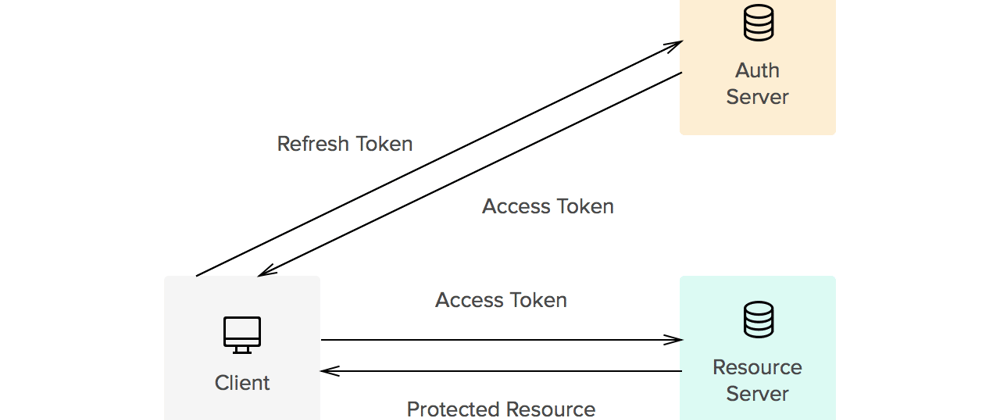
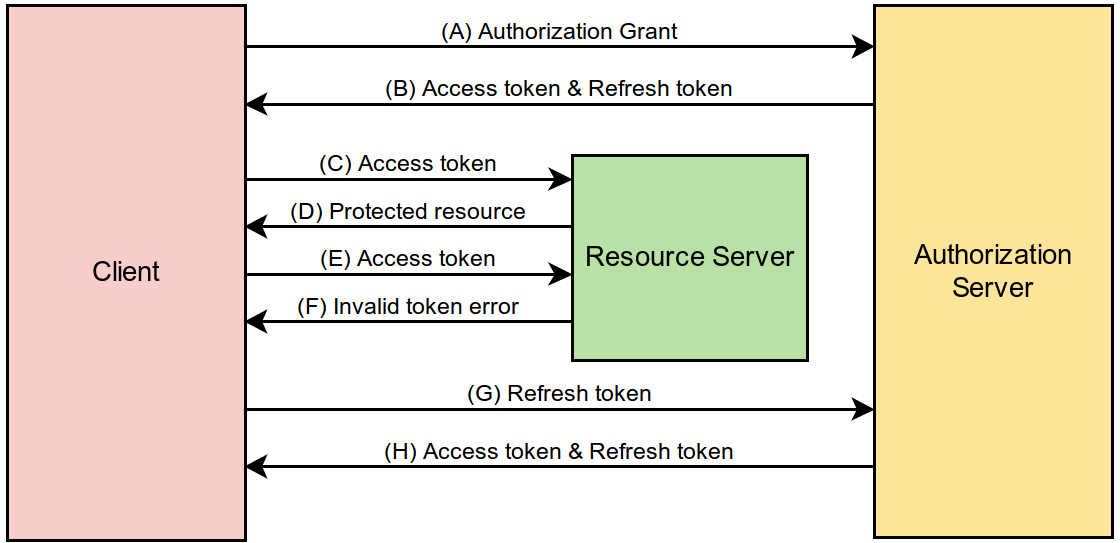
JWT Refersh Token with Middleware Using ASP.NET CORE 8.0

Working flow





Introduction

Refresh tokens and JSON Web Tokens (JWTs) are both used to authenticate and authorize users, but they have different purposes and functions:

* Refresh tokens

Allow users to stay logged in without having to re-enter their credentials. They are a special key that enables a client to retrieve new access tokens from an API or service. Refresh tokens typically have a longer expiration time than JWTs, so when a JWT expires, a refresh token can be used to get a new JWT. This improves the user experience and adds an extra layer of security for sensitive information.

* JWTs

A standard way for two parties to communicate securely, often used for managing authorization. JWTs can contain a variety of information, such as a user's name, email, and birth date. They are commonly used to create access tokens that allow users to access application or API resources.

