KeyCloak

Introduction

THIS DOCUMENT COVERS

- Introduction

Cheat Sheet

This gives details on end points on the server

http://localhost:8080/auth/realms/master/.well-known/openid-configuration

React and ASP.NET Core

This example shows how to use KeyCloak to secure a React front end and a .NET Core backend

Install KeyCloak.

The first step is to install KeyCloak and add an admin user as described here in the KeyCloak documentation.

https://www.keycloak.org/docs/latest/getting_started/index.html

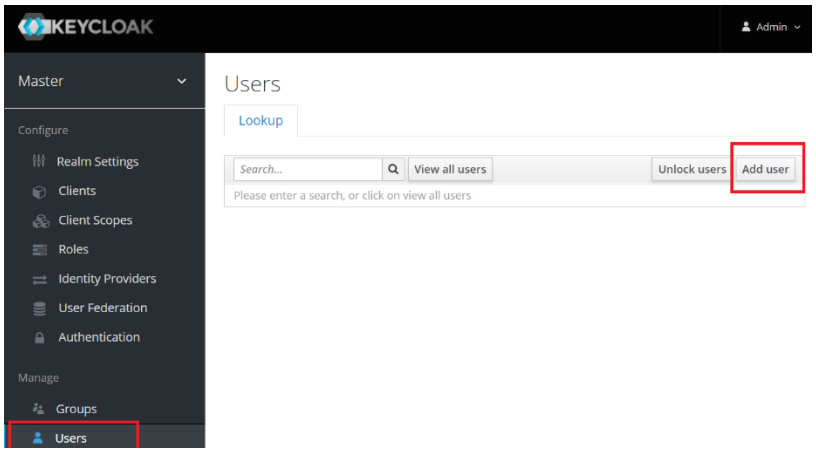
Configure KeyCloak.

For this tutorial we will just use the master realm.

ADD A USER.

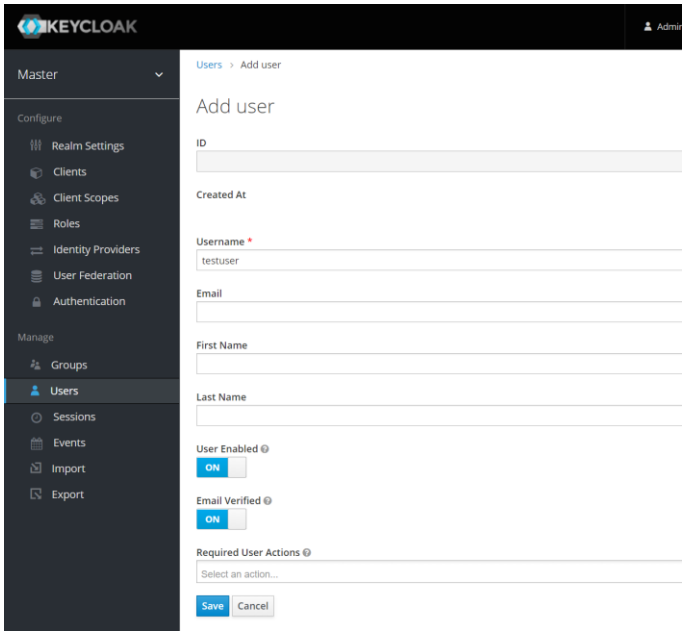
Open the Add user screen

We go to the Users tab and click add user.



Enter Name and turn on Email Verified

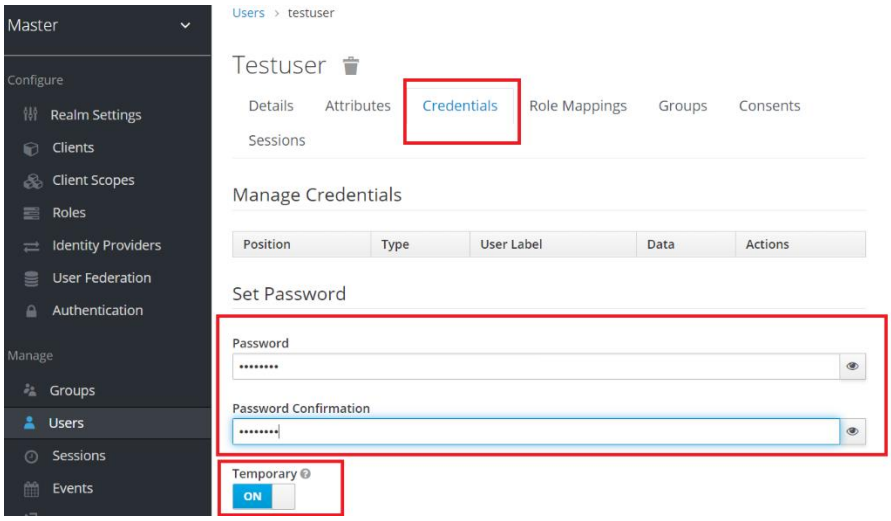
Now we enter the name as **testuser**, set **Email Verified** to "On" to indicate we do not need the user to verify the password we will set. Finally, we click save.



