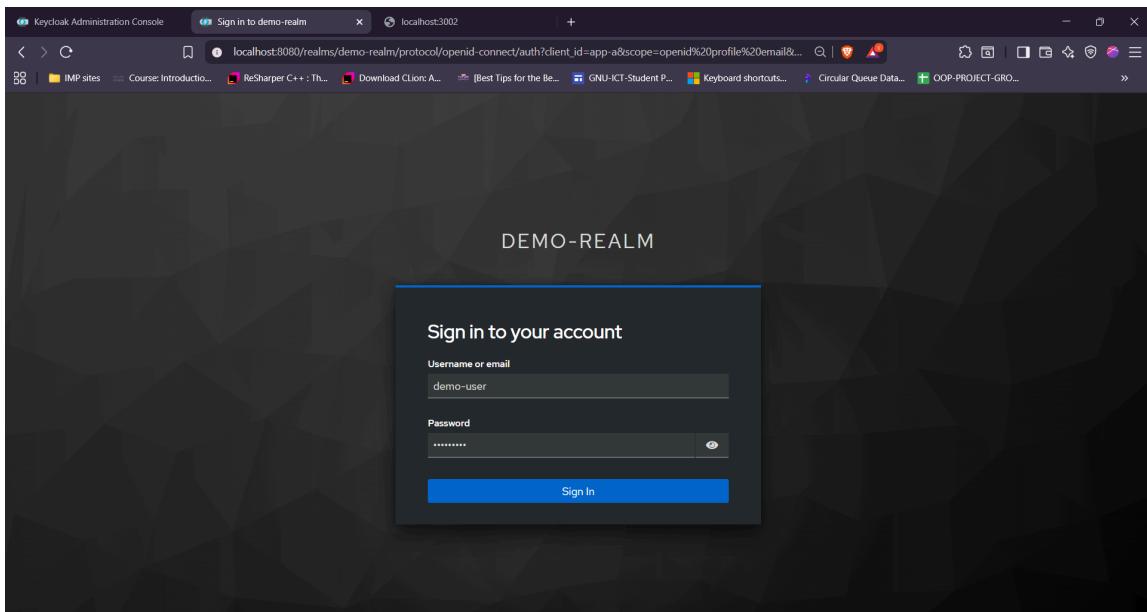
- Open ‘<http://localhost:3001/>’ in the browser.



- Click on ‘Login with Keycloak’.



- Enter username and password and click on ‘Sign in’.



- Click on ‘Go to App2’.



- Click on ‘Login with Keyclock’.