Pre-requisites:

Install :

Microsoft.AspNetCore.Authentication.JwtBearer;

Microsoft.IdentityModel.Tokens;

System.Data.SqlClient;

Microsoft.IdentityModel.Tokens;

StartUp.cs:

We have to configure the jwt Bearer authentication in StartUp.cs

public class StartUp

{

private readonly IWebHostEnvironment \_env;

public IConfiguration Configuration { get; set; }

public StartUp(IConfiguration configuration, IWebHostEnvironment env)

{

Configuration = configuration;

\_env = env;

}

public void ConfigureServices(IServiceCollection services)

{

services.Configure<JWT>(Configuration.GetSection("JWT"));

var key = Encoding.ASCII.GetBytes(Configuration["JWT:Secret"]);

configure Bearer Authentication

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(x =>

{

x.RequireHttpsMetadata = false; // Set to true in production

x.SaveToken = true;

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(key),

ValidateIssuer = true,

ValidateAudience = true,

ValidIssuer = Configuration["JWT:ValidIssuer"],

ValidAudience = Configuration["JWT:ValidAudience"]

};

});

Add Role Policy

services.AddAuthorization(options =>

{

options.AddPolicy("RequireAdminRole", policy => policy.RequireRole("Admin"));

options.AddPolicy("RequireUserRole", policy => policy.RequireRole("User"));

});

Configure swagger for JWT Bearer Authentication

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "APIApplication", Version = "v1" });

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = @"Please provide authorization token to access restricted features.",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT",

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

Context Accessor for response,request

services.AddHttpContextAccessor();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// using middleware for refresh tokens:

app.UseMiddleware<RefreshTokenMiddleware>();

app.UseAuthentication();

app.UseAuthorization();

}  
}

**RefreshTokenMiddleware.cs**

using JWTRefreshTokenDemo.JWTDALayer;

using Microsoft.Extensions.Options;

using Newtonsoft.Json.Linq;

using System.IdentityModel.Tokens.Jwt;

namespace JWTRefreshTokenDemo.Models

{

public class RefreshTokenMiddleware

{

private readonly RequestDelegate \_next;

private readonly IOptions<JWT> \_jwt;

private readonly IOptions<ConnectionStrings> \_config;

public RefreshTokenMiddleware

(

RequestDelegate next,

IOptions<JWT> jwt,

IOptions<ConnectionStrings> config

)

{

\_next = next;

\_jwt = jwt;

\_config = config;

}

EnabledUnathourizedRoute

private bool IsEnabledUnathourizedRoute(HttpContext context)

{

List<string> enableRoutes = new List<string>

{

"/api/Account/Login",

"/api/Account/Register"

};

bool isEnableRoutes = false;

if (context.Request.Path.Value is not null)

{

isEnableRoutes = enableRoutes.Contains(context.Request.Path.Value);

}

return isEnableRoutes;

}

Checking token expired or not

private bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

Checking Refresh token is expired or not

private bool IsRefreshTokenExpired(string refreshToken)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtRefreshToken = tokenHandler.ReadToken(refreshToken);

return jwtRefreshToken.ValidTo < DateTime.UtcNow;

}

Invoke method to checking refresh token and jwt token and its expired return new access token

public async Task Invoke(HttpContext context, IServiceProvider serviceProvider)

{

using var scope = serviceProvider.CreateScope();

var \_repo = scope.ServiceProvider.GetRequiredService<IAuthRepo>();

var \_helpers = scope.ServiceProvider.GetRequiredService<Helpers>();

Headers token

var token = context.Request.Headers["Authorization"].FirstOrDefault()?.Split(" ").Last();

int tokenExist = 0;

string dbRefreshToken = null;

string dbToken = null;

if (token != null)

{

Validate Headers token

tokenExist = await \_helpers.ValidateDBToken(token);

}

if (tokenExist > 0)

{

Get tokens from the SQL Database

var getTokens = await \_helpers.GetDBTokenDetailsByToken(token);

dbRefreshToken = getTokens.RefreshToken;

dbToken = getTokens.Token;

}

Checking for both Token and Refresh token is null or empty.

