JWT REFRESH TOKEN WITHOUT USING COOKIES STORAGE

PROGRAM.CS:

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

var startup = new StartUp(builder.Configuration);

startup.ConfigureServices(builder.Services);

var app = builder.Build();

startup.Configure(app, app.Environment);

app.Run();

StartUp.cs:

using JWTRoleAuthentication.CommonLayer.Models;

using JWTRoleAuthentication.JWTBLL;

using JWTRoleAuthentication.JWTDAL;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

namespace JWTRoleAuthentication

{

public class StartUp

{

public IConfiguration Configuration { get; }

public StartUp(IConfiguration configuration)

{

Configuration = configuration;

}

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"]);

// services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

//.AddJwtBearer(options =>

//{

// options.TokenValidationParameters = new TokenValidationParameters

// {

// ValidateIssuer = true,

// ValidateAudience = true,

// ValidateLifetime = true,

// ValidateIssuerSigningKey = true,

// ValidIssuer = Configuration["Jwt:Issuer"],

// ValidAudience = Configuration["Jwt:Issuer"],

// IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))

// };

//});

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"]

};

});

services.AddMvc();

services.AddAuthorization(options =>

{

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

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

});

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[] {}

}

});

});

services.AddControllers();

services.AddHttpContextAccessor();

services.AddCors(options =>

{

options.AddDefaultPolicy(

policy =>

{

policy.AllowAnyOrigin()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1"));

//app.UseSwagger();

//app.UseSwaggerUI(c =>

//{

// c.SwaggerEndpoint("/swagger/v1/swagger.json", "APIApplication v1");

// c.RoutePrefix = string.Empty; // To serve the Swagger UI at the app's root URL

//});

}

// using middleware for refresh tokens:

app.UseMiddleware<RefreshTokenMiddleware>();

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

Appsetting.json:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"AuthDBCon": "Server=DESKTOP-1JNARAI\\SAISQLSERVER;Database=AuthenticationDB;User Id=sa;Password=sai123;Integrated Security=False;MultipleActiveResultSets=true;TrustServerCertificate=True;Encrypt=False"

// "AuthDBCon": "Data Source=DESKTOP-1JNARAI\\SAISQLSERVER;Initial Catalog=AuthenticationDB;User ID=sa;Password=sai123 "

},

"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": 5,

"RefreshTokenValidityInMinutes": 30,

"Key": "ThisismySecretKey",

"Issuer": "Test.com"

},

"AppSettings": {

"key": "ByYM000OLlMQG6VVVp1OH7Xzyr7gHuw1qvUC5dcGt3SNM"

},

//"Jwt": {

// "Key": "ThisismySecretKey",

// "Issuer": "Test.com"

//}

}

Helpers.cs:

using JWTRoleAuthentication.JWTDAL;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using System.Data.SqlClient;

using System.Data;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Security.Cryptography;

using System.Text;

namespace JWTRoleAuthentication.CommonLayer.Models

{

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;

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;

}

// 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;

}

}

// sha256 password:

public string EncryptSha256Password(string password)

{

using var sha256 = SHA256.Create();

byte[] hashPassword = Encoding.UTF8.GetBytes(password);

byte[] EncryptPasswordstorage = sha256.ComputeHash(hashPassword);

return Convert.ToBase64String(EncryptPasswordstorage);

}

//public string GenerateAccessToken(IEnumerable<Claim> claims)

//{

// //var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Secret"]));

// // var secret = \_configuration["AppSettings:Key"] ?? throw new InvalidOperationException("Secret Key is not configured");

// // var Key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret));

// // var creds = new SigningCredentials(Key, SecurityAlgorithms.HmacSha256);

// var validIssuer = \_jwt.Value.ValidIssuer;

// var ValidAudience = \_jwt.Value.ValidAudience;

// var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

// var secret = \_jwt.Value.Secret;

// var Key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret));

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: ValidAudience,

// expires: DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

// claims: claims,

// signingCredentials: new SigningCredentials(Key, SecurityAlgorithms.HmacSha256)

// );

// return new JwtSecurityTokenHandler().WriteToken(token);

//}

//public string GenerateJwtAccessToken(string username)

//{

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

// using var scope = \_serviceProvider.CreateScope();

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

// // var user = \_repo.GetByEmail(username);

