JWT TOKEN AND REFRESH TOKEN AUTHENTICATION

JWT TOKEN:

JWT token is a base64url encoded string that is used to transmit the information between server and client. JWT token mostly contains the user information which is used for authorization.

JWT token can be sent through a URL, POST parameter, and HTTP header.  
The information that is sent by JWT is verified and trusted because it is [digitally signed](https://lazyhacker22.blogspot.com/2022/05/what-is-digital-signature.html).

**JWT token looks like:**  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9**.**eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ**.**SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

JWT token has three parts:<https://jwt.io/>

Header:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9  
*{“alg”: “HS256”,”typ”: “JWT”}*

**Payload:**  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ  
*{“sub”: “1234567890”,”name”: “John Doe”,”iat”: 1516239022}*

Signature:SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c  
*HMACSHA256(base64UrlEncode(header) + “.” +base64UrlEncode(payload), your-secret)*

**Uses of JWT token**\*Authorization:After successful authentication, the application receives the JWT token. That JWT token is passed in every request for accessing the services.  
\* The JWT token is also used for exchanging information.

**Refresh Token:**

A refresh token is a special key that enables a client for an API or service to retrieve new access tokens without requiring the user to perform a complete login. In other words, an application can exchange a valid refresh token for a new access token.

**Requirements for creating jwt tokens and refresh tokens in asp.net core6 web Api?**

**Install packages:** Microsoft.AspNetCore.Authentication.JwtBearer;

**Program.cs:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

// read the key from AppSettings:

var appSettingsread = configRoot.GetSection("AppSettings");

services.Configure<AppSettings>(appSettingsread);

// jwt authentication:

var settings = appSettingsread.Get<AppSettings>();

var secret = Encoding.UTF8.GetBytes(settings.Key);

// jwt authentication:

// default:

var key = configRoot["AppSettings:Key"] ?? throw new InvalidProgramException("Secret key not configured");

services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

// configure authentication

services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddCookie(options =>

options.Cookie.Name = "token"

)

.AddJwtBearer(options =>

{

options.SaveToken = true;

options.RequireHttpsMetadata = false;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

// ClockSkew = TimeSpan.Zero,

ClockSkew = new TimeSpan(0, 0, 5),

ValidAudience = configRoot["JWT:ValidAudience"],

ValidIssuer = configRoot["JWT:ValidIssuer"],

IssuerSigningKey = new SymmetricSecurityKey(secret),

//ValidateIssuerSigningKey = true,

//IssuerSigningKey = new SymmetricSecurityKey(secret),

//ValidateIssuer = false,

//ValidateAudience = false

};

options.Events = new JwtBearerEvents

{

OnMessageReceived = context =>

{

context.Token = context.Request.Cookies["token"];

return Task.CompletedTask;

}

};

});

Data Access Layer:

ILoginAuthRepo:

public interface ILoginAuthRepo

{

//bool Register(RegisterModel register);

// Task<RegisterModel> GetByEmail(string email);

public Response Register(RegisterModel register);

Response Login(LoginModel login);

Response GetByEmail(string email);

Response Refresh([FromBody] TokenModel model);

}

LoginAuthRepo:

Configs:

private readonly IOptions<ConnectionStrings> \_config;

private readonly IOptions<AppSettings> \_appsettings;

private readonly IConfiguration \_configuration;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

public LoginAuthRepo(

IOptions<ConnectionStrings> config,

IOptions<AppSettings> appsettings,

IConfiguration configuration,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor

)

{

\_config = config;

\_appsettings = appsettings;

\_configuration = configuration;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

}

Generate jwt token:

private JwtSecurityToken GenerateJwtToken(string username)

{

var authClaims = new List<Claim>

{

new Claim(ClaimTypes.Name,username),

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

};

var validIssuer = \_jwt.Value.ValidIssuer;

var ValidAudience = \_jwt.Value.ValidAudience;

var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

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

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;

}

GetPrincipalFromExpiredToken:

private ClaimsPrincipal? GetPrincipalFromExpiredToken(string token)

{

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

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;

}

Generate refresh token:

private static string RefreshTokenGenerator ()

{

var randomNumber = new byte[640];

using var generator = RandomNumberGenerator.Create();

generator.GetBytes(randomNumber);

return Convert.ToBase64String(randomNumber);

}

encrypted password

public static 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 static 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 static string EncryptSha256Password (string password)

{

using var sha256 = SHA256.Create();

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

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

return Convert.ToBase64String(EncryptPasswordstorage);

}

Register:

public Response Register(RegisterModel register)

{

Response resposne = new Response();

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

using (SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

SqlCommand checkEmail = new SqlCommand("check\_email\_register", conn);

checkEmail.CommandType = System.Data.CommandType.StoredProcedure;

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

string email = (string)checkEmail.ExecuteScalar();

if(email == register.Email)