if (string.IsNullOrEmpty(token) || string.IsNullOrEmpty(dbRefreshToken))

{

if (IsEnabledUnathourizedRoute(context))

{

await \_next(context);

}

else

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

context.Response.ContentType = "application/json";

await context.Response.WriteAsJsonAsync(new { message = "Invalid token credentials" });

}

return;

}

if (!string.IsNullOrEmpty(token) && !string.IsNullOrEmpty(dbRefreshToken) && token == dbToken)

{

Is Refresh token is expired or not

if (IsRefreshTokenExpired(dbRefreshToken))

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

context.Response.ContentType = "application/json";

await context.Response.WriteAsJsonAsync(new { message = "Refresh token is expired. Please login again." });

return;

}

Is Token is expired or not

if (IsTokenExpired(token))

{

Validate jwt token

var principal = \_helpers.GetPrincipalFromExpiredToken(token);

string userName = principal.Identity.Name;

var user = await \_repo.GetUserDetails(userName);

if (principal == null || userName != user.Register.UserName)

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

context.Response.ContentType = "application/json";

await context.Response.WriteAsJsonAsync(new { message = "Invalid token credentials" });

return;

}

Generate new access token

var newAccessToken = \_helpers.GenerateJwtToken(user.Register.UserName, user.Register.Email, user.Register.MemberNumber, user.Register.Role);

// still active Refreshtoken not expired. so we dont add new refreshtoken:

var newRefreshToken = dbRefreshToken;

int userID = user.Register.UserId;

var tokenModel = new TokenModel

{

Token = newAccessToken,

RefreshToken = newRefreshToken,

};

Update token into Sql database

var updateToken = await \_repo.UpdateTokenToDB(tokenModel, userID);

Send token into response headers

context.Response.Headers.Add("AccessToken", newAccessToken);

}

await \_next(context);

}

}

}

}

Helpers.cs

// validate Accesstoken token

public ClaimsPrincipal GetPrincipalFromExpiredToken(string token)

{

var secret = \_jwt.Value.Secret.ToString();

var validationTokenParameters = new TokenValidationParameters

{

ValidateAudience =false,

ValidateIssuer =false,

ValidateIssuerSigningKey =true,

ValidateLifetime =false,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret)),

};

var tokenHandler = new JwtSecurityTokenHandler();

var principal = tokenHandler.ValidateToken(token, validationTokenParameters, out SecurityToken securityToken);

if (securityToken is not JwtSecurityToken jwtSecurityToken || !jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256, StringComparison.InvariantCultureIgnoreCase))

throw new SecurityTokenException("Invalid Token");

return principal;

}

// Generate JWT Token

public string GenerateJwtToken(string userName, string email, string memberNumber, string role)

{

var authClaims = new List<Claim>

{

new Claim(ClaimTypes.Name, userName),

new Claim(ClaimTypes.Email, email),

new Claim(ClaimTypes.Role, role),

new Claim("MemberNumber",memberNumber),

new Claim(JwtRegisteredClaimNames.Jti,Guid.NewGuid().ToString())

};

var key = Encoding.UTF8.GetBytes(\_jwt.Value.Secret);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(authClaims),

Expires = DateTime.UtcNow.AddMinutes(\_jwt.Value.TokenValidityInMinutes),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature),

Issuer = \_jwt.Value.ValidIssuer,

Audience = \_jwt.Value.ValidAudience,

};

var tokenHandler = new JwtSecurityTokenHandler();

var securityToken = tokenHandler.CreateToken(tokenDescriptor);

var token = tokenHandler.WriteToken(securityToken);

return token;

}

// Generate refresh token

public string GenerateRefreshToken()

{

var randomNumber = new byte[32];

using (var randomNum = RandomNumberGenerator.Create())

{

randomNum.GetBytes(randomNumber);

}

var randomValue = Convert.ToBase64String(randomNumber);

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_jwt.Value.Secret.ToString());

var claim = new[]

{

new Claim("Randomvalue",randomValue)

};

var tokenDescriptor = new SecurityTokenDescriptor

{

Expires = DateTime.UtcNow.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256),

Subject = new ClaimsIdentity(claim)

