JWT Bearer Token and Refresh Token with using RefreshToken Middleware

To create RefreshToken Middleware to checking the token is expired or not.  
If its expired it will return’s new token.

StartUp.cs: // to configure the usage of RefreshTokenMiddleware.cs :

public void ConfigureServices(IServiceCollection services)

{

// dependeny injections

services.AddScoped<IAuthRepo, AuthRepo>();

services.AddScoped<IAuthService, AuthService>();

services.AddScoped<ITokenService, TokenService>();

services.AddScoped<Helpers>();

// configure

services.AddOptions();

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

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

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

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

// configure the JWT Brearer token 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"]

};

});

// configure the policy for role-based authentication:

services.AddAuthorization(options =>

{

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

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

});

// Testing for using Swagger:

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[] {}

}

});

});

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// using middleware for refresh tokens:

app.UseMiddleware<RefreshTokenMiddleware>();

}

Appsettingjson.cs:

"JWT": {

"ValidAudience": "https://localhost:7151",

"ValidIssuer": "https://localhost:7151",

"Secret": "THIS IS USED TO SIGN AND VERIFY JWT TOKENS, REPLACE IT WITH YOUR OWN SECRET, IT CAN BE ANY STRING",

"TokenValidityInMinutes": 1,

"RefreshTokenValidityInMinutes": 30,

"Key": "ThisismySecretKey",

"Issuer": "Test.com"

},

"AppSettings": {

"key": "ByYM000OLlMQG6VVVp1OH7Xzyr7gHuw1qvUC5dcGt3SNM"

}

Models:

public class AppSettings

{

public string Key { get; set; }

}

public class JWT

{

public string ValidAudience { get; set; }

public string ValidIssuer { get; set; }

public string Secret { get; set; }

public int TokenValidityInMinutes { get; set; }

public int RefreshTokenValidityInMinutes { get; set; }

public string Key { get; set; }

public string Issuer { get; set; }

}

RefreshTokenMiddleware:

public class RefreshTokenMiddleware

{

private readonly RequestDelegate \_next;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IOptions<ConnectionStrings> \_config;

public RefreshTokenMiddleware

(

RequestDelegate next,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor,

IOptions<ConnectionStrings> config

)

{

\_next = next;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

\_config = config;

}

// Refresh Token middleware method: In this method return new jwt token when token get’s expired and also save and update into the Sql Server Database

public async Task Invoke(HttpContext context, IServiceProvider serviceProvider)

{

using var scope = serviceProvider.CreateScope();

// configure the repositories services here:

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

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

// get headers token here:

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

int tokenExist = 0;

string dbRefreshToken = null;

string dbToken = null;

if (token != null)

{

// validate jwt token with sql server database jwt token

tokenExist = await \_helpers.ValidateDBToken(token);

}

if (tokenExist > 0)

{

// get token and refresh tokens from sql server database:

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

dbRefreshToken = getTokens.RefreshToken;

dbToken = getTokens.Token;

}

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

{

// Unauthorized routes like login,register, etc …..

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;

}

// checking token and DB refresh tokens is null or empty:

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

{

// validate given token is valid or not. If its valid use Identity.Name

var principal = \_helpers.GetPrincipalFromExpiredToken(token);

string userName = principal.Identity.Name;

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

// validating Sql database refresh token is expired or not

if (IsRefreshTokenExpired(dbRefreshToken))

{

// if refresh token is expired:

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

return;

}

// validating jwt token is expired or not

if (IsTokenExpired(token))

{

// validating token credential valid or not

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

{

If token is invalid

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

return;

}

// If token is valid and generate new accesstoken

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

// new refreshtoken if token expired add new refresh token.