// var authClaims = new List<Claim>

// {

// //new Claim(ClaimTypes.Name, user.Register.Email),

// //new Claim(ClaimTypes.Email, username),

// //new Claim("Id", user.Register.Id.ToString()),

// //new Claim("Role", user.Register.Role),

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

// };

// var validIssuer = \_jwt.Value.ValidIssuer;

// var validAudience = \_jwt.Value.ValidAudience;

// var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

// var secret = \_jwt.Value.Secret;

// var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret));

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: validAudience,

// expires: DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

// claims: authClaims,

// signingCredentials: new SigningCredentials(key, SecurityAlgorithms.HmacSha256)

// );

// return tokenHandler.WriteToken(token);

//}

//public JwtSecurityToken GenerateJwtToken(string username)

//{

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

// using var scope = \_serviceProvider.CreateScope();

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

// // var user = \_repo.GetByEmail(username);

// var authClaims = new List<Claim>

// {

// //new Claim(ClaimTypes.Name, user.Register.Email),

// //new Claim(ClaimTypes.Email, username),

// //new Claim("Id", user.Register.Id.ToString()),

// //new Claim("Role", user.Register.Role),

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

// };

// var validIssuer = \_jwt.Value.ValidIssuer;

// var validAudience = \_jwt.Value.ValidAudience;

// var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

// var secret = \_jwt.Value.Secret;

// var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secret));

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: validAudience,

// expires: DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

// claims: authClaims,

// signingCredentials: new SigningCredentials(key, SecurityAlgorithms.HmacSha256)

// );

// return 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.ToString()),

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

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

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

//};

// var validIssuer = \_jwt.Value.ValidIssuer;

// var validAudience = \_jwt.Value.ValidAudience;

// var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

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

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: validAudience,

// expires: DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

// claims: authClaims,

// signingCredentials: new SigningCredentials(key, SecurityAlgorithms.HmacSha256)

// );

// return tokenHandler.WriteToken(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);

}

//public string GenerateRefreshToken()

//{

// var randomNumber = new byte[640];

// using var generator = RandomNumberGenerator.Create();

// generator.GetBytes(randomNumber);

// return Convert.ToBase64String(randomNumber);

//}

public ClaimsPrincipal GetPrincipalFromExpiredToken(string token)

{

var secret = \_jwt.Value.Secret;

var tokenValidationParameters = 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, tokenValidationParameters, out SecurityToken securityToken);

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

throw new SecurityTokenException("Invalid Token");

return principal;

}

// public string GenerateRefreshToken()

//{

// var randomNumber = new byte[32];

// using (var rng = RandomNumberGenerator.Create())

// {

// rng.GetBytes(randomNumber);

// return Convert.ToBase64String(randomNumber);

// }

//}

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;

}

public bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

public bool IsRefreshTokenExpired(string refreshToken)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtRefreshToken = tokenHandler.ReadToken(refreshToken);

return jwtRefreshToken.ValidTo < DateTime.UtcNow;

}

public async Task<int> ValidateDBToken(string token)

{

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;

}

}

public async Task<TokenModel> GetDBTokenDetailsByToken(string token)

{

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;

}

//private bool IsTokenExpired(string token)

//{

// var tokenHandler = new JwtSecurityTokenHandler();

// var jwtToken = tokenHandler.ReadJwtToken(token);

// return jwtToken.ValidTo < DateTime.UtcNow;

//}

}

}

RefreshTokenMiddleware.cs:

using Microsoft.Extensions.Options;

using System.Data.SqlClient;

using System.Data;

using JWTRoleAuthentication.JWTDAL;

using System.Reflection;

using System.Security.Claims;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.Extensions.DependencyInjection;

namespace JWTRoleAuthentication.CommonLayer.Models

{

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;

}

With using cookie 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);

}

private bool IsEnabledUnathourizedRoute(HttpContext context)

{

List<string> enableRoutes = new List<string>

{

"/api/Account/Login",

"/api/Account/SignUp",

"/api/Account/Refresh",

"/api/Account/GetUserDetails"

};

bool isEnableRoutes = false;

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

{

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

}

return isEnableRoutes;

}

private bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

Without Cookies Storage:

private bool IsRefreshTokenExpired(string refreshToken)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtRefreshToken = tokenHandler.ReadToken(refreshToken);

return jwtRefreshToken.ValidTo < DateTime.UtcNow;