{

resposne.StatusCode = 400;

resposne.StatusMessage = "Email id already existed";

}

else

{

SqlCommand cmd = new SqlCommand("Usp\_Register", conn);

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

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

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

// without encypted password

// cmd.Parameters.AddWithValue("@Password", register.Password); EncryptSha256Password

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

// cmd.Parameters.AddWithValue("@Password", EncryptSha256Password(register.Password));

int i = cmd.ExecuteNonQuery();

conn.Close();

RegisterModel model = new RegisterModel();

if (i > 0)

{

model.UserName= register.UserName;

model.Email= register.Email;

model.Password= register.Password;

resposne.StatusCode = 200;

resposne.StatusMessage = "Successfully registered";

resposne.Register = model;

}

else

{

resposne.StatusCode = 400;

resposne.StatusMessage = "Registration has been failed";

resposne.Register = null;

}

}

return resposne;

}

}

Login:

public Response Login(LoginModel login)

{

Response response = new Response();

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

using (SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

SqlCommand checkLogins = new SqlCommand("check\_email\_password", conn);

checkLogins.CommandType = System.Data.CommandType.StoredProcedure;

checkLogins.Parameters.AddWithValue("@Email", login.Email);

checkLogins.Parameters.AddWithValue("@Password", EncryptedPassword(login.Password));

int Variable = (int)checkLogins.ExecuteScalar();

conn.Close();

if (Variable == 1)

{

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

var KEY = Encoding.UTF8.GetBytes(\_appsettings.Value.Key);

var issuer = \_jwt.Value.ValidIssuer;

var audience = \_jwt.Value.ValidAudience;

var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

var refreshTokenValidityInMinutes = \_jwt.Value.RefreshTokenValidityInMinutes;

var tokenDescriptor = new Microsoft.IdentityModel.Tokens.SecurityTokenDescriptor

{

Subject = new System.Security.Claims.ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, login.Email.ToString()),

//new Claim("Id", Guid.NewGuid().ToString()),

new Claim(JwtRegisteredClaimNames.Sub, login.Email),

new Claim(JwtRegisteredClaimNames.Email, login.Email),

new Claim(JwtRegisteredClaimNames.Jti,

Guid.NewGuid().ToString())

}),

//Expires = DateTime.UtcNow.AddMinutes(10),

//Issuer = \_configuration["JWT:ValidIssuer"],

//Audience = \_configuration["JWT:ValidAudience"],

Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

Issuer = issuer,

Audience = audience,

SigningCredentials = new Microsoft.IdentityModel.Tokens.SigningCredentials(new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(KEY), Microsoft.IdentityModel.Tokens.SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

var refreshToken = RefreshTokenGenerator();

//\_ = int.TryParse(\_configuration["JWT:RefreshTokenValidityInMinutes"], out int refreshTokenValidityInMinutes);

var Token = tokenHandler.WriteToken(token);

TokenModel model = new TokenModel();

model.RefreshToken = refreshToken;

model.ExpireRefreshTokenTime = DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes);

model.AccessToken = Token;

response.Token = model;

response.Login = login;

response.StatusCode = 200;

response.StatusMessage = "Login Successful";

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

{

Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

//HttpOnly = true,

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

\_httpContextAccessor.HttpContext.Response.Cookies.Append("X-Refresh-Token", refreshToken, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

\_httpContextAccessor.HttpContext.Response.Cookies.Append("X-Username", login.Email, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

}

else

{

response.StatusCode = 400;

response.StatusMessage = "Invalid UserName or Password";

}

return response;

}

}

Refresh Token:

public Response Refresh([FromBody] TokenModel model)

{

Response response = new Response();

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

using(SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

var principal = GetPrincipalFromExpiredToken(model.AccessToken);

if (principal == null)

{

response.StatusCode = 400;

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

}

var username = principal?.Identity?.Name;

// var user = GetByEmail(username);

// check by username

SqlCommand checkEmail = new SqlCommand("check\_email\_register", conn);

checkEmail.CommandType = CommandType.StoredProcedure;

checkEmail.Parameters.AddWithValue("@Email", username);

string email = (string)checkEmail.ExecuteScalar();

conn.Close();

if (email != username)

{

response.StatusCode = 400;

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

} else

{

var newAccessToken = GenerateJwtToken(username);

var newRefreshToken = RefreshTokenGenerator();

model.RefreshToken = newRefreshToken;

var jwtToken = new JwtSecurityTokenHandler().WriteToken(newAccessToken);

model.AccessToken = jwtToken;

model.ExpireRefreshTokenTime = DateTime.UtcNow.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes);

if (model.AccessToken != null)

{

response.StatusCode = 200;

response.StatusMessage = "Refresh successfully..";

response.Token = model;

}

}

// Store Token data into the Cookies …. Refresh and Access token data

\_httpContextAccessor.HttpContext.Response.Cookies.Append("token", model.AccessToken, new CookieOptions()