// var newRefreshToken = \_helpers.GenerateRefreshToken();

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

var newRefreshToken = dbRefreshToken;

string userID = user.Register.UserID.ToString().ToUpper();

var tokenModel = new TokenModel

{

Token = newAccessToken,

RefreshToken = newRefreshToken,

};

// update into sql server database and Response Headers

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

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

// context.Response.Headers.Add("RefreshToken", newRefreshToken);

// await \_next(context);

//return;

}

await \_next(context);

}

}

With using cookies storage:

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>();

var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

var refreshTokenValidityInMinutes = \_jwt.Value.RefreshTokenValidityInMinutes;

var refreshToken = context.Request.Cookies["RefreshToken"];

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

var tokenCookie = context.Request.Cookies["Token"];

string connection = \_config.Value.AuthDBCon.ToString();

if (string.IsNullOrEmpty(tokenCookie) || string.IsNullOrEmpty(token) || string.IsNullOrEmpty(refreshToken))

{

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(refreshToken) && !string.IsNullOrEmpty(tokenCookie) && token == tokenCookie)

{

if (IsRefreshTokenExpired(refreshToken))

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

return;

}

if (IsTokenExpired(token))

{

var principal = \_helpers.GetPrincipalFromExpiredToken(token);

if (principal == null)

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

return;

}

string userName = principal.Identity.Name;

string role = principal.Claims.FirstOrDefault(c => c.Type == ClaimTypes.Role)?.Value;

var storeId = principal.Claims.FirstOrDefault(c => c.Type == "StoreId")?.Value;

using (SqlConnection conn = new SqlConnection(connection))

{

await conn.OpenAsync();

SqlCommand checkUser = new SqlCommand("Get\_UserName", conn);

checkUser.CommandType = System.Data.CommandType.StoredProcedure;

checkUser.Parameters.AddWithValue("@UserName", userName);

string existUser = (string)checkUser.ExecuteScalar();

await conn.CloseAsync();

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

if (existUser != userName)

{

context.Response.StatusCode = StatusCodes.Status404NotFound;

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

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

return;

}

else

{

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

var newRefreshToken = \_helpers.GenerateRefreshToken();

\_httpContextAccessor.HttpContext.Response.Cookies.Append("Token", newAccessToken, new CookieOptions

{

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

\_httpContextAccessor.HttpContext.Response.Cookies.Append("RefreshToken", newRefreshToken, new CookieOptions

{

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

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

string userID = user.Register.UserID.ToString().ToUpper();

var tokenModel = new TokenModel

{

Token = newAccessToken,

RefreshToken = newRefreshToken,

RefreshTokenExpiresIn = DateTime.Now.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes),

};

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

// context.Response.StatusCode = StatusCodes.Status200OK;

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

// await context.Response.WriteAsJsonAsync(new { message = "Token refreshed successfully..." });

context.Request.Headers["Authorization"] = $"Bearer {newAccessToken}";

// await \_next(context);

// return;

}

}

}

}

await \_next(context);

}

}

AutenRepo.cs:

public class AuthRepo : IAuthRepo

{

private readonly IOptions<ConnectionStrings> \_options;

private readonly Helpers \_helpers;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly ValidationHelper \_validationHelper;

private readonly SqlHelpers \_sqlHelpers;

public AuthRepo(

IOptions<ConnectionStrings> options,

Helpers helpers,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor,

ValidationHelper validationHelper

)

{

\_options = options;

\_helpers = helpers;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

\_validationHelper = validationHelper;

\_sqlHelpers = new SqlHelpers(\_options.Value.AuthDBCon.ToString());

}

public async Task<RegisterResponse> GetUserDetails(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

try

{

var storedProcName = "GetDetailsByUserName";

var userNameParameter = new SqlParameter[]

{

new SqlParameter("@UserName",userName)

};

var user = await \_sqlHelpers.GetSingleRow(storedProcName, userNameParameter);

if(user != null)

{

response.Register = new Register();

response.Register = new Register

{

UserID = (Guid)(user["UserID"]),

FirstName = user["FirstName"].ToString(),

LastName = user["LastName"].ToString(),

UserName = user["UserName"].ToString(),

Email = user["Email"].ToString(),

DateOfBirth = Convert.ToDateTime(user["DateofBirth"]),

ZipCode = user["ZipCode"].ToString(),

MobileNumber = user["MobileNumber"].ToString(),

StoreID = Convert.ToInt32(user["StoreID"]),

Role = user["Role"].ToString()

