CUSTOME ADO.NET TO CONNECT SQL SERVER DB WITH IN DATA ACCESS LAYER

Updated AuthRepo.cs:

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;

private readonly SqlHelpers \_sqlHelpers;

public AuthRepo(

IOptions<ConnectionStrings> options,

Helpers helpers,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor

)

{

\_options = options;

\_helpers = helpers;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

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

}

public async Task<RegisterResponse> GetUserDetails(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

try

{

var storedProcName = "GetDetailsByUserName";

var userNameParameter = new SqlParameter[]

{

new SqlParameter("@UserName",userName)

};

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

if(user != null)

{

response.Register = new Register();

response.Register = new Register

{

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

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

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

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

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

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

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

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

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

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

};

response.StatusCode = 200;

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

}

response.StatusCode = 400;

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

}

catch(Exception ex)

{

response.StatusCode = 500;

response.StatusMessage = "Something went wrong.";

}

return response;

}

public async Task<RegisterResponse> GetTokensFromDB(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

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

try

{

var storedProcName = "GetDetailsByUserName";

var userNameParameter = new SqlParameter[]

{

new SqlParameter("@UserName",userName)

};

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

if(user != null)

{

response.Register = new Register();

response.Register = new Register

{

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

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

};

response.StatusCode = 200;

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

}

response.StatusCode = 400;

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

}

catch (Exception ex)

{

response.StatusCode = 500;

response.StatusMessage = "Something went wrong.";

}

return response;

}

public async Task<RegisterResponse> SignUp(Register register)

{

RegisterResponse response = new RegisterResponse();

try

{

var user = await GetUserDetails(register.UserName);

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

{

response.StatusCode = 400;

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

response.Register = null;

return response;

}

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

string refreshtoken = \_helpers.GenerateRefreshToken();

var storedProcName = "RegisterUser";

var parameters = new SqlParameter[]

{

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

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

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

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

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

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

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

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

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

new SqlParameter("@Token", token),

new SqlParameter("@RefreshToken", refreshtoken),

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

};

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

if (rowAffected > 0)

{

response.Register = new Register();

response.Register = register;

response.StatusCode = 200;

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

return response;

}

response.StatusCode = 400;

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

}

catch (Exception ex)

{

response.StatusCode = 500;

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

response.Register = null;

}

return response;

}

public async Task<LoginResponse> Login(LoginModel model)

{

LoginResponse response = null;

response= new LoginResponse();

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

try

{

var storedProcName = "Check\_UserName\_Password";

int existedUserNamePassword = 0;

//using (var cmd = \_sqlHelpers.CreateStoredProcedureCommand(storedProcName))

//{

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

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

// existedUserNamePassword = await \_sqlHelpers.ExecuteScalarInt(cmd);

//}

var parameters = new SqlParameter[]

{

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

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

};

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

if (existedUserNamePassword > 0)

{

var getUser = await GetUserDetails(model.UserName);

var user = await GetTokensFromDB(model.UserName);

response.TokenModel = new TokenModel();

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

{

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

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

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

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

response.StatusCode = 200;

response.StatusMessage = "User loggedin successfully.";

return response;

}

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

response.StatusCode = 200;

response.StatusMessage = "User loggedin successfully.";

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

}

else

{

response.StatusCode = 400;

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

}

}

catch(Exception ex)

{

response.StatusCode = 500;

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

}

return response;

}

public async Task<TokenResponse> Refresh(TokenModel model)

{

TokenResponse tokenResponse = null;

tokenResponse = new TokenResponse();

try

{

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

string userName = principal.Identity.Name;

var user = await GetUserDetails(userName);

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

{

tokenResponse.StatusCode = 400;

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

tokenResponse.TokenModel = null;

}

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

var newRefreshToken = \_helpers.GenerateRefreshToken();

tokenResponse.TokenModel = new TokenModel();

tokenResponse.TokenModel.Token = newAccessToken;

tokenResponse.TokenModel.RefreshToken = newRefreshToken;

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

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

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

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

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

tokenResponse.StatusCode = 200;

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

}

catch (Exception ex)

{

tokenResponse.StatusCode = 500;

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

}

return tokenResponse;

}

public async Task<TokenResponse> GetTokenRefreshTokenById(string userId)

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

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

try

{

var storedProcName = "GetTokens";