{

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

//HttpOnly = true,

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

\_httpContextAccessor.HttpContext.Response.Cookies.Append("X-Refresh-Token", model.RefreshToken, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

\_httpContextAccessor.HttpContext.Response.Cookies.Append("X-Username", username, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

return response;

}

}

Controller:

using BusinessLogicLayer;

using CommonLayer.Models;

using DataAccessLayer.Entities;

using DataAccessLayer.Interfaces;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace StudentThreeTier.Controllers

{

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

[ApiController]

public class AccountController : ControllerBase

{

private readonly ILoginAuthService \_loginAuthService;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly ILoginAuthRepo \_loginAuthRepo;

public AccountController(

ILoginAuthRepo loginAuthRepo,

ILoginAuthService loginAuthService,

IHttpContextAccessor httpContextAccessor

)

{

\_loginAuthService = loginAuthService;

\_httpContextAccessor = httpContextAccessor;

\_loginAuthRepo = loginAuthRepo;

}

[HttpPost]

[Route("Login")]

public IActionResult Login(LoginModel loginModel)

{

var result = \_loginAuthService.Login(loginModel);

if (result.StatusCode != 200)

{

return BadRequest(new { message = result.StatusMessage });

}

//if(result)

//{

// return Ok(result);

//}

//else

//{

// return BadRequest(new { message = "Invalid username or password" });

//}

return Ok(new { result.Token,result.StatusMessage,result.StatusCode});

}

[HttpPost]

[Route("RegisterDetails")]

[Produces("application/json")]

public IActionResult RegisterDetails(RegisterModel register)

{

var result = \_loginAuthService.Register(register);

if(result.StatusCode != 200)

{

return BadRequest(new { message = result.StatusMessage});

}

return Ok(result);

}

[HttpPost]

[Route("Refresh-token")]

public async Task<IActionResult> RefreshToken([FromBody] TokenModel model)

{

\_ = \_httpContextAccessor.HttpContext.Request.Cookies.TryGetValue("token", out var token);

\_ = \_httpContextAccessor.HttpContext.Request.Cookies.TryGetValue("X-Refresh-Token", out var cookieRefreshToken);

if (token != model.AccessToken || cookieRefreshToken != model.RefreshToken || model.ExpireRefreshTokenTime < DateTime.UtcNow)

//if ( model.ExpireRefreshTokenTime < DateTime.UtcNow)

{

return BadRequest("Invalid token or refresh credentials");

// throw new InvalidProgramException("Invalid token credentials");

}

var result = \_loginAuthService.Refresh(model);

if(result.StatusCode != 200)

{

return BadRequest(new {message = result.StatusMessage});

}

return Ok(new { token = result.Token.AccessToken, refreshToken = result.Token.RefreshToken,expire = result.Token.ExpireRefreshTokenTime

});

}

[HttpGet]

[Route("GetByEmailId")]

public async Task<IActionResult> GetByEmailId(string email)

{

if(email == null)

{

return BadRequest();

}

var result = \_loginAuthService.GetByEmail(email);

if(result.StatusCode != 200) {

return NotFound("Email Address is not found.");

}

return Ok(new {Status = result.StatusCode,Message = result.StatusMessage,Details = result.Register});

}

}

}

Business Access Layer:

ILoginAuthService:

namespace BusinessLogicLayer

{

public interface ILoginAuthService

{

Response Register(RegisterModel register);

Response Login(LoginModel login);

Response Refresh([FromBody] TokenModel model);

Response GetByEmail(string email);

}

}

LoginAuthService:

using CommonLayer.Models;

using DataAccessLayer.Entities;

using DataAccessLayer.Interfaces;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BusinessLogicLayer

{

public class LoginAuthService : ILoginAuthService

{

private readonly ILoginAuthRepo \_repo;

public LoginAuthService(ILoginAuthRepo repo)

{

\_repo = repo;

}

public Response GetByEmail(string email)

{

var result = \_repo.GetByEmail(email);

return result;

}

public Response Login(LoginModel login)

{

var result = \_repo.Login(login);

return result;

}

public Response Refresh([FromBody] TokenModel model)

{

var result = \_repo.Refresh(model);

return result;

}

public Response Register(RegisterModel register)

{

var result = \_repo.Register(register);

return result;

}

}

}

SQL QUERY PROCEDURES:

use StudentDB;

CREATE TABLE Employee(

Id int Identity(1,1) primary key,

FirstName varchar(100),

LastName varchar(100),

Email varchar(100),

PhoneNumber varchar(100),

DOB DATE,

Salary decimal

);

--drop table Login;

CREATE TABLE Register(

Id Int Identity(1,1) primary key,

UserName varchar(100),

Email varchar(100),

Password varchar(100)

);

select \* from Register;

Check exis

create proc check\_email\_password(

@Email varchar(100),

@Password varchar(100)