};

response.StatusCode = 200;

response.StatusMessage = "User details has been found.";

}

response.StatusCode = 400;

response.StatusMessage = "User details not found.";

}

catch(Exception ex)

{

response.StatusCode = 500;

response.StatusMessage = "Something went wrong.";

}

return response;

}

public async Task<RegisterResponse> GetTokensFromDB(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

//string connectionString = \_options.Value.AuthDBCon.ToString();

try

{

var storedProcName = "GetDetailsByUserName";

var userNameParameter = new SqlParameter[]

{

new SqlParameter("@UserName",userName)

};

var user = await \_sqlHelpers.GetSingleRow(storedProcName, userNameParameter);

if(user != null)

{

response.Register = new Register();

response.Register = new Register

{

Token = user["Token"].ToString(),

RefreshToken = user["RefreshToken"].ToString()

};

response.StatusCode = 200;

response.StatusMessage = "User tokens has been found.";

}

response.StatusCode = 400;

response.StatusMessage = "User tokens not found.";

}

catch (Exception ex)

{

response.StatusCode = 500;

response.StatusMessage = "Something went wrong.";

}

return response;

}

public async Task<RegisterResponse> SignUp(Register register)

{

RegisterResponse response = new RegisterResponse();

try

{

var user = await GetUserDetails(register.UserName);

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

{

response.StatusCode = 400;

response.StatusMessage = "Email Id already existed.";

response.Register = null;

return response;

}

// Generate token and refresh to save into sql server database

string token = \_helpers.GenerateJwtToken(register.UserName, register.Email, register.StoreID, register.DateOfBirth, register.Role);

string refreshtoken = \_helpers.GenerateRefreshToken();

var storedProcName = "RegisterUser";

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("@DateOfBirth", register.DateOfBirth),

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

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

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

new SqlParameter("@Token", token),

new SqlParameter("@RefreshToken", refreshtoken),

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

};

// save tokens into database

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

if (rowAffected > 0)

{

response.Register = new Register();

response.Register = register;

response.StatusCode = 200;

response.StatusMessage = "Registration has been successful.";

return response;

}

response.StatusCode = 400;

response.StatusMessage = "Registration has been failed.";

}

catch (Exception ex)

{

response.StatusCode = 500;

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

response.Register = null;

}

return response;

}

public async Task<LoginResponse> Login(LoginModel model)

{

LoginResponse response = null;

response= new LoginResponse();

// string connectionString = \_options.Value.AuthDBCon.ToString();

try

{

// validate username and password existed or not inside a database:

var storedProcName = "Check\_UserName\_Password";

int existedUserNamePassword = 0;

var parameters = new SqlParameter[]

{

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

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

};

existedUserNamePassword = await \_sqlHelpers.ExecuteIntScalar(storedProcName,parameters);

if (existedUserNamePassword > 0)

{

// Get token details using username

var getUser = await GetUserDetails(model.UserName);

var user = await GetTokensFromDB(model.UserName);

response.TokenModel = new TokenModel();

// validating refresh token is expired or not

if (\_helpers.IsRefreshTokenExpired(user.Register.RefreshToken.ToString()))

{

If refresh token is expired . Assign new refresh and jwt token:

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

response.TokenModel.Token = \_helpers.GenerateJwtToken(getUser.Register.UserName, getUser.Register.Email, getUser.Register.StoreID, getUser.Register.DateOfBirth, getUser.Register.Role);

// Update tokens into DB

await UpdateTokenToDB(response.TokenModel, getUser.Register.UserID.ToString().ToUpper());

// Send jwt token into Response Headers \_httpContextAccessor.HttpContext.Response.Headers.Add("AccessToken", response.TokenModel.Token);

response.StatusCode = 200;

response.StatusMessage = "User loggedin successfully.";

return response;

}

response.TokenModel.Token = user.Register.Token.ToString();

response.StatusCode = 200;

response.StatusMessage = "User loggedin successfully.";

