JWT TOKEN AND REFRESH TOKEN USING MIDDLEWARE

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();

});

}

}

}

Program.cs:

using JWTRoleAuthentication;

using Microsoft.AspNetCore.Hosting;

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();

appsettingjson:

{

"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": 3,

"RefreshTokenValidityInMinutes": 30,

"Key": "ThisismySecretKey",

"Issuer": "Test.com"

},

"AppSettings": {

"key": "ByYM000OLlMQG6VVVp1OH7Xzyr7gHuw1qvUC5dcGt3SNM"

},

//"Jwt": {

// "Key": "ThisismySecretKey",

// "Issuer": "Test.com"

//}

}

AuthRepo.cs: DAL

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<LoginResponse> Login(LoginModel model)

{

LoginResponse response = null;

response= new LoginResponse();

// response.TokenModel = new TokenModel();

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);

string userId = getUser.Register.UserID.ToString().ToUpper();

// var tokenfromdb = await GetTokensFromDB(userId);

var tokensRefresh = await GetTokenRefreshTokenById(userId);

response.TokenModel = new TokenModel

{

Token = user.Register.Token.ToString(),

RefreshToken = user.Register.RefreshToken.ToString(),

RefreshTokenExpiresIn = DateTime.UtcNow.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes)

};

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

{

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

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

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

{

Expires = response.TokenModel.RefreshTokenExpiresIn,

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

response.StatusCode = 200;

response.StatusMessage= "User loggedin successfully.";

}

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;

}

private bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

public async Task<RegisterResponse> SignUp(Register register)

{

RegisterResponse response = new RegisterResponse();

response.Register = register;

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();

var refreshTokenExpire = DateTime.UtcNow.AddMinutes(\_jwt.Value.RefreshTokenValidityInMinutes) ;

using(SqlConnection conn = new SqlConnection(connectionString)) {

await conn.OpenAsync();

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

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

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

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

if(userName == register.UserName)

{

response.StatusCode = 400;

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

}

else

{

// FOR REGISTER ONLY

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

// FOR RefreshToken AND Register

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);

cmd.Parameters.AddWithValue("@RefreshTokenExpires", refreshTokenExpire);

await cmd.ExecuteNonQueryAsync();

}

await conn.CloseAsync();

}

response.StatusCode= 200;

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

}catch(Exception ex)

{

response.StatusCode = 500;

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

}

return response;

}

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> Refresh(TokenModel model)

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

try

{

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

if (principal == null)

{

tokenResponse.StatusCode = 400;

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

}

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;

// string role = principal.Claims.FirstOrDefault(c => c.ValueType == "role")?.Value;

// Check if username exists in your database or system

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

{

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

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

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

{

Expires = tokenResponse.TokenModel.RefreshTokenExpiresIn,

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

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> 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; // Or appropriate status code for no rows updated

tokenResponse.StatusMessage = "No tokens updated.";

}

}

catch (SqlException ex)

{

// Log or handle SQL exceptions

Console.WriteLine($"SQL Exception: {ex.Message}");

tokenResponse.StatusCode = 500;

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

}

}

return tokenResponse;

}

}

}

Helper.cs:

using JWTRoleAuthentication.JWTDAL;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

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 IServiceProvider \_serviceProvider;

public Helpers(

IOptions<JWT> jwt,

IServiceProvider serviceProvider,

IHttpContextAccessor httpContextAccessor,

IOptions<AppSettings> appsettings,

IConfiguration configuration

)

{

\_jwt = jwt;

\_serviceProvider = serviceProvider;

\_httpContextAccessor = httpContextAccessor;

\_appsettings = appsettings;

\_configuration = configuration;

}

// 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 bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

}

}

Uisng RefreshToken Middleware:

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;

}

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;

}

private bool IsRefreshTokenExpired(string refreshToken)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtRefreshToken = tokenHandler.ReadToken(refreshToken);

return jwtRefreshToken.ValidTo < DateTime.UtcNow;

}

}

}

public string GenerateRefreshToken() {

var randomNumber = new byte[32];

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

rng.GetBytes(randomNumber); }

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_secretKey);

var tokenDescriptor = new SecurityTokenDescriptor {

Expires = DateTime.UtcNow.AddDays(7), // Set the expiration time as needed

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature) }; var securityToken = tokenHandler.CreateToken(tokenDescriptor); var refreshToken = tokenHandler.WriteToken(securityToken); return refreshToken;

}

SQL SERVER DATABASE PROCS

Create DATABASE AuthenticationDB

USE AuthenticationDB

SELECT \* FROM dbo.Register;

SELECT \* FROM dbo.RefreshToken;