)

as

begin

select Email,Password from Register where Email=@Email and Password =@Password;

end

exec check\_email\_password @Email='saikumar@gmail.com',@Password='sai123'

USE REFRESH TOKEN MIDDLEWARE

SET UP:

Add In startUp.cs:

public IConfiguration configRoot { get; set; }

// constructor

public StartUp(IConfiguration configuration, IWebHostEnvironment env)

{

configRoot = configuration;

\_env = env;

}

// configure services

public async void ConfigureServices(IServiceCollection services)

{

// add AddHttpContextAccessor();

services.AddHttpContextAccessor();

// login auth

services.AddScoped<ILoginAuthRepo , LoginAuthRepo>();

services.AddScoped<ILoginAuthService , LoginAuthService>();

services.AddScoped<ITokenServices, TokenServices>();

}

public void Configure(WebApplication app,IWebHostEnvironment env)

{

// using middleware

app.UseMiddleware<TokenRefreshMiddlewareService>();

}

using CommonLayer.Models;

using DataAccessLayer.Entities;

using Microsoft.AspNetCore.Http;

using Microsoft.Data.SqlClient;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Options;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Reflection;

using System.Security.Claims;

using System.Text;

using System.Threading.Tasks;

using Twilio.TwiML.Voice;

using System.Threading.Tasks;

using Amazon.Auth.AccessControlPolicy;

using Task = System.Threading.Tasks.Task;

namespace DataAccessLayer.Interfaces

{

public class TokenRefreshMiddlewareService

{

private readonly RequestDelegate \_next;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IOptions<ConnectionStrings> \_config;

public TokenRefreshMiddlewareService

(

RequestDelegate next,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor,

IOptions<ConnectionStrings> config

)

{

\_next = next;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

\_config = config;

}

// public async Task InvokeAsync(HttpContext context)

// {

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

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

// if (!string.IsNullOrEmpty(accessToken) && \_tokenService.GetPrincipalFromExpiredToken(accessToken) != null)

// {

// var principal = \_tokenService.GetPrincipalFromExpiredToken(accessToken);

// var claims = new[]

// {

// new Claim(ClaimTypes.Name, principal.Identity.Name),

// new Claim("Id",Guid.NewGuid().ToString()),

// };

// //var newAccessToken = \_tokenService.GenerateAccessToken(claims);

// var newAccessToken = \_tokenService.GenerateJwtToken(principal?.Identity.Name.ToString());

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

// context.Response.Headers.Add("token", newAccessToken.ToString());

// context.Response.Cookies.Append("refreshToken", newRefreshToken, new CookieOptions { HttpOnly = true });

// }

// await \_next(context);

// }

// public async System.Threading.Tasks.Task Invoke(HttpContext context, IServiceProvider serviceProvider)

public async Task Invoke(HttpContext context, IServiceProvider serviceProvider)

{

using var scope = serviceProvider.CreateScope();

var tokenService = scope.ServiceProvider.GetRequiredService<ITokenServices>();

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

var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

var refreshTokenValidityInMinutes = \_jwt.Value.RefreshTokenValidityInMinutes;

// var contextToken = \_httpContextAccessor.HttpContext.Request.Cookies["token"];

// var contextCookieRefreshToken = \_httpContextAccessor.HttpContext.Request.Cookies["X-Refresh-Token"];

// var token = context.Request.Cookies["token"];

var refreshToken = context.Request.Cookies["X-Refresh-Token"];

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

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

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

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

{

using (SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

var principal = tokenService.GetPrincipalFromExpiredToken(token);

var userName = principal?.Identity?.Name;

var claimId = principal?.Claims?.FirstOrDefault(c => c.Type == "Id");

var userId = claimId?.Value;

var user = \_repo.GetByEmail(userName);

if (principal != null)

{

SqlCommand checkEmail = new SqlCommand("check\_email\_register", conn);

checkEmail.CommandType = CommandType.StoredProcedure;

checkEmail.Parameters.AddWithValue("@Email", userName);

string email = (string)checkEmail.ExecuteScalar();

if (email != userName)

{

throw new Exception("Invalid token");

}

// delete -old refresh token from database.

// RevokeRefreshToken(refreshToken);

var newAccessToken = tokenService.GenerateJwtAccessToken(userName);

var newRefreshToken = tokenService.GenerateRefreshToken();

var expireRefreshToken = DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes);

context.Response.Cookies.Append("token", newAccessToken.ToString(), new CookieOptions

{

// Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

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

context.Response.Cookies.Append("X-Refresh-Token", refreshToken, new CookieOptions()

{

// Expires= DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes),

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

context.Response.Cookies.Append("X-Username", userName, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

UpdateRefreshToken(userId, userName, newAccessToken.ToString(), refreshToken, expireRefreshToken);

}

}

}

await \_next(context);

}

//try

//{

// string connection = \_config.Value.SNCon.ToString();