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

}

else

{

response.StatusCode = 400;

response.StatusMessage = "Invalid UserName or Password.";

}

}

catch(Exception ex)

{

response.StatusCode = 500;

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

}

return response;

}

public async Task<TokenResponse> Refresh(TokenModel model)

{

TokenResponse tokenResponse = null;

tokenResponse = new TokenResponse();

try

{

var principal = \_helpers.GetPrincipalFromExpiredToken(model.Token);

string userName = principal.Identity.Name;

var user = await GetUserDetails(userName);

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

{

tokenResponse.StatusCode = 400;

tokenResponse.StatusMessage = "Invalid access token or refresh token";

tokenResponse.TokenModel = null;

}

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

var newRefreshToken = \_helpers.GenerateRefreshToken();

tokenResponse.TokenModel = new TokenModel();

tokenResponse.TokenModel.Token = newAccessToken;

tokenResponse.TokenModel.RefreshToken = newRefreshToken;

string userID = user.Register.UserID.ToString().ToUpper();

var updatetoken = await UpdateTokenToDB(tokenResponse.TokenModel, userID);

//\_httpContextAccessor.HttpContext.Response.Headers.Add("Authorization", $"Bearer {newAccessToken}");

\_httpContextAccessor.HttpContext.Response.Headers.Add("AccessToken", newAccessToken);

\_httpContextAccessor.HttpContext.Response.Headers.Add("RefreshToken", newRefreshToken);

tokenResponse.StatusCode = 200;

tokenResponse.StatusMessage = "Token refreshed successfully....";

}

catch (Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "An error occurred while refreshing token";

}

return tokenResponse;

}

public async Task<TokenResponse> GetTokenRefreshTokenById(string userId)

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

// string connectionString = \_options.Value.AuthDBCon.ToString();

try

{

var storedProcName = "GetTokens";

var userIdParameter = new SqlParameter("@UserID", userId);

var user = await \_sqlHelpers.GetSingleRow(storedProcName, userIdParameter);

if(user != null)

{

tokenResponse.TokenModel = new TokenModel

{

Token = user["Token"].ToString(),

RefreshToken = user["RefreshToken"].ToString(),

};

tokenResponse.StatusCode = 200;

tokenResponse.StatusMessage = "Token details has been found.";

}

}

catch (Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Something went wrong.";

}

return tokenResponse;

}

public async Task<TokenResponse> UpdateTokenToDB(TokenModel model, string userId)

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

var storedProcName = "UpdateTokens";

var parameters = new SqlParameter[]

{

new SqlParameter("@UserID", userId),

new SqlParameter("@Token",model.Token),

new SqlParameter("@RefreshToken", model.RefreshToken)

};

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

try

{

if(rowsAffected > 0)

{

tokenResponse.StatusCode = 200;

tokenResponse.StatusMessage = "Updated token successfully.";

}

else

{

tokenResponse.StatusCode = 400;

tokenResponse.StatusMessage = "Tokens updation has been failed.";

}

}

catch(Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Something went wrong.";

}

return tokenResponse;

}

}

Helpers.cs:

public class Helpers

{

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IOptions<AppSettings> \_appsettings;

private readonly IConfiguration \_configuration;

private readonly IOptions<ConnectionStrings> \_config;

private readonly IServiceProvider \_serviceProvider;

private readonly SqlHelpers \_sqlHelpers;

public Helpers(

IOptions<JWT> jwt,

IServiceProvider serviceProvider,

IHttpContextAccessor httpContextAccessor,

IOptions<AppSettings> appsettings,

IConfiguration configuration,

IOptions<ConnectionStrings> config

)

{

\_jwt = jwt;

\_serviceProvider = serviceProvider;

\_httpContextAccessor = httpContextAccessor;

\_appsettings = appsettings;

\_configuration = configuration;

\_config = config;

\_sqlHelpers = new SqlHelpers(\_config.Value.AuthDBCon.ToString());

}

// encrypted password

public string EncryptedPassword(string password)

{

if (string.IsNullOrEmpty(password))

{

return "";