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

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

if(user != null)

{

tokenResponse.TokenModel = new TokenModel

{

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

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

};

tokenResponse.StatusCode = 200;

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

}

}

catch (Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Something went wrong.";

}

return tokenResponse;

}

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

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

var storedProcName = "UpdateTokens";

var parameters = new SqlParameter[]

{

new SqlParameter("@UserID", userId),

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

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

};

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

try

{

if(rowsAffected > 0)

{

tokenResponse.StatusCode = 200;

tokenResponse.StatusMessage = "Updated token successfully.";

}

else

{

tokenResponse.StatusCode = 400;

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

}

}

catch(Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Something went wrong.";

}

return tokenResponse;

}

}

}

Previous AuthRepo.cs:

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;

private readonly SqlHelpers \_sqlHelpers;

public AuthRepo(

IOptions<ConnectionStrings> options,

Helpers helpers,

IOptions<JWT> jwt,

IHttpContextAccessor httpContextAccessor

)

{

\_options = options;

\_helpers = helpers;

\_jwt = jwt;

\_httpContextAccessor = httpContextAccessor;

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

}

public async Task<RegisterResponse> GetUserDetails(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

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

try

{

var storedProcName = "GetDetailsByUserName";

var userNameParameter = new SqlParameter[]

{

new SqlParameter("@UserName",userName)

};

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

if(user != null)

{

response.Register = new Register();

response.Register = new Register

{

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

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

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

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

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

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

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

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

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

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

};

response.StatusCode = 200;

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

}

response.StatusCode = 400;

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

//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()

// };

// }

// }

// }

//}

}

catch(Exception ex)

{

response.StatusCode = 500;

response.StatusMessage = "Something went wrong.";

}

return response;

}

public async Task<RegisterResponse> GetTokensFromDB(string userName)

{

RegisterResponse response = null;

response = new RegisterResponse();

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

try

{

var storedProcName = "GetDetailsByUserName";

var userNameParameter = new SqlParameter[]

{

new SqlParameter("@UserName",userName)

};

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

if(user != null)

{

response.Register = new Register();

response.Register = new Register

{

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

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

};

response.StatusCode = 200;

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

}

response.StatusCode = 400;

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

//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()

// };

// }

// }

// }

//}

}

catch (Exception ex)

{

response.StatusCode = 500;

response.StatusMessage = "Something went wrong.";

}

return response;

}

public async Task<RegisterResponse> SignUp(Register register)

{

RegisterResponse response = new RegisterResponse();

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

try

{

var user = await GetUserDetails(register.UserName);

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

{

response.StatusCode = 400;

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

response.Register = null;

return response;

}

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

string refreshtoken = \_helpers.GenerateRefreshToken();

var storedProcName = "RegisterUser";

var parameters = new SqlParameter[]

{

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

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

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

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

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

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

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

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

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

new SqlParameter("@Token", token),

new SqlParameter("@RefreshToken", refreshtoken),

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

};

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

if (rowAffected > 0)

{

response.Register = new Register();

response.Register = register;

response.StatusCode = 200;

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

return response;

}

response.StatusCode = 400;

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

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

//{

// await conn.OpenAsync();

// var user = await GetUserDetails(register.UserName);

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

// {

// response.StatusCode = 400;

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

// response.Register = null;

// await conn.CloseAsync();

// return response;

// }

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

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

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

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

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

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

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

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

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

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

// cmd.Parameters.AddWithValue("@StoreID", register.StoreID);'Object reference not set to an

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

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

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

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

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

// await cmd.ExecuteNonQueryAsync();

// await conn.CloseAsync();

//}

}

catch (Exception ex)

{

response.StatusCode = 500;

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

response.Register = null;

}

return response;

}

public async Task<LoginResponse> Login(LoginModel model)

{

LoginResponse response = null;

response= new LoginResponse();

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

try

{

using(SqlConnection conn = new SqlConnection(connectionString))

{

await conn.OpenAsync();

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

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

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

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

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

await conn.CloseAsync();

if (variable > 0 )

{

var getUser = await GetUserDetails(model.UserName);

var user = await GetTokensFromDB(model.UserName);

response.TokenModel = new TokenModel();

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

{

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

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

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

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

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

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

response.StatusCode = 200;

response.StatusMessage = "User loggedin successfully.";

return response;