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

// {

// using (SqlConnection conn = new SqlConnection(connection))

// {

// conn.Open();

// var principal = tokenService.GetPrincipalFromExpiredToken(token);

// var userName = principal?.Identity?.Name;

// var claimId = principal?.Claims?.FirstOrDefault(c => c.Type == "Id");

// var userId = claimId?.Value;

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

// if (principal != null)

// {

// SqlCommand checkEmail = new SqlCommand("check\_email\_register",conn);

// checkEmail.CommandType = CommandType.StoredProcedure;

// checkEmail.Parameters.AddWithValue("@Email", userName);

// string email = (string)checkEmail.ExecuteScalar();

// if (email != userName)

// {

// throw new Exception("Invalid token");

// }

// // delete -old refresh token from database.

// // RevokeRefreshToken(refreshToken);

// var newAccessToken = tokenService.GenerateJwtAccessToken(userName);

// var newRefreshToken = tokenService.GenerateRefreshToken();

// var expireRefreshToken = DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes);

// context.Response.Cookies.Append("token", newAccessToken.ToString(), new CookieOptions

// {

// Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

// HttpOnly = true,

// SameSite = SameSiteMode.Strict

// });

// context.Response.Headers.Add("X-Access-Token", newAccessToken);

// context.Response.Cookies.Append("X-Refresh-Token", refreshToken, new CookieOptions() {

// // Expires= DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes),

// HttpOnly = true, SameSite = SameSiteMode.Strict });

// context.Response.Cookies.Append("X-Username", userName, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

// UpdateRefreshToken(userId, userName,newAccessToken.ToString(),refreshToken,expireRefreshToken);

// }

// else

// {

// // Log error or handle case where principal is null

// // \_httpContextAccessor.Response.StatusCode = 401; // Unauthorized

// // await context.Response.WriteAsync("Unauthorized: Invalid token.");

// }

// }

// }

// else

// {

// // Log error or handle case where token is null or empty

// // \_httpContextAccessor.Response.StatusCode = 401; // Unauthorized

// // throw new Exception("Invalid token or token credentials. Please login again.");

// }

//}

//catch (Exception ex)

//{

// // throw new Exception(ex.Message);

//}

}

// Save Refresh-token

public void SaveRefreshToken(string userId,string userName, string accessToken, string refreshToken, DateTime expireRefreshTokenTime)

{

string connectionString = \_config.Value.SNCon.ToString();

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// using(SqlCommand cmd = new SqlCommand("EXEC Save\_Refresh\_Token @AccessToken,@RefreshToken,@ExpireRefreshTokenTime",conn))

using (SqlCommand cmd = new SqlCommand("Save\_Refresh\_Token", conn))

{

cmd.CommandType = CommandType.StoredProcedure;

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

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

cmd.Parameters.AddWithValue("@AccessToken", accessToken);

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

cmd.Parameters.AddWithValue("@ExpireRefreshTokenTime", expireRefreshTokenTime);

cmd.ExecuteNonQuery();

}

conn.Close();

}

}

// Update Refresh-token

public void UpdateRefreshToken(string userId, string userName, string accessToken, string refreshToken, DateTime expireRefreshTokenTime)

{

string connectionString = \_config.Value.SNCon.ToString();

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// using(SqlCommand cmd = new SqlCommand("EXEC Save\_Refresh\_Token @AccessToken,@RefreshToken,@ExpireRefreshTokenTime",conn))

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

{

cmd.CommandType = CommandType.StoredProcedure;

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

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

cmd.Parameters.AddWithValue("@AccessToken", accessToken);

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

cmd.Parameters.AddWithValue("@ExpireRefreshTokenTime", expireRefreshTokenTime);

cmd.ExecuteNonQuery();

}

conn.Close();

}

}

// Delete Refresh-token

public void RevokeRefreshToken(string refreshToken)

{

string connectionString = \_config.Value.SNCon.ToString();

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// using(SqlCommand command = new SqlCommand("EXEC Revoke\_RefreshToken @RefreshToken",conn))

using (SqlCommand cmd = new SqlCommand("Revoke\_RefreshToken", conn))

{

cmd.CommandType = CommandType.StoredProcedure;

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

cmd.ExecuteNonQuery();

}

conn.Close();

}

}

}

}

ItokenServices.cs

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Security.Claims;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer.Interfaces

{

public interface

{

string GenerateAccessToken(IEnumerable<Claim> claims);

string GenerateRefreshToken();

ClaimsPrincipal GetPrincipalFromExpiredToken(string token);

JwtSecurityToken GenerateJwtToken(string username);

string GenerateJwtAccessToken(string username);

}

}

TokenServices.cs:

using CommonLayer.Models;

using DataAccessLayer.Entities;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Security.Claims;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer.Interfaces