Now go to the credential

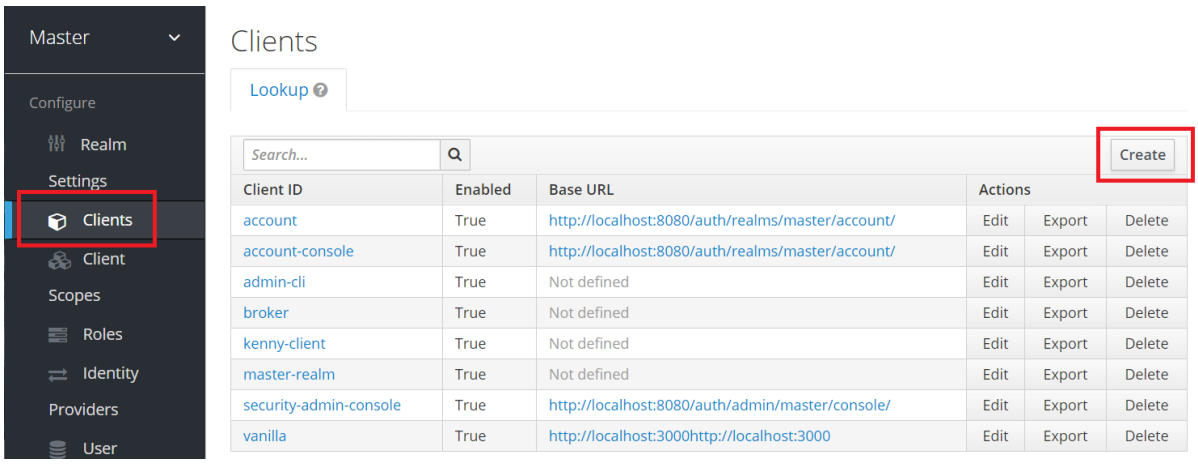
Set the password for the user

Open the Credentials tab for the user and enter the password. Set the Temporary flag to false so we do not have to update it on first logon. Enter the password as t**estuser.**



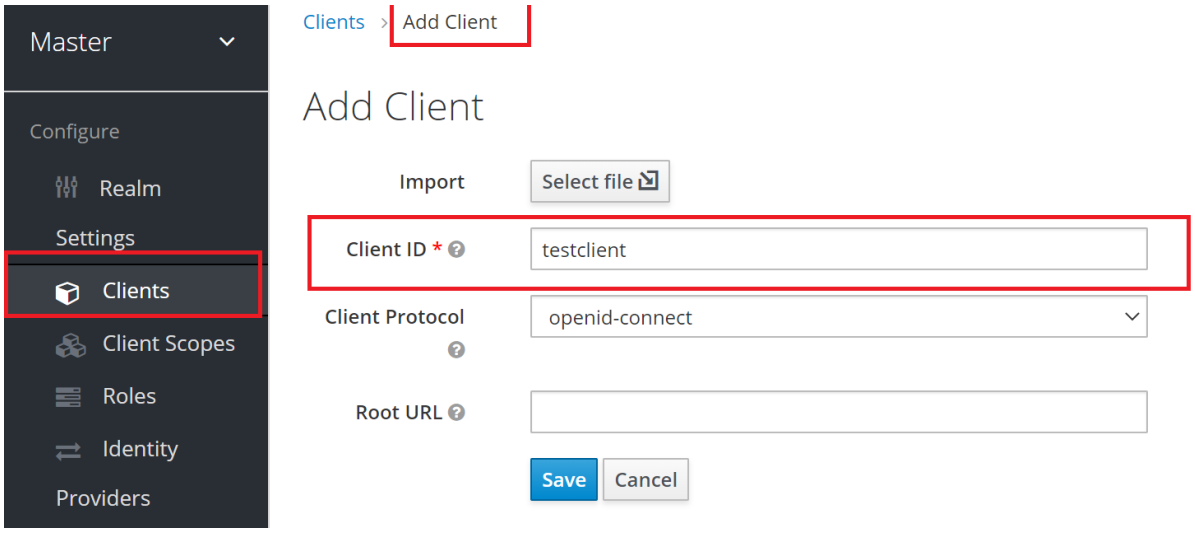
ADD A CLIENT.

Open the Add Client screen



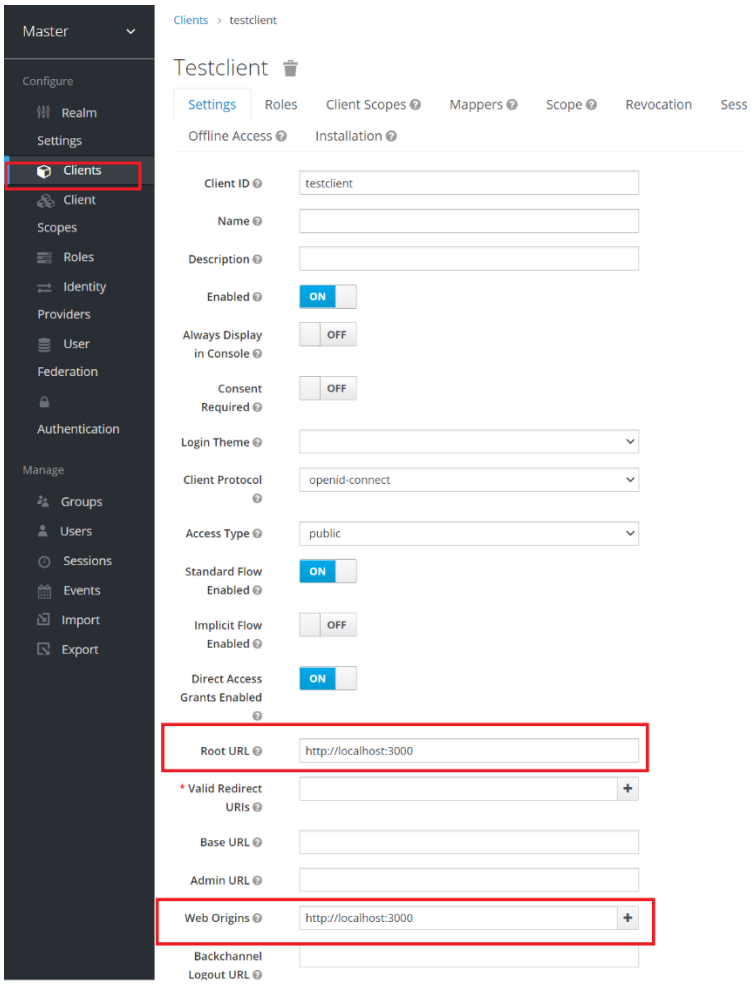
Enter the Client

Set the name as **testclient and** click save.



