**Request OAuth Token**

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Threading.Tasks;

using Microsoft.Extensions.Options;

using Newtonsoft.Json;

using ExampleName.Models;

namespace ExampleName

{

public class OAuthTokenService : IOAuthTokenService

{

private OAuthToken token = new OAuthToken();

private readonly OAuthSettings \_OAuthSettings;

public OAuthTokenService(IOptions<OAuthSettings> OAuthSettings)

{

this.\_OAuthSettings = OAuthSettings.Value;

}

public async Task<string> GetToken()

{

if (!this.token.IsValidAndNotExpiring)

{

this.token = await this.GetNewAccessToken();

}

return token.AccessToken;

}

private async Task<OAuthToken> GetNewAccessToken()

{

var token = new OAuthToken();

var client = new HttpClient();

var client\_id = this.\_OAuthSettings.ClientId;

var client\_secret = this.\_OAuthSettings.ClientSecret;

var clientCreds = System.Text.Encoding.UTF8.GetBytes($"{client\_id}:{client\_secret}");

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue("Basic", System.Convert.ToBase64String(clientCreds));

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

var postMessage = new Dictionary<string, string>

{

{ "grant\_type", "client\_credentials" },

{ "scope", "access\_token" },

{ "Resource", \_OAuthSettings.Resource }

};

var request = new HttpRequestMessage(HttpMethod.Post, this.\_OAuthSettings.TokenUrl)

{

Content = new FormUrlEncodedContent(postMessage)

};

var response = await client.SendAsync(request);

if (response.IsSuccessStatusCode)

{

var json = await response.Content.ReadAsStringAsync();

token = JsonConvert.DeserializeObject<OAuthToken>(json);

token.ExpiresAt = DateTime.UtcNow.AddSeconds(this.token.ExpiresIn);

}

else

{

throw new ApplicationException("Unable to retrieve access token from OAuth provider");

}

return token;

}

private class OAuthToken

{

[JsonProperty(PropertyName = "access\_token")]

public string AccessToken { get; set; }

[JsonProperty(PropertyName = "expires\_in")]

public int ExpiresIn { get; set; }

public DateTime ExpiresAt { get; set; }

public string Scope { get; set; }

[JsonProperty(PropertyName = "token\_type")]

public string TokenType { get; set; }

public bool IsValidAndNotExpiring

{

get

{

return !String.IsNullOrEmpty(this.AccessToken) && this.ExpiresAt > DateTime.UtcNow.AddSeconds(30);

}

}

}

}

}

**Get and Use Token:**

private HttpClient \_client;

private readonly IOAuthTokenService \_tokenService;

private readonly ApiEndpoints \_endpoints;

public GTFSApiService(HttpClient client, IOAuthTokenService tokenService, Options<ApiEndpoints> endpoints)

{

\_client = client;

\_tokenService = tokenService;

\_endpoints = endpoints.Value;

}

**Example on how we use it with the an api endpoint:**

public async Task<HttpResponseMessage> GetClosestStops(decimal lat, decimal lng, int number)

{

var token = await \_tokenService.GetToken();

\_client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

return await \_client.GetAsync(\_endpoints.ClosestStops + lat + "/" + lng + "/" + number);

}