}

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

int tokenExist = 0;

string dbRefreshToken = null;

string dbToken = null;

if (token != null)

{

tokenExist = await \_helpers.ValidateDBToken(token);

}

if (tokenExist > 0)

{

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

dbRefreshToken = getTokens.RefreshToken;

dbToken = getTokens.Token;

}

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)

{

var principal = \_helpers.GetPrincipalFromExpiredToken(token);

string userName = principal.Identity.Name;

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

if (IsRefreshTokenExpired(dbRefreshToken))

{

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

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

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

//var tokenModel = new TokenModel

//{

// Token = newAccessToken,

// RefreshToken = newRefreshToken,

//};

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

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

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))

{

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;

}

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,

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

};

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);

}

}

}

}

DalLayer:

using JWTRoleAuthentication.CommonLayer.Models;

using Microsoft.Extensions.Options;

using Microsoft.Win32;

using System;

using System.Data;

using System.Data.SqlClient;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

namespace JWTRoleAuthentication.JWTDAL

{

public class AuthRepo : IAuthRepo

{

private readonly IOptions<ConnectionStrings> \_options;

private readonly Helpers \_helpers;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

public AuthRepo(

IOptions<ConnectionStrings> options,

Helpers helpers,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor

)

{

\_options = options;

\_helpers = helpers;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

}

public async Task<RegisterResponse> GetUserDetails(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

response.Register = new Register();

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

try

{

using(SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

using (SqlCommand cmd = new SqlCommand("GetDetailsByUserName",conn))

{

cmd.CommandType = System.Data.CommandType.StoredProcedure;

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

using(SqlDataReader reader = await cmd.ExecuteReaderAsync())

{

while(await reader.ReadAsync())

{

response.Register = new Register

{

UserID = reader.GetGuid(reader.GetOrdinal("UserID")),

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

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

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

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

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

DateOfBirth = reader.GetDateTime(reader.GetOrdinal("DateOfBirth")),

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

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

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

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

};

}

}

}

}

response.StatusCode = 200;

response.StatusMessage = "User details has been 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();

response.Register = new Register();

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

try

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

using (SqlCommand cmd = new SqlCommand("GetDetailsByUserName", conn))

{

cmd.CommandType = System.Data.CommandType.StoredProcedure;

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

using (SqlDataReader reader = await cmd.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

response.Register = new Register

{

//UserID = reader.GetGuid(reader.GetOrdinal("UserID")),

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

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

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

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

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

//DateOfBirth = reader.GetDateTime(reader.GetOrdinal("DateOfBirth")),

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

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

//StoreID = Convert.ToInt32(reader["StoreID"]),

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

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

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

};

}

}

}

}

response.StatusCode = 200;

response.StatusMessage = "User details has been 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();

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

try

{

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

string refreshtoken = \_helpers.GenerateRefreshToken();

using (SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

var user = await GetUserDetails(register.UserName);

if (user.Register.UserName != null)

{

response.StatusCode = 400;

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

response.Register = null;

await conn.CloseAsync();

return response;

}

SqlCommand cmd = new SqlCommand("RegisterUser", conn);

cmd.CommandType = System.Data.CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@FirstName", register.FirstName);

cmd.Parameters.AddWithValue("@LastName", register.LastName);

cmd.Parameters.AddWithValue("@UserName", register.UserName);

cmd.Parameters.AddWithValue("@Email", register.Email);

cmd.Parameters.AddWithValue("@Password", \_helpers.EncryptedPassword(register.Password));

cmd.Parameters.AddWithValue("@DateOfBirth", register.DateOfBirth);

cmd.Parameters.AddWithValue("@ZipCode", register.ZipCode);

cmd.Parameters.AddWithValue("@MobileNumber", register.MobileNumber);

cmd.Parameters.AddWithValue("@StoreID", register.StoreID);

//cmd.Parameters.AddWithValue("@Token", register.Token);

//cmd.Parameters.AddWithValue("@RefreshToken", register.RefreshToken);

cmd.Parameters.AddWithValue("@Token", token);

cmd.Parameters.AddWithValue("@RefreshToken", refreshtoken);

cmd.Parameters.AddWithValue("@Role", register.Role);

await cmd.ExecuteNonQueryAsync();

await conn.CloseAsync();