Configure the Client

We need to set the base URL to the URL of the React client we will add later. As for this example we will be using create-react-app we will set this to http://localhost:3000. We add the same URL to the Web Origins so we don't end with CORS errors in the React client. Once entered click save.

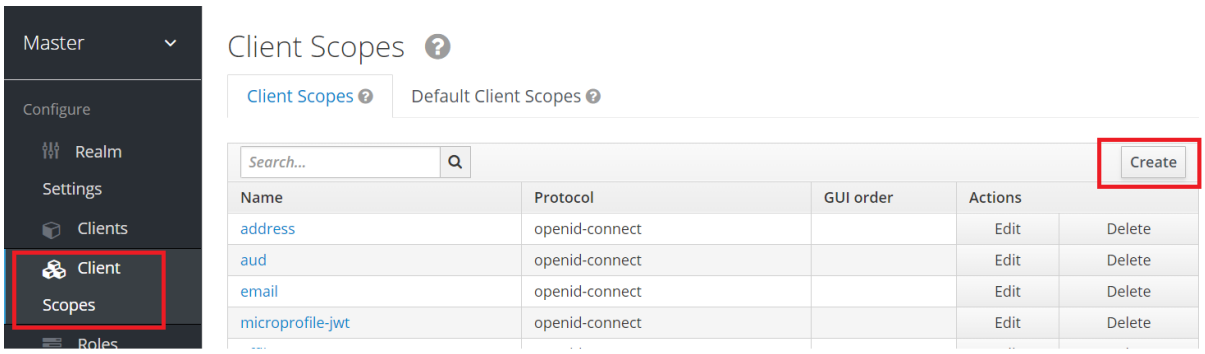


CREATE A CLIENT SCOPE AND MAPPER.

This is the price that adds the client audience in the **aud** field of a generated token. This is super important.

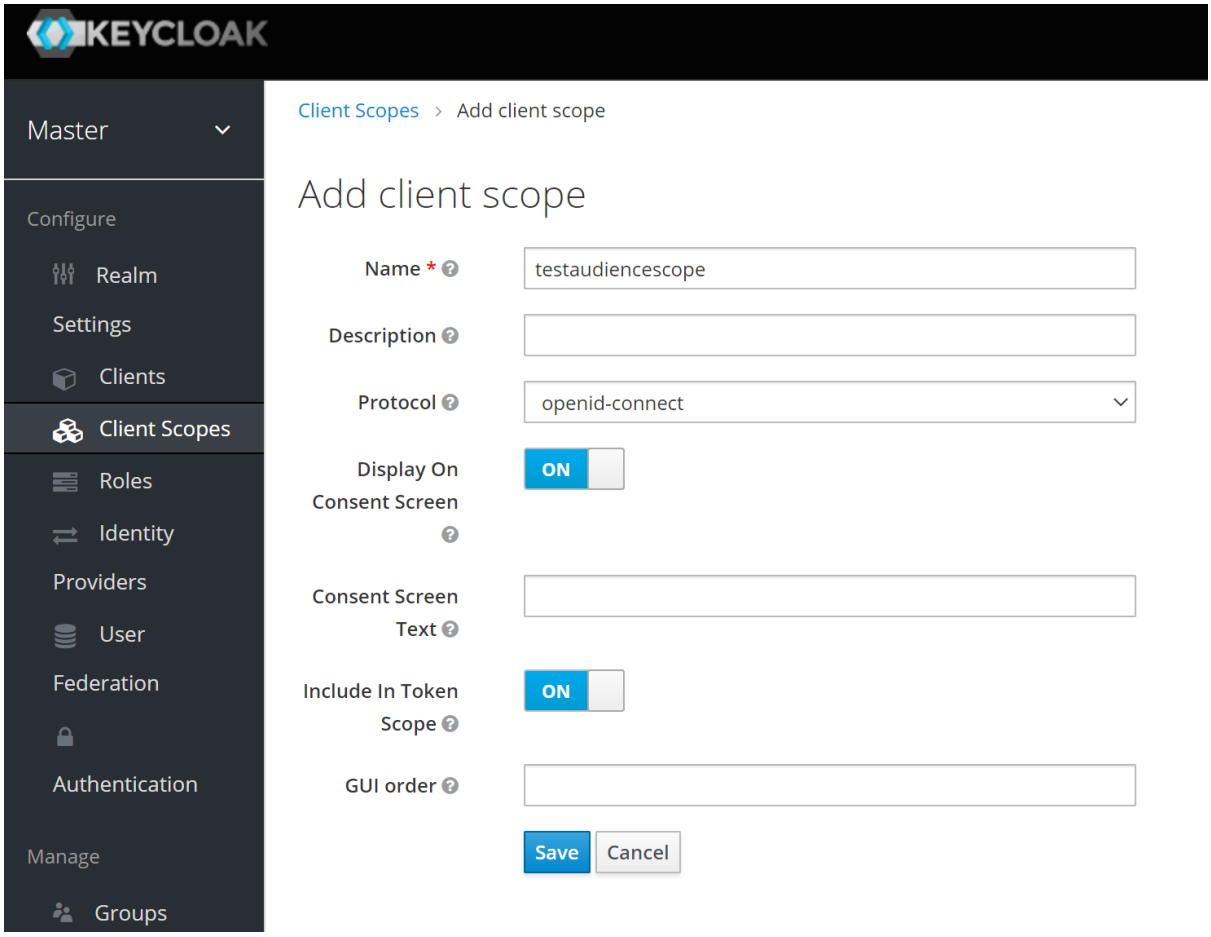
Open Create Client Scope Screen

Click Create the from the Client Scopes screen.