{

public class TokenServices : ITokenServices

{

private readonly IOptions<ConnectionStrings> \_config;

private readonly IOptions<AppSettings> \_appsettings;

private readonly IConfiguration \_configuration;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IServiceProvider \_serviceProvider;

// private readonly UserManager<ApplicationUser> \_userManager;

public TokenServices(

IOptions<ConnectionStrings> config,

IOptions<AppSettings> appsettings,

IConfiguration configuration,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor,

IServiceProvider serviceProvider

// UserManager<ApplicationUser> userManager,

)

{

\_config = config;

\_appsettings = appsettings;

\_configuration = configuration;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

\_serviceProvider = serviceProvider;

// \_userManager = userManager;

}

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

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<ILoginAuthRepo>();

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

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<ILoginAuthRepo>();

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

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

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;

}

}

}

using CommonLayer.Models;

using DataAccessLayer.Entities;

using Microsoft.Data.SqlClient;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Protocols.OpenIdConnect;

using System;

using System.Collections.Generic;

using System.Data.Common;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Security.Claims;

using System.Security.Cryptography;

using Microsoft.Extensions.Configuration;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using Microsoft.AspNetCore.Mvc.TagHelpers;

using Microsoft.AspNetCore;

using Microsoft.AspNetCore.Http;

using Twilio.Jwt.AccessToken;

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.DependencyInjection;

namespace DataAccessLayer.Interfaces

{

public class LoginAuthRepo: ILoginAuthRepo

{

private readonly IOptions<ConnectionStrings> \_config;

private readonly IOptions<AppSettings> \_appsettings;

private readonly IConfiguration \_configuration;

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly ITokenServices \_services;

private readonly TokenService \_tokenService;

// private readonly UserManager<ApplicationUser> \_userManager;

public LoginAuthRepo(

IOptions<ConnectionStrings> config,

IOptions<AppSettings> appsettings,

IConfiguration configuration,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor,

ITokenServices services

// TokenService tokenService

)

{

\_config = config;

\_appsettings = appsettings;

\_configuration = configuration;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

\_services = services;

// \_tokenService = tokenService;

// \_userManager = userManager;

}

public Response GetByEmail(string email)

{

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

using (SqlConnection conn = new SqlConnection(connection))

{

Response response = new Response();

conn.Open();

// check email:

SqlCommand checkEmail = new SqlCommand("check\_email\_register", conn);

checkEmail.CommandType = System.Data.CommandType.StoredProcedure;

checkEmail.Parameters.AddWithValue("@Email", email);

string emailid = (string)checkEmail.ExecuteScalar();

if( emailid != email )

{

response.StatusCode = 404;

response.StatusMessage = "Invalid email id";

}

else

{

SqlCommand cmd = new SqlCommand("usp\_getby\_email", conn);

SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd);

dataAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;

dataAdapter.SelectCommand.Parameters.AddWithValue("@Email", email);

DataTable dataTable = new DataTable();

dataAdapter.Fill(dataTable);

var register = new RegisterModel();

conn.Close();

if (dataTable.Rows.Count > 0)

{

register.Id = Convert.ToInt32(dataTable.Rows[0]["Id"]);

register.UserName = Convert.ToString(dataTable.Rows[0]["UserName"]);

register.Email = Convert.ToString(dataTable.Rows[0]["Email"]);

register.Password = Convert.ToString(dataTable.Rows[0]["Password"]);

register.Role = Convert.ToString(dataTable.Rows[0]["Role"]);

response.StatusMessage = "Ok";

response.StatusCode = 200;

response.Register = register;

}

else

{

return null;

}

}

// return await Task.FromResult(register);

// return register;

return response;

}

}

public Response Login(LoginModel login)

{

Response response = new Response();

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

using (SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

SqlCommand checkLogins = new SqlCommand("check\_email\_password", conn);

checkLogins.CommandType = System.Data.CommandType.StoredProcedure;

checkLogins.Parameters.AddWithValue("@Email", login.Email);

checkLogins.Parameters.AddWithValue("@Password", EncryptedPassword(login.Password));

int Variable = (int)checkLogins.ExecuteScalar();

conn.Close();

if (Variable == 1)

{

var getbyemail = GetByEmail(login.Email);

Console.WriteLine(login);

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

var KEY = Encoding.UTF8.GetBytes(\_appsettings.Value.Key);

var issuer = \_jwt.Value.ValidIssuer;

var audience = \_jwt.Value.ValidAudience;

var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

var refreshTokenValidityInMinutes = \_jwt.Value.RefreshTokenValidityInMinutes;

var userId = getbyemail?.Register?.Id.ToString();

// var tokenDescriptor = new Microsoft.IdentityModel.Tokens.SecurityTokenDescriptor

// {

// //Subject = new System.Security.Claims.ClaimsIdentity(new Claim[]

