Integrating KeyCloak Authorization with ASP.NET Core Tutorial

Basem Mohammed

ASP.NET Core Setup:

https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&tabs=visual-studio-code

KeyCloak Setup:

1. Download and extract keycloak-25.0.2.zip.
2. On the directory terminal, run: `bin\kc.bat start-dev`
3. On a browser, go to http://localhost:8080/.
    a. Complete the admin setup.
    b. Create a new realm.
    c. Create a client with:
        i. Client authentication on
        ii. A redirect URI to allow your API to receive authentication responses from Keycloak
    d. Once the client has been set up, the client secret can be retrieved from the ‘Credentials’ tab.
    e. Create a user and set a password.
    f. Return to the newly created client, create a role, and assign the user to the role.

ASP.NET Core set up to use KeyCloak:

1. Install .NET packages:

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package Microsoft.IdentityModel.Protocols.OpenIdConnect
```

2. Add KeyCloak settings to *appsettings.json*.

```
{
"Authentication": {
    "Keycloak": {
      "Authority": "http://localhost:8080/realms/todorealm", // replace
with your URL
      "Audience": "account", // replace with your audience
      "RequireHttpsMetadata": false
}}}
```

3. Update ‘Program.cs’ to use authentication.

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using TodoApi.Models;

var builder = WebApplication.CreateBuilder(args);
ConfigureServices(builder.Services, builder.Configuration);
var app = builder.Build();
ConfigureMiddleware(app);
app.Run();

void ConfigureServices(IServiceCollection services, IConfiguration
configuration)
{
    services.AddControllers();
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));

    services.AddEndpointsApiExplorer();
    services.AddSwaggerGen();

    ConfigureAuthentication(services, configuration);

    services.AddAuthorization();
}

void ConfigureAuthentication(IServiceCollection services, IConfiguration
configuration)
{
    var keycloakSettings = configuration.GetSection("Authentication:Keycloak");
    var authority = keycloakSettings["Authority"];
    var audience = keycloakSettings["Audience"];
    var requireHttpsMetadata = false;

    services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.Authority = authority;
        options.Audience = audience;
        options.RequireHttpsMetadata = requireHttpsMetadata;
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = authority,
            ValidateAudience = true,
            ValidAudience = audience,
            ValidateLifetime = true
        };
    });
}

void ConfigureMiddleware(WebApplication app)
{
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }
    app.UseHttpsRedirection();
    app.UseAuthentication();
    app.UseAuthorization();
    app.MapControllers();
        ValidAudience = audience,
            ValidateLifetime = true
        };
    });
}
```

4. Add the [Authorize] attribute to relevant controllers.

Add Policies

1. Create the method 'ConfigureAuthorization' in the Program.cs file.

```
void ConfigureAuthorization(IServiceCollection services, IConfiguration
configuration)
{
    services.AddAuthorization(opt=>
    {
        opt.AddPolicy("Policy1", policy =>
        {
            policy.RequireAuthenticatedUser();
            policy.RequireAssertion(context =>
            {
               //Policy Conditions
            });
        });
      opt.AddPolicy("Policy2", policy =>
        {
          policy.RequireRole("role_1")
          ...
        });

    });
}
```

2. Replace `services.AddAuthorization();` with `ConfigureAuthorization(services, configuration);` in `ConfigureServices`.
3. On relevant controllers, replace the [Authorize] attribute with [Authorize(Policy = "PolicyX")].
   NOTE: For combining policies, attributes must follow, e.g.

```
[Authorize(Policy = "Policy1")]
[Authorize(Policy = "Policy2")]
[Route("api/[controller]")]
[ApiController]
```

Test API

Example POST request for token retrieval

```
curl -X POST \
"http://localhost:8080/realms/<your-realm>/protocol/openid-connect/token" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d "client_id=<your-client-id>" \
-d "client_secret=<your-client-secret>" \
-d "grant_type=password" \
-d "username=<username>" \
-d "password=<user-password>"
```

RESPONSE

The response is in application/json format with the schema:

```
{
    "type": "object",
    "properties": {
        "access_token": {
            "type": "string"
        },
        "expires_in": {
            "type": "integer"
        },
        "refresh_expires_in": {
            "type": "integer"
        },
        "refresh_token": {
            "type": "string"
        },
        "token_type": {
            "type": "string"
        },
        "not-before-policy": {
            "type": "integer"
        },
        "session_state": {
            "type": "string"
        },
        "scope": {
            "type": "string"

    }
}
```

Example POST request to the ASP.NET Core API (with authentication)

```
curl -X POST 'http://localhost:5272/api/TodoItems' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer <access-token>\
-d '{"id":0,"name":"string","isComplete":true}'
```

Example GET request to the ASP.NET Core API (with authentication)

```
curl -X GET 'http://localhost:5272/api/TodoItems' \
-H 'accept: text/plain' \
-H 'Authorization: Bearer <access-token>'
```