}

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

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

response.StatusCode = 200;

response.StatusMessage= "User loggedin successfully.";

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

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

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

}

else

{

response.StatusCode = 400;

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

}

}

}catch(Exception ex)

{

response.StatusCode = 500;

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

}

return response;

}

public async Task<TokenResponse> Refresh(TokenModel model)

{

TokenResponse tokenResponse = null;

tokenResponse = new TokenResponse();

try

{

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

string userName = principal.Identity.Name;

var user = await GetUserDetails(userName);

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

{

tokenResponse.StatusCode = 400;

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

tokenResponse.TokenModel = null;

}

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

var newRefreshToken = \_helpers.GenerateRefreshToken();

tokenResponse.TokenModel = new TokenModel();

tokenResponse.TokenModel.Token = newAccessToken;

tokenResponse.TokenModel.RefreshToken = newRefreshToken;

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

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

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

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

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

tokenResponse.StatusCode = 200;

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

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

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

//{

// await conn.OpenAsync();

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

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

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

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

// await conn.CloseAsync();

// var user = await GetUserDetails(existUser);

// if (existUser != userName)

// {

// tokenResponse.StatusCode = 400;

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

// }

// else

// {

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

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

// tokenResponse.TokenModel.Token = newAccessToken;

// tokenResponse.TokenModel.RefreshToken = newRefreshToken;

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

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

// //{

// // HttpOnly = true,

// // SameSite = SameSiteMode.Strict

// //});

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

// //{

// // HttpOnly = true,

// // SameSite = SameSiteMode.Strict

// //});

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

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

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

// tokenResponse.StatusCode = 200;

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

// }

//}

}

catch (Exception ex)

{

tokenResponse.StatusCode = 500;

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

}

return tokenResponse;

}

public async Task<TokenResponse> GetTokenRefreshTokenById(string userId)

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

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

try

{

var storedProcName = "GetTokens";

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

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

if(user != null)

{

tokenResponse.TokenModel = new TokenModel

{

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

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

};

tokenResponse.StatusCode = 200;

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

}

//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"])

// };

// }

// }

// }

//}

}

catch (Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Something went wrong.";

}

return tokenResponse;

}

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

{

TokenResponse tokenResponse = new TokenResponse();

tokenResponse.TokenModel = new TokenModel();

var storedProcName = "UpdateTokens";

var parameters = new SqlParameter[]

{

new SqlParameter("@UserID", userId),

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

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

};

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

try

{

if(rowsAffected > 0)

{

tokenResponse.StatusCode = 200;

tokenResponse.StatusMessage = "Updated token successfully.";

}

else

{

tokenResponse.StatusCode = 400;

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

}

}

catch(Exception ex)

{

tokenResponse.StatusCode = 500;

tokenResponse.StatusMessage = "Something went wrong.";

}

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

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

// {

// await conn.OpenAsync();

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

// cmd.CommandType = CommandType.StoredProcedure;

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

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

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

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

// int rowsAffected = await cmd.ExecuteNonQueryAsync();

// await conn.CloseAsync();

// try

// {

// if (rowsAffected > 0)

// {

// tokenResponse.StatusCode = 200;

// tokenResponse.StatusMessage = "Updated token successfully.";

// }

// else

// {

// tokenResponse.StatusCode = 404;

// tokenResponse.StatusMessage = "No tokens updated.";

// }

// }

// catch (SqlException ex)

// {

// tokenResponse.StatusCode = 500;

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

// }

// }

return tokenResponse;

}

}

}

Sqlhelpers.cs:

using Dapper;

using Microsoft.AspNetCore.Mvc.TagHelpers.Cache;

using System.Data;

using System.Data.SqlClient;

using System.Runtime.InteropServices;

namespace JWTRoleAuthentication.CommonLayer.Models

{

//public class SqlHelpers : IDisposable

public class SqlHelpers : IAsyncDisposable

{

private readonly string \_connectionString;

private SqlConnection \_connection;

public SqlHelpers(string connectionString)

{

\_connectionString = connectionString;

\_connection = new SqlConnection(connectionString);

}

//public void Dispose()

//{

// if (\_connection != null)

// {

// if(\_connection.State != System.Data.ConnectionState.Closed)

// {

// \_connection.Close();