---------- delete from dbo.Register

----- delete from dbo.RefreshToken

drop table Register

CREATE TABLE Register (

UserID UNIQUEIDENTIFIER PRIMARY KEY NOT NULL DEFAULT NEWID(),

FirstName VARCHAR(255),

LastName VARCHAR(255),

UserName VARCHAR(255),

Email VARCHAR(255),

Password VARCHAR(255),

DateOfBirth DATE,

ZipCode VARCHAR(255),

MobileNumber VARCHAR(255),

StoreID INT,

Token VARCHAR(1000),

RefreshToken VARCHAR(500),

Role VARCHAR(255)

);

CREATE TABLE RefreshToken(

Id int primary key identity(1,1),

Token VARCHAR(1000),

RefreshToken VARCHAR(500),

RefreshTokenExpires DATETIME,

UserID VARCHAR(5000)

)

SELECT \* FROM dbo.Register;

SELECT \* FROM dbo.RefreshToken;

---------- delete from dbo.Register

----- delete from dbo.RefreshToken

CREATE PROCEDURE User\_Register

(

@FirstName VARCHAR(255),

@LastName VARCHAR(255),

@UserName VARCHAR(255),

@Email VARCHAR(255),

@Password VARCHAR(255),

@DateOfBirth DATE,

@ZipCode VARCHAR(255),

@MobileNumber VARCHAR(255),

@StoreID INT,

@Token VARCHAR(1000),

@RefreshToken VARCHAR(500),

@Role VARCHAR(255)

)

AS

BEGIN

INSERT INTO dbo.Register(FirstName,LastName,UserName,Email,Password,DateOfBirth,ZipCode,MobileNumber,StoreID,Token,RefreshToken,Role)

VALUES(@FirstName,@LastName,@UserName,@Email,@Password,@DateOfBirth,@ZipCode,@MobileNumber,@StoreID,@Token,@RefreshToken,@Role)

END

EXEC User\_Register @FirstName='saikumar',@LastName='pusam',@UserName='pusamsaikumar302@gmail.com',@Email='pusamsaikumar302@gmail.com',@Password='woruqpwre',@DateOfBirth='09-11-1994',@ZipCode='507134',@MobileNumber='9979608677',@StoreID=1,@Token='',@RefreshToken='',@Role='Admin'

CREATE PROCEDURE Check\_UserName (

@UserName VARCHAR(255)

)

AS

BEGIN

(SELECT COUNT(1) FROM dbo.Register WHERE UserName=@UserName)

END

EXEC Check\_UserName @UserName='pusamsaikumar302@gmail.com'

CREATE PROCEDURE Get\_UserName(

@UserName VARCHAR(255)

)

AS

BEGIN

SELECT UserName from dbo.Register WHERE UserName = @UserName

END

EXEC Get\_UserName @UserName='pusamsaikumar302@gmail.com'

CREATE PROCEDURE GetDetailsByUserName(

@UserName varchar(255)

)

AS

BEGIN

SELECT \* FROM dbo.Register WHERE UserName=@UserName

END

EXEC GetDetailsByUserName @UserName='pusamsaikumar302@gmail.com'

CREATE PROC Check\_UserName\_Password(

@UserName VARCHAR(255),

@Password VARCHAR(255)

)

AS

BEGIN

SELECT COUNT(1) From dbo.Register WHERE UserName=@UserName AND Password = @Password

END

EXEC Check\_UserName\_Password @UserName='vinay@gmail.com',@Password='dmluYXkxMjM='

SELECT \* FROM dbo.Register;

--- drop table RefreshToken

--CREATE TABLE RefreshToken(

--Id int primary key identity(1,1),

--Token VARCHAR(1000),

--RefreshToken VARCHAR(500),

--RefreshTokenExpires DATETIME,

--UserID VARCHAR(5000)

--)

CREATE TABLE RefreshToken(

Id int primary key identity(1,1),

Token VARCHAR(1000),

RefreshToken VARCHAR(500),

UserID VARCHAR(5000)

)

select \* from dbo.RefreshToken;

---- drop proc SaveTokens

--ALTER PROC SaveTokens(

-- @Token VARCHAR(1000),

-- @RefreshToken VARCHAR(500),

-- -@RefreshTokenExpires DATETIME;

--)

-- AS

-- BEGIN

-- INSERT INTO dbo.RefreshToken(Token,RefreshToken,RefreshTokenExpires) VALUES(@Token,@RefreshToken,@RefreshTokenExpires)

-- END

CREATE PROC SaveTokens(

@Token VARCHAR(1000),

@RefreshToken VARCHAR(500)

)