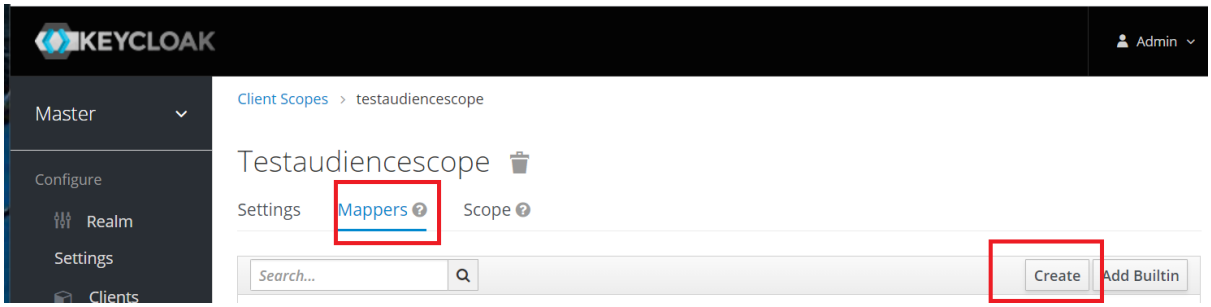
ADD A NEW CLIENT SCOPE

Add a scope with name **testaudiencescope** and click save.



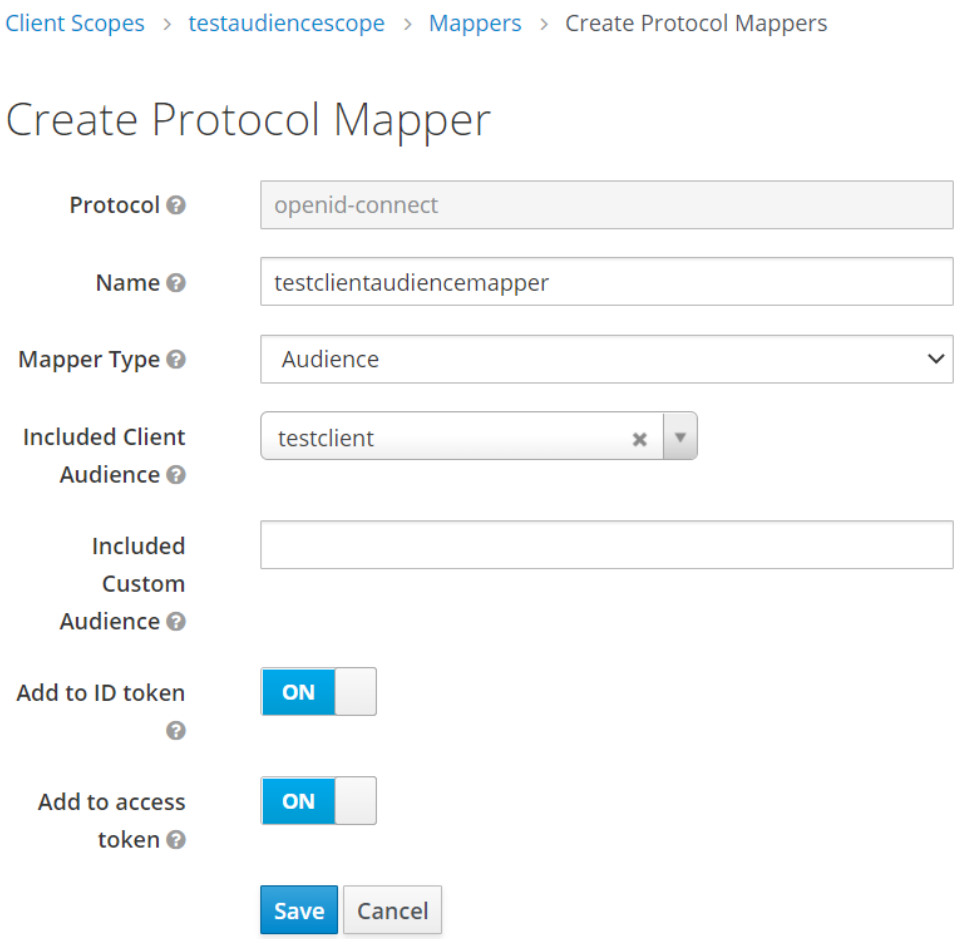
Open the Create Mapper Screen

From our new client scope move to the Mappers tab and click Create.

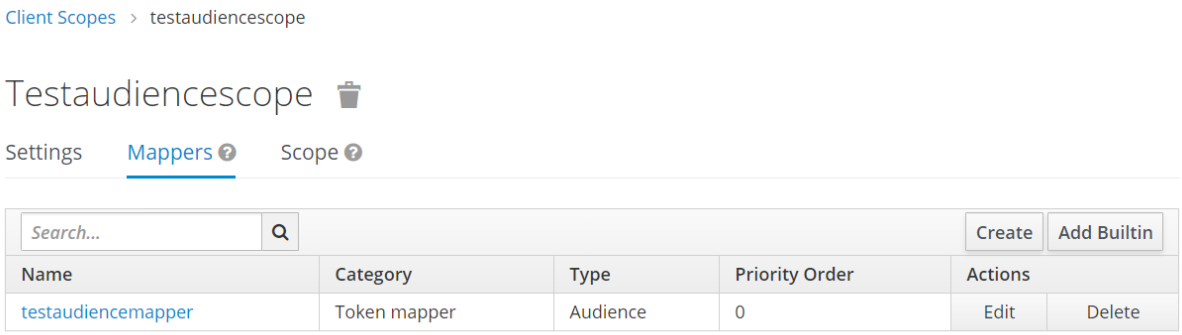


Configure the Mapper.

Configure the mapper as follows.