// //{

// // new Claim(ClaimTypes.Name, login.Email.ToString()),

// // new Claim("Id", getbyemail?.Register?.Id.ToString()),

// // new Claim(ClaimTypes.Role, getbyemail.Register.Role.ToString()),

// // new Claim(JwtRegisteredClaimNames.Sub, login.Email),

// // new Claim(JwtRegisteredClaimNames.Email, login.Email),

// // new Claim(JwtRegisteredClaimNames.Jti,

// // Guid.NewGuid().ToString())

// //}),

// //Expires = DateTime.UtcNow.AddMinutes(10),

// //Issuer = \_configuration["JWT:ValidIssuer"],

// //Audience = \_configuration["JWT:ValidAudience"],

// Subject = new ClaimsIdentity(new Claim[]

//{

// new Claim(ClaimTypes.Name, login.Email.ToString()),

// new Claim("Id", userId),

// new Claim(ClaimTypes.Role, getbyemail.Register.Role.ToString()),

// new Claim(JwtRegisteredClaimNames.Sub, login.Email),

// new Claim(JwtRegisteredClaimNames.Email, login.Email),

// new Claim(JwtRegisteredClaimNames.Jti,

// Guid.NewGuid().ToString())

//}),

// Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

// Issuer = issuer,

// Audience = audience,

// SigningCredentials = new Microsoft.IdentityModel.Tokens.SigningCredentials(new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(KEY), Microsoft.IdentityModel.Tokens.SecurityAlgorithms.HmacSha256Signature)

// };

// var token = tokenHandler.CreateToken(tokenDescriptor);

// var refreshToken = RefreshTokenGenerator();

var token = \_services.GenerateJwtToken(login.Email);

var refreshToken = \_services.GenerateRefreshToken();

//\_ = int.TryParse(\_configuration["JWT:RefreshTokenValidityInMinutes"], out int refreshTokenValidityInMinutes);

var Token = tokenHandler.WriteToken(token);

TokenModel model = new TokenModel();

model.RefreshToken = refreshToken;

model.ExpireRefreshTokenTime = DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes);

model.AccessToken = Token;

response.Token = model;

response.Login = login;

response.StatusCode = 200;

response.StatusMessage = "Login Successful";

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

{

// Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

//HttpOnly = true,

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

\_httpContextAccessor.HttpContext.Response.Cookies.Append("X-Refresh-Token", refreshToken, new CookieOptions() {

// Expires = DateTime.UtcNow.AddMinutes(refreshTokenValidityInMinutes),

HttpOnly = true, SameSite = SameSiteMode.Strict });

\_httpContextAccessor.HttpContext.Response.Cookies.Append("X-Username", login.Email, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

SaveRefreshToken(userId,login.Email,model.AccessToken, model.RefreshToken, model.ExpireRefreshTokenTime);

}

else

{

response.StatusCode = 400;

response.StatusMessage = "Invalid UserName or Password";

}

return response;

}

}

private static string RefreshTokenGenerator()

{

var randomNumber = new byte[640];

using var generator = RandomNumberGenerator.Create();

generator.GetBytes(randomNumber);

return Convert.ToBase64String(randomNumber);

}

// register user details

public Response Register(RegisterModel register)

{

Response resposne = new Response();

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

using (SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

SqlCommand checkEmail = new SqlCommand("check\_email\_register", conn);

checkEmail.CommandType = System.Data.CommandType.StoredProcedure;

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

string email = (string)checkEmail.ExecuteScalar();

if(email == register.Email)

{

resposne.StatusCode = 400;

resposne.StatusMessage = "Email id already existed";

}

else

{

SqlCommand cmd = new SqlCommand("Usp\_Register", conn);

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

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

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

// without encypted password

// cmd.Parameters.AddWithValue("@Password", register.Password); EncryptSha256Password

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

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

// cmd.Parameters.AddWithValue("@Password", EncryptSha256Password(register.Password));

int i = cmd.ExecuteNonQuery();

conn.Close();

RegisterModel model = new RegisterModel();

if (i > 0)

{

model.UserName= register.UserName;

model.Email= register.Email;

model.Password= register.Password;

resposne.StatusCode = 200;

resposne.StatusMessage = "Successfully registered";

resposne.Register = model;

}

else

{

resposne.StatusCode = 400;

resposne.StatusMessage = "Registration has been failed";

resposne.Register = null;

}

}

return resposne;

}

}

// get refresh token

//public async Task Refresh([FromBody] TokenModel model)

//{

// Response response = new Response();

// string accessToken = model.AccessToken;

// string refreshToken = model.RefreshToken;

// var principal = GetPrincipalFromExpiredToken(model.AccessToken);

// if (principal == null)

// {

// response.StatusCode = 400;

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

// }

// var username = principal?.Identity?.Name;

// // check by username

// var newAccessToken = GenerateJwtToken(username);

// var newRefreshToken = RefreshTokenGenerator();

// model.RefreshToken = newRefreshToken;