};

var securityToken = tokenHandler.CreateToken(tokenDescriptor);

var refreshToken =tokenHandler.WriteToken(securityToken);

return refreshToken;

}

Create jwt and refresh token in Register method:

public async Task<RegisterResponse> SignUp(Register register)

{

RegisterResponse registerResponse = new RegisterResponse();

try

{

var user = await GetUserDetails(register.Email);

Validate Email

if (user != null && user.Register != null && user.Register.Email != null)

{

registerResponse.StatusCode = 400;

registerResponse.StatusMessage = "Email address already existed.";

return registerResponse;

}

Validate MemberNumber

var validateMemberNumber = await \_helpers.ValidateMemberNumber(register.MemberNumber);

if (validateMemberNumber > 0) {

registerResponse.StatusCode = 400;

registerResponse.StatusMessage = "MemberNumber already existed.";

return registerResponse;

}

Generate JWT Token

var generateJWTToken = \_helpers.GenerateJwtToken(register.UserName,register.Email,register.MemberNumber,register.Role);

Generate Refresh token

var generateRefreshToken = \_helpers.GenerateRefreshToken();

var storedProcName = "Create\_User\_Register";

var parameters = new SqlParameter[]

{

new SqlParameter("@FirstName",register.FirstName),

new SqlParameter("@LastName",register.LastName),

new SqlParameter("@UserName",register.UserName),

new SqlParameter("@Email",register.Email),

new SqlParameter("@Password",\_helpers.EncryptedPassword(register.Password)),

new SqlParameter("@MemberNumber",register.MemberNumber),

new SqlParameter("@Role",register.Role),

new SqlParameter("@AccessToken",generateJWTToken),

new SqlParameter("@RefreshToken",generateRefreshToken)

};

Insert into SQL Database

int rowsAffected = await \_sqlHelpers.InsertTable(storedProcName,parameters);

if (rowsAffected > 0) {

registerResponse.StatusCode =200;

registerResponse.StatusMessage = "User registered successfully";

return registerResponse;

}

registerResponse.StatusCode = 400;

registerResponse.StatusMessage = "User registration has been failed.";

}

catch (Exception ex) {

registerResponse.StatusCode = 500;

registerResponse.StatusMessage = ex.Message;

}

return registerResponse ;

}

Send jwt token into response headers while successfully logged in:

public async Task<LoginResponse> Login(LoginModel model)

{

LoginResponse loginResponse = new LoginResponse();

try

{

// validate email and password

int validateEmailAndPassword = await \_helpers.ValidateUserNameandPassword(model.Email, model.Password);

if(validateEmailAndPassword > 0)

{

var user = await GetUserDetails(model.Email);

var dbTokens = await GetTokensFromDB(model.Email);

loginResponse.TokenModel = new TokenModel();

// check refresh token expired:

if(\_helpers.IsRefreshTokenExpired(dbTokens.TokenModel.RefreshToken.ToString()))

{

loginResponse.TokenModel.RefreshToken = \_helpers.GenerateRefreshToken();

loginResponse.TokenModel.Token = \_helpers.GenerateJwtToken(user.Register.UserName,user.Register.Email,user.Register.MemberNumber,user.Register.Role);

await UpdateTokenToDB(loginResponse.TokenModel,user.Register.UserId);

\_httpContextAccessor.HttpContext.Response.Headers.Add("AccessToken",loginResponse.TokenModel.Token);

loginResponse.StatusCode = 200;

loginResponse.StatusMessage = "User loggedin successfully";

return loginResponse;

}

// loginResponse.TokenModel.Token = dbTokens.TokenModel.Token;

loginResponse.StatusCode = 200;

loginResponse.StatusMessage = "User loggedin successfully";

\_httpContextAccessor.HttpContext.Response.Headers.Add("AccessToken", dbTokens.TokenModel.Token.ToString());

return loginResponse;

}

loginResponse.StatusCode = 400;

loginResponse.StatusMessage = "Invalid Email or Password.";

}

catch (Exception ex) {

loginResponse.StatusCode = 500;

loginResponse.StatusMessage = "Something went wrong. Please try again.";

}

return loginResponse;

}