}

response.StatusCode = 200;

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

response.Register = register;

}

catch (Exception ex)

{

response.StatusCode = 500;

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

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

{

using(SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

SqlCommand checkLogin = new SqlCommand("Check\_UserName\_Password", conn);

checkLogin.CommandType = System.Data.CommandType.StoredProcedure;

checkLogin.Parameters.AddWithValue("@UserName", model.UserName);

checkLogin.Parameters.AddWithValue("@Password", \_helpers.EncryptedPassword(model.Password));

int variable = (int)checkLogin.ExecuteScalar();

await conn.CloseAsync();

if (variable > 0 )

{

var getUser = await GetUserDetails(model.UserName);

var user = await GetTokensFromDB(model.UserName);

response.TokenModel = new TokenModel();

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

{

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);

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

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

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

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

response.StatusCode = 200;

response.StatusMessage = "User loggedin successfully.";

return response;

}

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

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

response.StatusCode = 200;

response.StatusMessage= "User loggedin successfully.";

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

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

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

}

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....";

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

//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 GetUserDetails(existUser);

// if (existUser != userName)

// {

// tokenResponse.StatusCode = 400;

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

// }

// else

// {

// 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.Token = newAccessToken;

// tokenResponse.TokenModel.RefreshToken = newRefreshToken;

// //tokenResponse.TokenModel.RefreshTokenExpiresIn = DateTime.Now.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes);

// //\_httpContextAccessor.HttpContext.Response.Cookies.Append("Token", tokenResponse.TokenModel.Token, new CookieOptions

// //{

// // HttpOnly = true,

// // SameSite = SameSiteMode.Strict

// //});

// //\_httpContextAccessor.HttpContext.Response.Cookies.Append("RefreshToken", tokenResponse.TokenModel.RefreshToken, new CookieOptions

// //{

// // HttpOnly = true,

// // SameSite = SameSiteMode.Strict

// //});

// \_httpContextAccessor.HttpContext.Request.Headers["Authorization"] = $"Bearer {tokenResponse.TokenModel.Token}";

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

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

// 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

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

using (SqlCommand cmd = new SqlCommand("GetTokens", conn))

{

cmd.CommandType = System.Data.CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@UserID",userId);

using (SqlDataReader reader = await cmd.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

tokenResponse.TokenModel = new TokenModel

{

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

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

// RefreshTokenExpiresIn = Convert.ToDateTime(reader["RefreshTokenExpires"])

};

}

}

}

}

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();

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

using (SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

SqlCommand cmd = new SqlCommand("UpdateTokens", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@UserID", userId); // Example: Replace with actual UserID value

cmd.Parameters.AddWithValue("@Token", model.Token);

cmd.Parameters.AddWithValue("@RefreshToken", model.RefreshToken);

// cmd.Parameters.AddWithValue("@RefreshTokenExpires", model.RefreshTokenExpiresIn);

int rowsAffected = await cmd.ExecuteNonQueryAsync();

await conn.CloseAsync();

try

{

if (rowsAffected > 0)

{

tokenResponse.StatusCode = 200;

tokenResponse.StatusMessage = "Updated token successfully.";

}

else

{

tokenResponse.StatusCode = 404;

tokenResponse.StatusMessage = "No tokens updated.";

}

}

catch (SqlException ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Failed to update tokens. Please try again later.";

}

}

return tokenResponse;

}

}

}

Controller.cs:

using JWTRoleAuthentication.CommonLayer.Models;

using JWTRoleAuthentication.JWTBLL;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Options;

using System.Diagnostics.CodeAnalysis;

namespace JWTRoleAuthentication.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class AccountController : ControllerBase

{

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IAuthService \_authService;

private readonly Helpers \_helpers;

private readonly IOptions<JWT> \_jwt;

public AccountController(

IHttpContextAccessor httpContextAccessor,

IAuthService authService,

Helpers helpers,

IOptions<JWT> jwt

)

{

\_httpContextAccessor = httpContextAccessor;

\_authService = authService;

\_helpers = helpers;

\_jwt = jwt;

}

[HttpGet]

[Route("GetUserDetails")]

public async Task<RegisterResponse> GetUserDetails(string UserName)

{

RegisterResponse response = new RegisterResponse();

response.Register = new Register();

try

{

var user = await \_authService.GetUserDetails(UserName);

response.Register =user.Register;

response.StatusCode = 200;

response.StatusMessage = "Successfully retrived user details";