// }

// \_connection.Dispose();

// \_connection = null;

// }

//}

public async ValueTask DisposeAsync()

{

if (\_connection != null)

{

if (\_connection.State != System.Data.ConnectionState.Closed)

{

await \_connection.CloseAsync();

}

\_connection.Dispose();

\_connection = null;

}

}

public async Task EnsureConnection()

{

if (\_connection != null)

{

if(\_connection.State != ConnectionState.Open)

{

await \_connection.OpenAsync();

}

}

}

// sql command:

public SqlCommand CreateCommand(string commandText, CommandType commandType = CommandType.StoredProcedure)

{

var command = \_connection.CreateCommand();

command.CommandText = commandText;

command.CommandType = commandType;

return command;

}

// Create StoredProcedure command:

//public SqlCommand CreateStoredProcedureCommand(string storedProcedureName)

//{

// return CreateCommand(storedProcedureName,CommandType.StoredProcedure);

//}

public SqlCommand CreateStoredProcedureCommand(string storedProcedureName)

{

//var command = \_connection.CreateCommand();

//command.CommandText = storedProcedureName;

//command.CommandType = CommandType.StoredProcedure;

//return command;

var command = new SqlCommand(storedProcedureName, \_connection);

command.CommandType = CommandType.StoredProcedure;

return command;

}

// SqlDataAdapter and SqlDataReader

// To Read Table Data using SqlDataAdapter and SqlDataReader for

public async Task<DataTable> ExecuteDataTableAsync(SqlCommand command)

{

DataTable dataTable = new DataTable();

await EnsureConnection();

using(var adapter = new SqlDataAdapter(command))

{

await Task.Run(() => adapter.Fill(dataTable));

return dataTable;

}

}

public async Task<DataTable> ReadDataTable(SqlCommand command)

{

DataTable dataTable = new DataTable();

await EnsureConnection();

using(var reader = await ExecuteReader(command))

{

dataTable.Load(reader);

return dataTable;

}

}

public async Task<DataSet> GetMultipleTablesData(string[] storedProcNames, SqlParameter[] parameters = null)

{

await EnsureConnection();

var dataSet = new DataSet();

var tasks = storedProcNames.Select(async procName =>

{

using (var cmd = CreateStoredProcedureCommand(procName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

using (var adapter = new SqlDataAdapter(cmd))

{

var dataTable = new DataTable();

await Task.Run(() => adapter.Fill(dataTable));

dataSet.Tables.Add(dataTable);

}

}

});

await Task.WhenAll(tasks);

return dataSet;

}

// To Read database records using SqlDataReader: for read single row and multiple rows and datarowcollections.

// use following methods:

public async Task<SqlDataReader> ExecuteReader(SqlCommand command)

{

// await EnsureConnection();

//return await command.ExecuteReaderAsync(CommandBehavior.CloseConnection);

return await command.ExecuteReaderAsync();

}

public async Task<DataRow> GetSingleRow(string storedProcedureName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcedureName))

{

command.Parameters.AddRange(parameters);

using (var reader = await ExecuteReader(command))

{

var dataTable = new DataTable();

dataTable.Load(reader);

if (dataTable.Rows.Count > 0)

{

return dataTable.Rows[0];

}

}

}

return null;

}

public async Task<List<DataRow>> GetMultipleRows(string storedProcedureName, SqlParameter[] parameters)

{

await EnsureConnection();

var result = new List<DataRow>();

using (var command = CreateStoredProcedureCommand(storedProcedureName))

{

if(parameters != null)

{

command.Parameters.AddRange(parameters);

}

using (var reader = await ExecuteReader(command))

{

var dataTable = new DataTable();

dataTable.Load(reader);

if(dataTable.Rows.Count > 0)

{

result = dataTable.AsEnumerable().Select((row) => row).ToList();

}

}

}

return result;

}

public async Task<DataRowCollection> GetMultipleDataRows(string storedProcedureName, SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcedureName))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

using (var reader = await ExecuteReader(command))

{

var dataTable = new DataTable();

dataTable.Load(reader);

if(dataTable.Rows.Count > 0)

{

return dataTable.Rows;

}

}

}

return null;

}

// ExecuteNonQuery: for Update and Insert Data Records

// using following methods:

public async Task<int> ExecuteNonQueryAsync(SqlCommand command)

