1. Create a folder and name it: **Security\_Projects**. This folder will contain the security assignment and is just a container.
2. Open your terminal and run **create-react-app** inside “the Security\_Projects” folder. Name your react project assignment\_security.

*> create-react-app assignment\_security*

1. Navigate to your project folder and open it on VSCode.
2. Open your **src** folder and create a new folder inside it: **components**. This folder will include two components: **Header.js** and **Home.js. “**Home.js” is just the header of the application with just a title. “Home.js” is the main component and will contain the primary contents of the app.
3. Open **App.css** and delete all code you see inside this file. This code won’t be used for the application. Instead, write the following code to start with:

\* {

padding: 0;

margin: 0;

box-sizing: border-box;

}

This will give us a basic CSS reset and will help us a little for future CSS codes.

1. Add another folder next to the components folder and name it: **resources.** Create a **CSS**-folder inside the resources folder. This will contain all CSS files that will be linked with the corresponding component.

(Header.css 🡪 Header.js)

(Home.css 🡪 Home.js)

1. Now go to **App.js** and delete everything between:

<div *className*="App">

</div>

Now add your two components inside your **div** as shown below:

<div *className*="App">

<Header />

<Home />

</div>

1. Import your two components inside the **App.js** file, before your App () function

import Header from "./components/Header";

import Home from "./components/Home";

function App() {

return (

<div *className*="App">

<Header />

<Home />

</div>

);

}

1. Open your **Header.js** and add the following code to create a simple header for the app.

import React from "react";

import "./../resources/css/Header.css";

const Header = () => {

return (

<header>

<h1>Assignment Security - React, IdentityServer4</h1>

</header>

);

};

export default Header;

1. Now let’s open the **Header.css** file to change the layout of our header:

header {

background-color: darkgrey;

padding: 50px 0;

text-align: center;

}

This code will suffice.

1. Now let’s open up our main component: **Home.js.**

To start with, create a class called **Home:**

import React, { Component } from "react";

import JSONTree from "react-json-tree";

import "./../resources/css/Home.css";

class Home extends Component {

render() {

return (

<div *className*="home">

<div *className*="buttons">

<button *onClick*={this.login} *className*={"btn-login"}>

Login

</button>

<button *onClick*={this.api} *className*={"btn-api"}>

Call API

</button>

<button *onClick*={this.logout} *className*={"btn-logout"}>

Logout

</button>

</div>

<div *className*={"json-headers"}>

<h1>User Data</h1>

<h1>API Data</h1>

</div>

<div *className*={"trees"}>

<JSONTree *data*={this.state.user} *theme*={"bright"} />

<JSONTree *data*={this.state.api} *theme*={"bright"} />

</div>

</div>

);

}

}

Then, add this JSX-code in your render method.

* 3 buttons:
  + Login button
  + Button to call the API
  + Logout button
* 2 headers (h1)
* 2 JSONTrees to display json data easily
  + User data
  + Data retrieved from API

Don’t forget to add the **react-json-tree** library to your project.

*> npm i react-json-tree*

!! Be sure to be inside your project folder when running this in your terminal !!

1. Add a **constructor** in your Home class.

class Home extends Component {

constructor() {

*super*();

this.state = {

api: {},

user: {}

};

this.login = this.login.bind(this);

this.logout = this.logout.bind(this);

this.api = this.api.bind(this);

}

Inside your constructor:

* Call **super()**
* Initialize your **state**.
  + Api object = data object about api
  + User object = data object about user of the application.
* **Bind** the **methods** that will be used when pressed on buttons made earlier in the render method.

1. Create **login**, **logout** and **api** methods in Home class just below the constructor.

login() {

}

logout() {

}

api() {

}

These methods will be called when pressed on the buttons on our homepage:

Login -> login();

Call API -> api();

Logout -> logout();

1. Add another 2 methods **callAPI() and getUser()** just below the **api()** method.

callAPI(token) {

}

getUser() {

}

The **callAPI** method will be used for the actual call to the API. For this to happen, we need an access token retrieved when logging in into the server (IdentityServer4). Add a parameter (token) in for callAPI for this **access\_token**.

The **getUser** method will be used to retrieve (meta)data about the user.

1. Create a folder **services** and add **AuthService.js** to this folder.

AuthService will be our client that communicates with the server.

import { UserManager } from "oidc-client";

export class AuthService {

constructor() {

this.userManager = new UserManager({

authority: "https://demo.identityserver.io",

client\_id: "spa",

redirect\_uri: "http://localhost:3000/signin-callback.html",

post\_logout\_redirect\_uri: "http://localhost:3000",

response\_type: "code",

scope: "openid profile email api"

});

}

getUser() {

return this.userManager.getUser();

}

login() {

return this.userManager.signinRedirect();

}

logout() {

return this.userManager.signoutRedirect();

}

}

Inside the **constructor**, you create a new **UserManager** **object** with several key: value parameters as shown above.

1. Let’s go back to **Home.js** and make use of our AuthService.js

Import the AuthService.

import { AuthService } from "../services/AuthService";

Create new **AuthService** object in the **constructor**.

constructor() {

*. . .*

this.authService = new AuthService();

. . .

this.callAPI = this.callAPI.bind(this);

}

login() {

this.authService.login();

}

logout() {

this.authService.logout();

}

api() {

this.authService.getUser().then(*user* => {

if (user && user.access\_token) {

return this.callAPI(user.access\_token);

} else {

alert("You are not logged in");

}

});

}

callAPI(token) {

const headers = {

Accept: "application/json",

Authorization: "Bearer " + token

};

axios

.get("https://demo.identityserver.io/api/test", { headers })

.then(*data* => {

this.setState({

api: data.data

});

return data.data;

});

}

getUser() {

this.authService

.getUser()

.then(*user* => {

if (!user) {

alert("You are not logged in");

}

this.setState({

user: user

});

})

.catch(*error* => {

alert(error);

});

}

**add the axios library to your project, which will be used to retrieve data from an api.**

*> npm i axios*

import axios from "axios";

1. Add **componentDidMount()** in your **Home.js** component and call your getUser() method.

This method will be called when your render method finishes.

componentDidMount() {

this.getUser();

}

1. We still need to add a **callback html page** when signing in. Put this html page right beside your **index.html** file in your **public folder**.

<!DOCTYPE html>

<html lang="en">

<head>

<title>Assignment Security</title>

</head>

<body>

<noscript>

You need to enable JavaScript to run this app.

</noscript>

<h1>Authentication callback processing...</h1>

<script src="https://cdnjs.cloudflare.com/ajax/libs/oidc-client/1.8.2/oidc-client.js"></script>

<script>

new Oidc.UserManager({ response\_mode: "query" })

.signinRedirectCallback()

.then(function() {

window.location = "index.html";

})

.catch(function(*e*) {

console.error(e);

});

</*script*>

</*body*>

</*html*>

1. Ok, now we just need to make our app prettier, mainly our **Home** component.

**Home.css** should look like this:

.home {

width: 1500px;

margin: 0 auto;

}

.trees, .json-headers {

padding: 0 50px;

display: flex;

}

.trees > \*, .json-headers > \* {

width: 50%;

}

.buttons {

display: flex;

justify-content: center;

padding: 20px;

}

button {

padding: 10px 20px;

margin: 0 10px;

color: white;

cursor: pointer;

user-select: none;

border: none;

border-radius: 5px;

}

.btn-login {

background-color: rgb(156, 56, 56);

}

.btn-api {

background-color: rgb(35, 194, 186);

}

.btn-logout {

background-color: rgb(0, 0, 0);

}