}catch(Exception ex)

{

response.StatusCode = 500;

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

}

return response;

}

[HttpPost]

[Route("SignUp")]

public async Task<IActionResult> SignUp(Register register)

{

var registerResponse = new RegisterResponse();

var result = await \_authService.SignUp(register);

registerResponse.Register = result.Register;

registerResponse.StatusCode = result.StatusCode;

registerResponse.StatusMessage = result.StatusMessage;

try

{

switch (result.StatusCode)

{

case 200:

return Ok(registerResponse);

case 400:

return BadRequest(registerResponse);

case 500:

return StatusCode(registerResponse.StatusCode, new { StatusCode = 500, StatusMessage = registerResponse.StatusMessage });

default:

return StatusCode(registerResponse.StatusCode, registerResponse);

}

}

catch(Exception ex)

{

registerResponse.StatusCode = result.StatusCode;

registerResponse.StatusMessage = result.StatusMessage;

return StatusCode(registerResponse.StatusCode, registerResponse.StatusMessage);

}

}

[HttpPost]

[Route("Login")]

public async Task<IActionResult> Login(LoginModel model)

{

var response = new LoginResponse();

var result = await \_authService.Login(model);

response.TokenModel = result.TokenModel;

response.StatusMessage = result.StatusMessage;

response.StatusCode = result.StatusCode;

try

{

switch (result.StatusCode) {

case 200:

return Ok(response);

case 400:

return BadRequest(response);

case 500:

return StatusCode(response.StatusCode, new {StatusCode = response.StatusCode,StatusMessage = result.StatusMessage});

default:

return StatusCode(response.StatusCode, response);

}

}

catch(Exception ex)

{

response.StatusMessage = result.StatusMessage;

response.StatusCode = result.StatusCode;

return StatusCode(response.StatusCode, new { StatusCode = response.StatusCode, StatusMessage = response.StatusMessage });

}

// return Ok(response);

}

[HttpPost]

[Route("Refresh")]

public async Task<IActionResult> Refresh(TokenModel model)

{

TokenResponse response = new TokenResponse();

int tokenExist = 0;

if ( model.Token != null)

{

tokenExist = await \_helpers.ValidateDBToken(model.Token);

}

string dbRefreshToken = null;

string dbToken = null;

if (tokenExist > 0)

{

var getTokens = await \_helpers.GetDBTokenDetailsByToken(model.Token);

dbRefreshToken = getTokens.RefreshToken;

dbToken = getTokens.Token;

}

if (dbRefreshToken != model.RefreshToken || dbToken != model.Token || \_helpers.IsRefreshTokenExpired(model.RefreshToken) )

{

return BadRequest("Invalid Token or RefreshToken Credentials.");

}

try

{

var result = await \_authService.Refresh(model);

response.TokenModel = result.TokenModel;

response.StatusMessage = result.StatusMessage;

response.StatusCode = result.StatusCode;

if (result.StatusCode == 400)

{

return BadRequest($"{result.StatusMessage}");

}

return StatusCode(response.StatusCode, response);

}

catch(Exception ex)

{

response.StatusCode = response.StatusCode;

response.StatusMessage = response.StatusMessage;

return StatusCode(response.StatusCode, new { StatusCode = response.StatusCode, StatusMessage = response.StatusMessage });

}

// return Ok(response);

}

}

}

Services:

using JWTRoleAuthentication.CommonLayer.Models;

using JWTRoleAuthentication.JWTDAL;

namespace JWTRoleAuthentication.JWTBLL

{

public class AuthService : IAuthService

{

private readonly IAuthRepo \_authRepo;

public AuthService(IAuthRepo authRepo)

{

\_authRepo = authRepo;

}

public async Task<RegisterResponse> SignUp(Register register)

{

var result = await \_authRepo.SignUp(register);

return result;

}

public async Task<RegisterResponse> GetUserDetails(string userName)

{

var result = await \_authRepo.GetUserDetails(userName);

return result;

}

public async Task<LoginResponse> Login(LoginModel model)

{

var result = await \_authRepo.Login(model);

return result;

}

public async Task<TokenResponse> Refresh(TokenModel model)

{

var result = await \_authRepo.Refresh(model);

return result;

}

}

}