AS

BEGIN

INSERT INTO dbo.RefreshToken(Token,RefreshToken) VALUES(@Token,@RefreshToken)

END

EXEC SaveTokens @Token='',@RefreshToken=''

DROP PROCEDURE RegisterUser

--CREATE PROCEDURE RegisterUser

-- @FirstName VARCHAR(255),

-- @LastName VARCHAR(255),

-- @UserName VARCHAR(255),

-- @Email VARCHAR(255),

-- @Password VARCHAR(255),

-- @DateOfBirth DATE,

-- @ZipCode VARCHAR(255),

-- @MobileNumber VARCHAR(255),

-- @StoreID INT,

-- @Token VARCHAR(1000),

-- @RefreshToken VARCHAR(255),

-- @Role VARCHAR(255),

-- @RefreshTokenExpires DATETIME

--AS

--BEGIN

-- -- Start a transaction

-- BEGIN TRANSACTION;

-- BEGIN TRY

-- DECLARE @NewUserIDTable TABLE (UserID UNIQUEIDENTIFIER);

-- -- Insert into Register table and capture the new UserID

-- INSERT INTO Register (

-- FirstName,

-- LastName,

-- UserName,

-- Email,

-- Password,

-- DateOfBirth,

-- ZipCode,

-- MobileNumber,

-- StoreID,

-- Token,

-- RefreshToken,

-- Role

-- )

-- OUTPUT INSERTED.UserID INTO @NewUserIDTable

-- VALUES (

-- @FirstName,

-- @LastName,

-- @UserName,

-- @Email,

-- @Password,

-- @DateOfBirth,

-- @ZipCode,

-- @MobileNumber,

-- @StoreID,

-- @Token,

-- @RefreshToken,

-- @Role

-- );

-- DECLARE @NewUserID UNIQUEIDENTIFIER;

-- SELECT @NewUserID = UserID FROM @NewUserIDTable;

-- -- Insert into RefreshToken table

-- INSERT INTO RefreshToken (

-- Token,

-- RefreshToken,

-- RefreshTokenExpires,

-- UserID

-- )

-- VALUES (

-- @Token,

-- @RefreshToken,

-- @RefreshTokenExpires,

-- @NewUserID

-- );

-- -- Commit the transaction

-- COMMIT TRANSACTION;

-- END TRY

-- BEGIN CATCH

-- -- Rollback the transaction in case of error

-- ROLLBACK TRANSACTION;

-- -- Raise the error to the caller

-- THROW;

-- END CATCH

--END;

CREATE PROCEDURE RegisterUser

@FirstName VARCHAR(255),

@LastName VARCHAR(255),

@UserName VARCHAR(255),

@Email VARCHAR(255),

@Password VARCHAR(255),

@DateOfBirth DATE,

@ZipCode VARCHAR(255),

@MobileNumber VARCHAR(255),

@StoreID INT,

@Token VARCHAR(1000),

@RefreshToken VARCHAR(500),

@Role VARCHAR(255)

AS

BEGIN

-- Start a transaction

BEGIN TRANSACTION;

BEGIN TRY

DECLARE @NewUserIDTable TABLE (UserID UNIQUEIDENTIFIER);

-- Insert into Register table and capture the new UserID

INSERT INTO Register (

FirstName,

LastName,

UserName,

Email,

Password,

DateOfBirth,

ZipCode,

MobileNumber,

StoreID,

Token,

RefreshToken,

Role

)

OUTPUT INSERTED.UserID INTO @NewUserIDTable

VALUES (

@FirstName,

@LastName,

@UserName,

@Email,

@Password,

@DateOfBirth,

@ZipCode,

@MobileNumber,

@StoreID,

@Token,

@RefreshToken,

@Role

);

DECLARE @NewUserID UNIQUEIDENTIFIER;

SELECT @NewUserID = UserID FROM @NewUserIDTable;

-- Insert into RefreshToken table

INSERT INTO RefreshToken (

Token,

RefreshToken,

UserID

)

VALUES (

@Token,

@RefreshToken,

@NewUserID

);

-- Commit the transaction

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

-- Rollback the transaction in case of error

ROLLBACK TRANSACTION;

-- Raise the error to the caller

THROW;

END CATCH

END;

EXEC RegisterUser

@FirstName = 'John',

@LastName = 'Doe',

@UserName = 'johndoe',

@Email = 'john.doe@example.com',

@Password = 'password123',

@DateOfBirth = '1990-01-01',

@ZipCode = '12345',

@MobileNumber = '555-555-5555',

@StoreID = 1,

@Token = 'sampleToken',

@RefreshToken = 'sampleRefreshToken',

@Role = 'User'