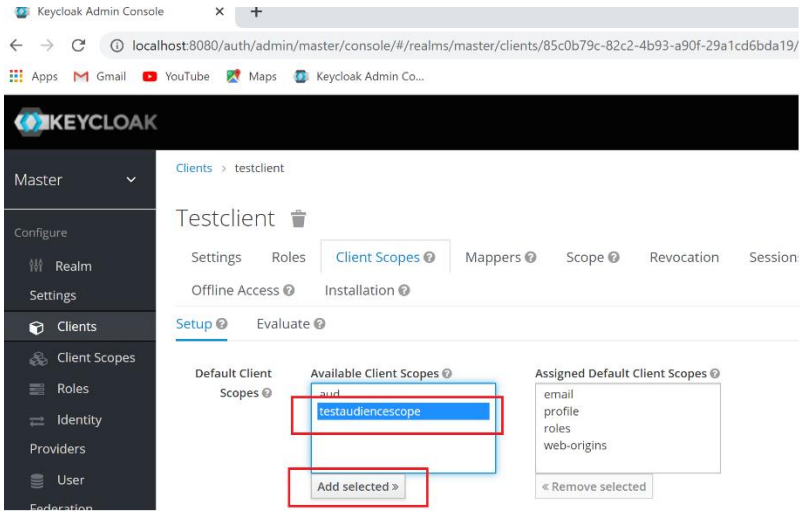


The mappers tab should look like this now.



Add Scope to client

Select the testaudiencescope and add to the Default Client Scopes



Create the Frontend react App.

Create the app

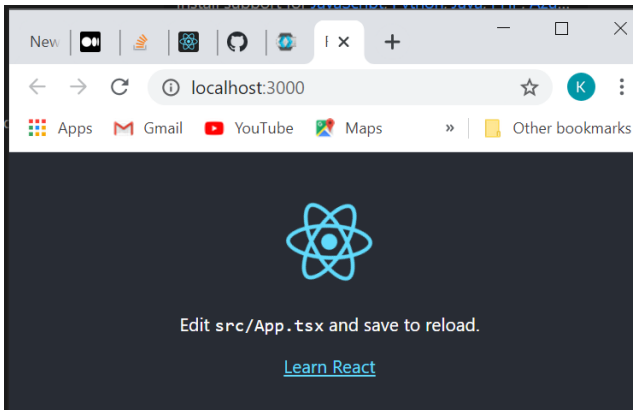
Assuming you have node and npm installed open a command prompt and enter the following command.

```
npx create-react-app ui --template typescript
```

Open visual studio in the new ui folder. Open a terminal and enter

```
Npm start
```