// var jwtToken = new JwtSecurityTokenHandler().WriteToken(newAccessToken);

// model.AccessToken = jwtToken;

// await Task.FromResult(response);

//}

// create token:

private JwtSecurityToken GenerateJwtToken(string username)

{

var user = GetByEmail(username);

var authClaims = new List<Claim>

{

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

new Claim(ClaimTypes.Email,username),

// new Claim("Id",Guid.NewGuid().ToString()),

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

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;

}

//GetPrincipalFromExpiredToken:

private ClaimsPrincipal? GetPrincipalFromExpiredToken(string token)

{

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

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;

}

// refresh token

public Response Refresh([FromBody] TokenModel model)

{

Response response = new Response();

//string accessToken = model.AccessToken;

//string refreshToken = model.RefreshToken;

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

using(SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

var principal = GetPrincipalFromExpiredToken(model.AccessToken);

if (principal == null)

{

response.StatusCode = 400;

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

}

var username = principal?.Identity?.Name;

var claimId = principal?.Claims?.FirstOrDefault(c=> c.Type == "Id");

var userId = claimId?.Value?.ToString();

// var user = GetByEmail(username);

// check by username

SqlCommand checkEmail = new SqlCommand("check\_email\_register", conn);

checkEmail.CommandType = CommandType.StoredProcedure;

checkEmail.Parameters.AddWithValue("@Email", username);

string email = (string)checkEmail.ExecuteScalar();

conn.Close();

if (email != username)

{

response.StatusCode = 400;

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

} else

{

// revoke old refresh-token

// RevokeRefreshToken(model.RefreshToken);

var newAccessToken = GenerateJwtToken(username);

var newRefreshToken = RefreshTokenGenerator();

model.RefreshToken = newRefreshToken;

var jwtToken = new JwtSecurityTokenHandler().WriteToken(newAccessToken);

model.AccessToken = jwtToken;

model.ExpireRefreshTokenTime = DateTime.UtcNow.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes);

if (model.AccessToken != null)

{

response.StatusCode = 200;

response.StatusMessage = "Refresh successfully..";

response.Token = model;

// save new refresh-token

// SaveRefreshToken(userId,username,model.AccessToken,model.RefreshToken,model.ExpireRefreshTokenTime);

}

}

\_httpContextAccessor.HttpContext.Response.Cookies.Append

("token", model.AccessToken, new CookieOptions()

{

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

//HttpOnly = true,

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

\_httpContextAccessor.HttpContext.Response.Cookies.Append

("X-Refresh-Token", model.RefreshToken, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

\_httpContextAccessor.HttpContext.Response.Cookies.Append

("X-Username", username, new CookieOptions() { HttpOnly = true, SameSite = SameSiteMode.Strict });

return response;

}

}

// encrypted password

public static 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 static 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 static string EncryptSha256Password(string password)

{

using var sha256 = SHA256.Create();

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

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

return Convert.ToBase64String(EncryptPasswordstorage);

}

// Save Refresh-token

public void SaveRefreshToken( string userId,string userName,string accessToken,string refreshToken, DateTime expireRefreshTokenTime)

{

string connectionString = \_config.Value.SNCon.ToString();

using(SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// using(SqlCommand cmd = new SqlCommand("EXEC Save\_Refresh\_Token @AccessToken,@RefreshToken,@ExpireRefreshTokenTime",conn))

using (SqlCommand cmd = new SqlCommand("Save\_Refresh\_Token", conn))

{

cmd.CommandType = CommandType.StoredProcedure;

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

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

cmd.Parameters.AddWithValue("@AccessToken", accessToken);

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

cmd.Parameters.AddWithValue("@ExpireRefreshTokenTime",expireRefreshTokenTime);

cmd.ExecuteNonQuery();

}

conn.Close();

}

}

// Delete Refresh-token

public void RevokeRefreshToken(string refreshToken)

{

string connectionString = \_config.Value.SNCon.ToString();

using( SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// using(SqlCommand command = new SqlCommand("EXEC Revoke\_RefreshToken @RefreshToken",conn))

using (SqlCommand cmd = new SqlCommand("Revoke\_RefreshToken", conn))

{

cmd.CommandType = CommandType.StoredProcedure;

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

cmd.ExecuteNonQuery ();

}

conn.Close ();

}

}

public RefreshTokenModel GetUserIdandEmail(string userId,string UserName)

{

RefreshTokenModel refreshTokenModel = new RefreshTokenModel();

string connectionString = \_config.Value.SNCon.ToString();

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

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

{

cmd.CommandType = CommandType.StoredProcedure;

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

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

using (SqlDataReader reader = cmd.ExecuteReader())

{

if (reader.Read())

{

refreshTokenModel = new RefreshTokenModel

{

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

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

};

}

}

}

}

return refreshTokenModel;

}

public string GenerateJwtAccessToken(string username)

{

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

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

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

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

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

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);

}