-----DROP PROCEDURE UpdateTokens

-- CREATE PROCEDURE UpdateTokens

-- @UserID UNIQUEIDENTIFIER,

-- @Token VARCHAR(1000),

-- @RefreshToken VARCHAR(255),

-- @RefreshTokenExpires DATETIME

--AS

--BEGIN

-- -- Start a transaction

-- BEGIN TRANSACTION;

-- BEGIN TRY

-- -- Update the Register table

-- UPDATE Register

-- SET

-- Token = @Token,

-- RefreshToken = @RefreshToken

-- WHERE

-- UserID = @UserID;

-- -- Update the RefreshToken table

-- UPDATE RefreshToken

-- SET

-- Token = @Token,

-- RefreshToken = @RefreshToken,

-- RefreshTokenExpires = @RefreshTokenExpires

-- WHERE

-- UserID = @UserID;

-- -- Commit the transaction

-- COMMIT TRANSACTION;

-- END TRY

-- BEGIN CATCH

-- -- Rollback the transaction in case of error

-- ROLLBACK TRANSACTION;

-- -- Raise the error to the caller

-- THROW;

-- END CATCH

--END;

CREATE PROCEDURE UpdateTokens

@UserID UNIQUEIDENTIFIER,

@Token VARCHAR(1000),

@RefreshToken VARCHAR(500)

AS

BEGIN

-- Start a transaction

BEGIN TRANSACTION;

BEGIN TRY

-- Update the Register table

UPDATE Register

SET

Token = @Token,

RefreshToken = @RefreshToken

WHERE

UserID = @UserID;

-- Update the RefreshToken table

UPDATE RefreshToken

SET

Token = @Token,

RefreshToken = @RefreshToken

WHERE

UserID = @UserID;

-- Commit the transaction

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

-- Rollback the transaction in case of error

ROLLBACK TRANSACTION;

-- Raise the error to the caller

THROW;

END CATCH

END;

EXEC UpdateTokens

@UserID = 'FAFE4DD1-E9B3-4741-A975-D0FEB11B5A6D',

@Token = 'newTokenValue',

@RefreshToken = 'newRefreshTokenValue'

ALTER PROC GetTokens(

@UserID UNIQUEIDENTIFIER

)

AS

BEGIN

SELECT \* FROM dbo.RefreshToken WHERE UserId = @UserID ;

--select \* from dbo.Register WHERE UserId = @UserID ;

END

EXEC GetTokens @UserID='C189D37F-5680-4EB5-AF22-E7CF0948A08E'

select \* from Register;

EXEC Check\_UserName @UserName='saikumar@gmail.com'

EXEC Get\_UserName @UserName='saikumar@gmail.com'

ALTER PROCEDURE Get\_UserByUserName

@UserName NVARCHAR(255)

AS

BEGIN

SET NOCOUNT ON;

SELECT UserID,FirstName,LastName,UserName,Email,Password,DateOfBirth,ZipCode,MobileNumber,StoreID,Token,RefreshToken,Role

FROM dbo.Register

WHERE UserName = @UserName;

END

exec Get\_UserByUserName @UserName='vinay@gmail.com'

exec GetDetailsByUserName @UserName='vinay@gmail.com'

SELECT COUNT(1) FROM RefreshToken WHERE Token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..O4HFKJnXgRiVH9-Jnmyo4ElLwjNNaQVimO04VNOoRN8'

SELECT \* FROM dbo.RefreshToken WHERE Token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..O4HFKJnXgRiVH9-Jnmyo4ElLwjNNaQVimO04VNOoRN8'

CREATE PROC ValidateDBToken(

@Token VARCHAR(1000)

)

AS

BEGIN

SELECT COUNT(1) FROM dbo.RefreshToken WHERE Token = @Token

END

EXEC ValidateDBToken @Token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..O4HFKJnXgRiVH9-Jnmyo4ElLwjNNaQVimO04VNOoRN8'

CREATE PROC GetDBTokensByToken(

@Token VARCHAR(1000)

)

AS

BEGIN

SELECT \* FROM dbo.RefreshToken WHERE Token = @Token

END

EXEC GetDBTokensByToken @Token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..O4HFKJnXgRiVH9-Jnmyo4ElLwjNNaQVimO04VNOoRN8'

CREATE PROC GetDBTokensByTokens(

@Token VARCHAR(1000)

)

AS

BEGIN

SELECT \* FROM dbo.Register WHERE Token = @Token

END

EXEC GetDBTokensByTokens @Token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..O4HFKJnXgRiVH9-Jnmyo4ElLwjNNaQVimO04VNOoRN8'