Make sure you can see the react app screen



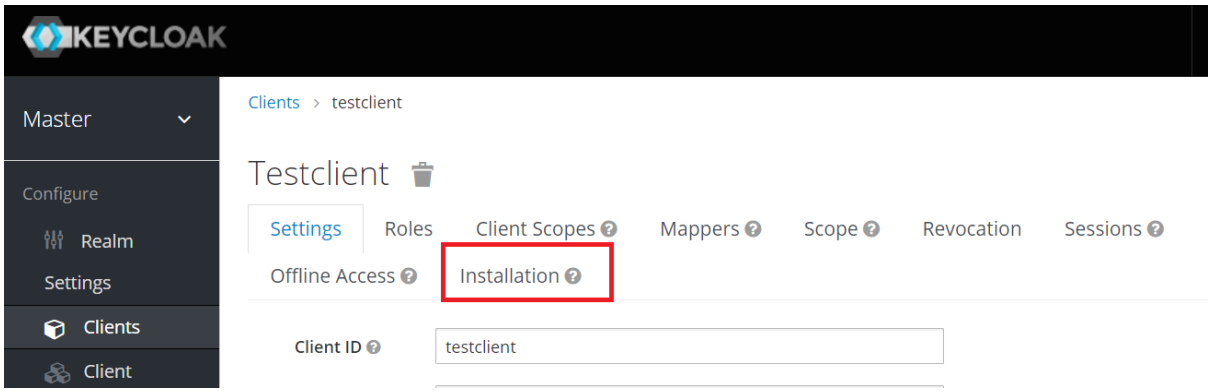
Add KeyCloak npm Package

From the command line run the command

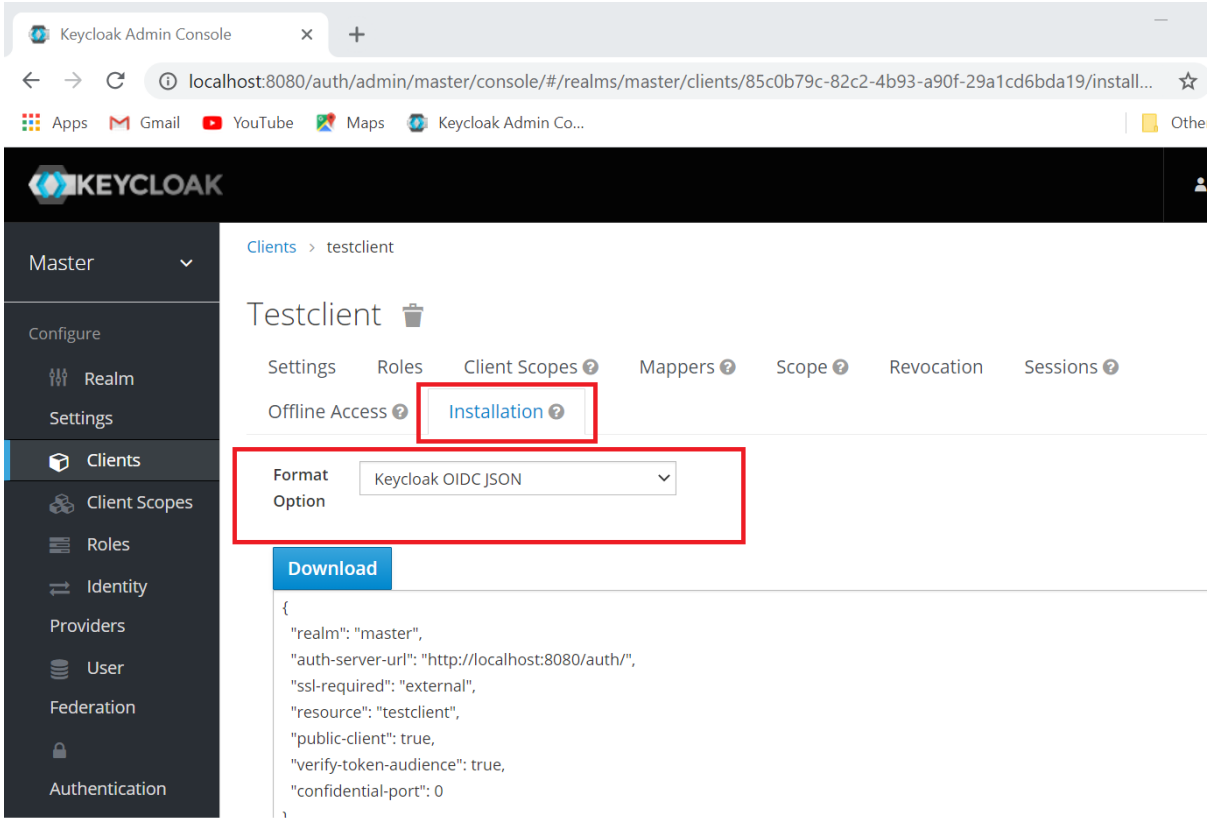
```
npm install keycloak-js
```

Add Keycloak settings

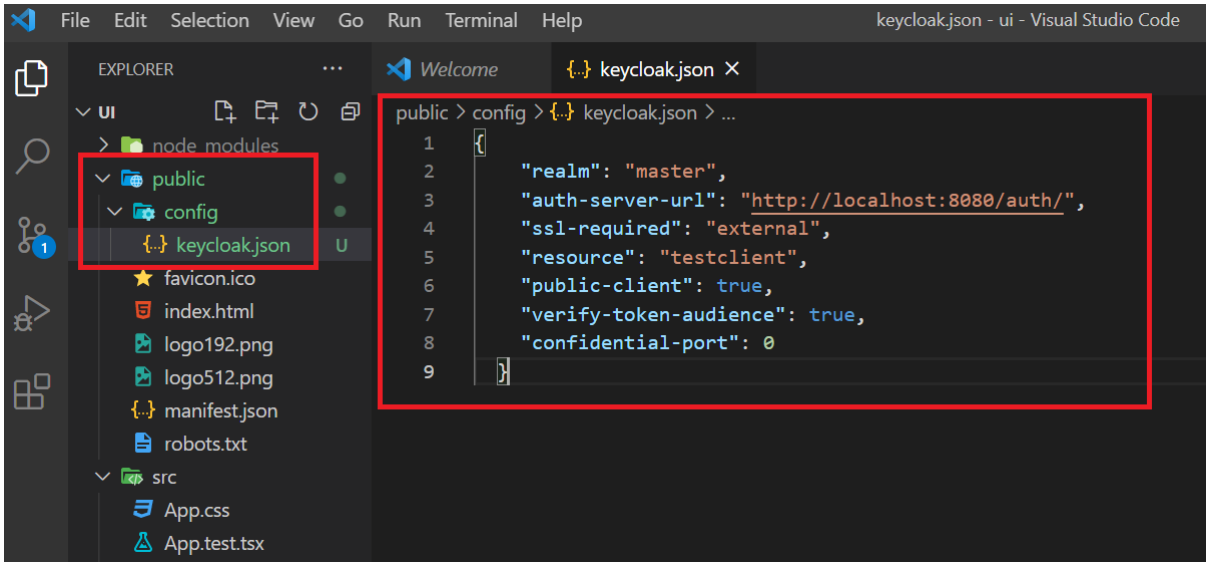
From KeyCloak admin UI go to our client that we created in the previous sections and click installation.



From installation screen select KeyCloak OIDC JSON from the Format Option drop down



Copy the JSON into a file called keycloak.json in the React app's public/config folder