}

else

{

byte[] passwordBase64 = ASCIIEncoding.ASCII.GetBytes(password);

string encryptPassword = Convert.ToBase64String(passwordBase64);

return encryptPassword;

}

}

// decrypted password

public string DecryptedPassword(string password)

{

if (string.IsNullOrEmpty(password))

{

return "";

}

else

{

byte[] encryptPassword = Convert.FromBase64String(password);

string DecryptedPassword = ASCIIEncoding.ASCII.GetString(encryptPassword);

return DecryptedPassword;

}

}

// Generate jwt token:

public string GenerateJwtToken(string userName, string email, int storeId, DateTime dateOfBirth, string role)

{

var tokenHandler = new System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();

var authClaims = new List<Claim>

{

new Claim(ClaimTypes.Name, userName),

new Claim(ClaimTypes.Email, email),

new Claim(ClaimTypes.Role, role),

new Claim("DateOfBirth", dateOfBirth.ToString("dd-MM-yyyy")),

new Claim("StoreId", storeId.ToString()),

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

};

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

var token = new JwtSecurityToken(

\_jwt.Value.ValidIssuer,

\_jwt.Value.ValidAudience,

expires: DateTime.UtcNow.AddMinutes(\_jwt.Value.TokenValidityInMinutes),

claims: authClaims,

signingCredentials: new SigningCredentials(key, SecurityAlgorithms.HmacSha256)

);

return tokenHandler.WriteToken(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);

// Create an array of claims

var claims = new[]

{

new Claim("randomValue", randomValue)

};

var tokenDescriptor = new SecurityTokenDescriptor

{

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

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

Subject = new ClaimsIdentity(claims)

};

var securityToken = tokenHandler.CreateToken(tokenDescriptor);

var refreshToken = tokenHandler.WriteToken(securityToken);

return refreshToken;

}

Validate jwt token expire or not

public bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

Validate Refresh Token expire or not

public bool IsRefreshTokenExpired(string refreshToken)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtRefreshToken = tokenHandler.ReadToken(refreshToken);

return jwtRefreshToken.ValidTo < DateTime.UtcNow;

}

Validate sql server database jwt token

public async Task<int> ValidateDBToken(string token)

{

var storedProcName = "ValidateDBToken";

var tokenParameter = new SqlParameter("@Token", token);

int tokenExist = await \_sqlHelpers.ExecuteIntScalar(storedProcName,tokenParameter);

if (tokenExist > 0)

{

return 1;

}

return 0;

//using (var conn = new SqlConnection(\_config.Value.AuthDBCon.ToString()))

//{

// await conn.OpenAsync();

// var command = new SqlCommand("ValidateDBToken", conn);

// command.CommandType = CommandType.StoredProcedure;

// command.Parameters.AddWithValue("@Token", token);

// int tokenExists = (int) await command.ExecuteScalarAsync();

// await conn.CloseAsync();

// if(tokenExists > 0)

// {

// return 1;

// }

// return 0;

//}

}

Get sql server database jwt tokens

public async Task<TokenModel> GetDBTokenDetailsByToken(string token)

{

var storedProcName = "GetDBTokensByToken";

var tokenParameter = new SqlParameter("@Token", token);

var user = await \_sqlHelpers.GetSingleRow(storedProcName,tokenParameter);

if (user != null)

{

return new TokenModel

{

Token = user["Token"].ToString(),

RefreshToken = user["RefreshToken"].ToString()

};

}

//using (var connection = new SqlConnection(\_config.Value.AuthDBCon.ToString()))

//{

// await connection.OpenAsync();

// var command = new SqlCommand("GetDBTokensByToken", connection);

// command.CommandType = CommandType.StoredProcedure;

// command.Parameters.AddWithValue("@Token", token);

// using (var reader = await command.ExecuteReaderAsync())

// {

// if (await reader.ReadAsync())

// {

// return new TokenModel

// {

// Token = reader["Token"].ToString(),

// RefreshToken = reader["RefreshToken"].ToString(),

// // UserID = reader["UserID"].ToString()

// };

// }

// }

//}

return null;

}