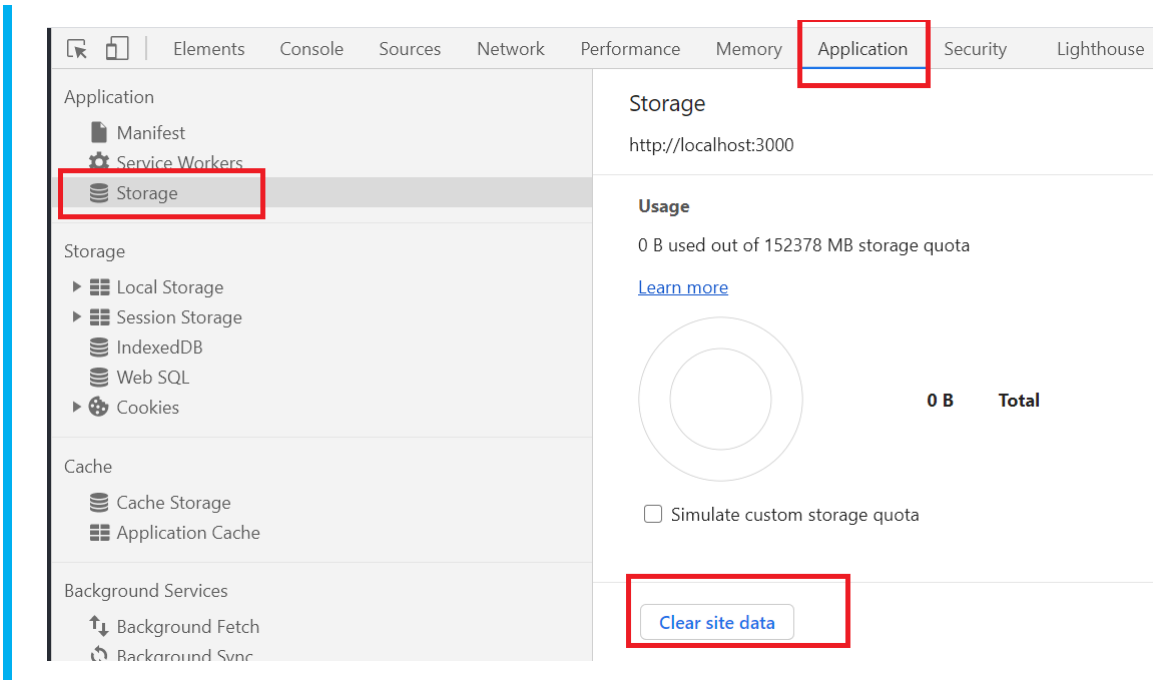
Add Logic to connect to KeyCloak

Replace index.tsx with the following code

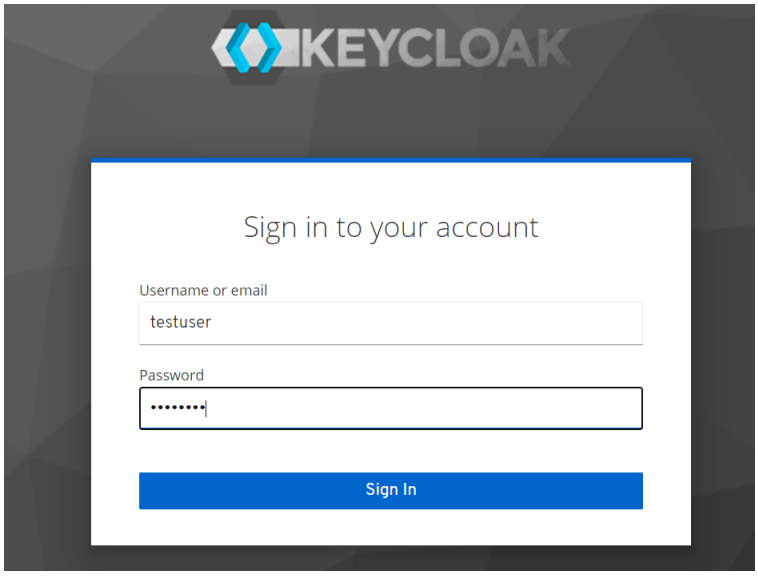
```
import React, { ReactElement } from 'react';

import ReactDOM from 'react-dom';
import './index.css';
import reportWebVitals from './reportWebVitals';
import Keycloak from 'keycloak-js';

const keycloak = Keycloak(`/config/keycloak.json?ts=${new Date().getTime()}`);

async function DoWork()
{
  await keycloak.init({onLoad:'login-
required', enableLogging:true, checkLoginIframe:false});

  await  keycloak.updateToken(120);

  const token = keycloak.token;
  const tokenParsed = keycloak.tokenParsed;

  ReactDOM.render(
            <React.StrictMode>
              <App json={tokenParsed}></App>
            </React.StrictMode>,
            document.getElementById('root')
          );
}

function App(props:any) : ReactElement
{
  return <pre>{JSON.stringify(props.json,null, 2)}</pre>
}


DoWork();
```

When we reload the app we should be asked to login to KeyCloak.

> **Clear KeyCloak Cached Data**
>
> If we make changes we often need to clear out keycloak settings. We can do this in Chrome by opening developer settings. Going to Application Tab. Selecting Storage and Clear Site Data



Enter the username of testuser and password of testuser.



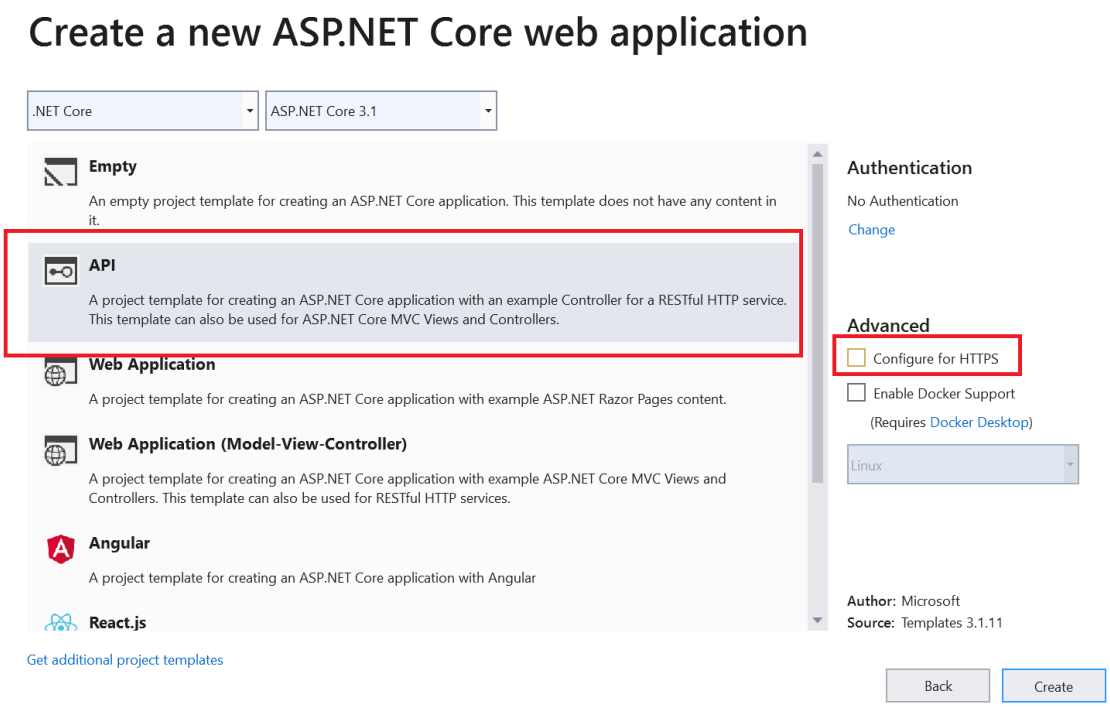
We should see a token similar to the following, Note we have the testclient in the audience. This is key to use the token from .net

```
{
  "exp": 1613042771,
  "iat": 1613042711,
  "auth_time": 1613042696,
  "jti": "a2c0dbd8-5f53-4cc8-9b23-9f7c3e633757",
  "iss": "http://localhost:8080/auth/realms/master",

    "aud": [
      "testclient",


    "master-realm",
    "account"
  ],
  "sub": "bd93aa68-622e-4907-aeb7-59967e7e1490",
  "typ": "Bearer",
  "azp": "testclient",
  "nonce": "c64e4324-f5ad-4ea0-9ba8-7942b5f4c242",
  "session_state": "badf9150-4718-4d4b-9693-565e6b3c0344",
  "acr": "0",
  "allowed-origins": [
    "http://localhost:3000"
  ],
  "realm_access": {
    "roles": [
      "create-realm",
      "offline_access",
      "admin",
      "uma_authorization"
    ]
  },
  "resource_access": {
    "master-realm": {
      "roles": [
        "view-identity-providers",
        "view-realm",
        "manage-identity-providers",
        "impersonation",
        "create-client",
        "manage-users",
        "query-realms",
        "view-authorization",
        "query-clients",
        "query-users",
        "manage-events",
        "manage-realm",
        "view-events",
        "view-users",
        "view-clients",
        "manage-authorization",
        "manage-clients",
        "query-groups"
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid profile email testaudiencescope",
  "email_verified": false,
  "preferred_username": "admin"
}
```

Create the Backend.

CREATE THE PROJECT

Open visual studio and create a new application using **ASP.NET Core Web Application**. Give the solution and project a name and then select the API template. Untick the configure HTTP checkbox



CREATE PACKAGES FOR JWT AND OPENID

Make sure the project is targetint .NET 5.0 and add the following dependencies to the csproj file.

```
<Project Sdk="Microsoft.NET.Sdk.Web">

	<PropertyGroup>
		<TargetFramework>net5.0</TargetFramework>
	</PropertyGroup>

	<ItemGroup>
		<PackageReference
Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="5.0.3" />
		<PackageReference
Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="5.0.3" />
	</ItemGroup>

</Project>
```

USE AUTHENTICATION ON ENDPOINT ACTION

Add the authorization attribute to the end point `WeatherForecastController.cs`

```
[HttpGet]
[Authorize]
public IEnumerable<WeatherForecast> Get()
{
    var rng = new Random();
    return Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = rng.Next(-20, 55),
        Summary = Summaries[rng.Next(Summaries.Length)]
    })
    .ToArray();
}
```

USE KEYCLOAK TO STARTUP.CS

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddCors();

        services.AddControllers();

        var auth = services.AddAuthentication();

        auth.AddJwtBearer("myscheme", options =>
        {
            options.Authority = "http://localhost:8080/auth/realms/master";
            options.Audience = "testclient";
            options.RequireHttpsMetadata = false;
        });

        services.AddAuthorization(options =>
        {
            options.DefaultPolicy = new AuthorizationPolicyBuilder()
                .AddAuthenticationSchemes(new { "myscheme" })
                .RequireAuthenticatedUser()
                .Build();
        });
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();

        app.UseCors(builder =>
            builder.AllowAnyMethod()
                .AllowAnyHeader()
                .AllowCredentials()
                .SetIsOriginAllowed(s => true));

        app.UseAuthentication();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
     {
         endpoints.MapControllers();
     });
    }
}
```

Run

Make sure we run the project profile and not the IIS one.

Add Code To Front End to hit authenticated endpoint

```
import React, { ReactElement } from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import Keycloak from 'keycloak-js';

const keycloak = Keycloak(`/config/keycloak.json?ts=${new Date().getTime()}`);

async function DoWork() {
  await keycloak.init(
    {
      onLoad: 'login-required',
      enableLogging: true,
      checkLoginIframe: false
    });

  await keycloak.updateToken(120);
  const token = keycloak.token;
  const tokenParsed = keycloak.tokenParsed;

  const result = await fetch('http://localhost:5000/weatherforecast',
    {
      mode: "cors",
      headers: [
        ['authorization', `Bearer ${keycloak.token}`]
      ]
    }
  );

  ReactDOM.render(
    <React.StrictMode>
      <App json={await result.json()}></App>
    </React.StrictMode>,
    document.getElementById('root')
  );
}

function App(props: any): ReactElement {
  return <pre>{JSON.stringify(props.json, null, 2)}</pre>
}


DoWork();
```

Make Sure You Can See the Result


```
[
  {
    "date": "2021-02-12T11:40:36.7960094+00:00",
    "temperatureC": 30,
    "temperatureF": 85,
    "summary": "Freezing"
  },
  {
    "date": "2021-02-13T11:40:36.7960137+00:00",
    "temperatureC": 27,
    "temperatureF": 80,
    "summary": "Warm"
  },
  {
    "date": "2021-02-14T11:40:36.796014+00:00",
    "temperatureC": 28,
    "temperatureF": 82,
    "summary": "Chilly"
  },
  {
    "date": "2021-02-15T11:40:36.7960142+00:00",
    "temperatureC": -5,
    "temperatureF": 24,
    "summary": "Hot"
  },
  {
    "date": "2021-02-16T11:40:36.7960145+00:00",
    "temperatureC": 10,
    "temperatureF": 49,
    "summary": "Chilly"
  }
]
```

Using KeyCloak

Get Server Endpoint details

This gives details on end points on the server

http://localhost:8080/auth/realms/master/.well-known/openid-configuration

Get Token.

The following shows how to use PostMan to get a token. Note the data is form encoded the HTTP verb is POST.