{

await EnsureConnection();

return await command.ExecuteNonQueryAsync();

}

public async Task<int> InsertTable(string storedProcName,params SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcName))

{

if(parameters != null)

{

command.Parameters.AddRange(parameters);

}

int rowAffected = await ExecuteNonQueryAsync(command);

return rowAffected;

}

}

public async Task<int> UpdateTable(string storedProcName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var command = CreateStoredProcedureCommand(storedProcName))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

int rowAffected = await ExecuteNonQueryAsync(command);

return rowAffected;

}

}

// ExecuteScalar methods:

//public async Task<object> ExecuteScalarAsync(SqlCommand command)

//{

// await EnsureConnection();

// return await command.ExecuteScalarAsync();

//}

public async Task<String> ExecuteScalarString(SqlCommand command)

{

await EnsureConnection();

return (string)await command.ExecuteScalarAsync();

}

public async Task<int> ExecuteScalarInt(SqlCommand command)

{

await EnsureConnection();

return (int)await command.ExecuteScalarAsync();

}

public async Task<int> ExecuteIntScalar(string storeProcName, params SqlParameter[] parameters)

{

await EnsureConnection();

using(var cmd = CreateStoredProcedureCommand(storeProcName))

{

if(parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

int variable = (int)await ExecuteScalarInt(cmd);

return variable;

}

}

public async Task<string> ExecuteStringScalar(string storeProcName, params SqlParameter[] parameters)

{

await EnsureConnection();

using (var cmd = CreateStoredProcedureCommand(storeProcName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

string variable = (string)await ExecuteScalarString(cmd);

return variable;

}

}

// IDataReader Methods:

// use following methods:

public async Task<IDataReader> ExecuteIDataReader(SqlCommand command)

{

await EnsureConnection();

return await command.ExecuteReaderAsync(CommandBehavior.CloseConnection);

}

public async Task<IDataReader> IDataReaderAsync(string storeProcName, params SqlParameter[] parameters )

{

await EnsureConnection();

using(var cmd = CreateStoredProcedureCommand(storeProcName))

{

if (parameters !=null)

{

cmd.Parameters.AddRange(parameters);

}

return await ExecuteIDataReader(cmd);

}

}

public async Task<DataRowCollection> IDataReaderMultiRow(string storeProcName, params SqlParameter[] parameters)

{

using(IDataReader reader = await IDataReaderAsync(storeProcName, parameters))

{

var dataTable = new DataTable();

dataTable.Load(reader);

return dataTable.Rows;

}

}

public async Task<List<DataRow>> IDataReaderMultiRows(string storeProcName, params SqlParameter[] parameters)

{

using (IDataReader reader = await IDataReaderAsync(storeProcName, parameters))

{

var dataTable = new DataTable();

dataTable.Load(reader);

return dataTable.Rows.Cast<DataRow>().ToList();

}

}

public async Task<List<IDataRecord>> IDataRecordsData(string storedProcName, params SqlParameter[] parameters)

{

var records = new List<IDataRecord>();

await EnsureConnection();

using (var cmd = CreateStoredProcedureCommand(storedProcName))

{

if (parameters != null)

{

cmd.Parameters.AddRange(parameters);

}

using (IDataReader reader = await ExecuteIDataReader(cmd))

{

while (reader.Read())

{

records.Add(reader);

}

}

}

return records;

}

// DAPPER :

// Dynamic paramter : INSERT , UPDATE , DELETE

public async Task<int> DapperExecuteNonQueryAsync(string storedProcedure, DynamicParameters parameters)

{

using (var connection = \_connection)

{

await EnsureConnection();

int rowsAffected = await connection.ExecuteAsync(storedProcedure, parameters, commandType: CommandType.StoredProcedure);

return rowsAffected;

}

}

// USING: Get details by Id, or other values: Single row value

//

public async Task<T> DapperQuerySingleRow<T>(string storedProcName, DynamicParameters parameters)

{

using (var connection = \_connection)

{

await EnsureConnection();

var resultData = await \_connection.QuerySingleOrDefaultAsync<T>(storedProcName, parameters, commandType: CommandType.StoredProcedure);

return resultData;

}

}

// USING : Get multiple rows data

//public async Task<IEnumerable<T>> DapperListTableValues<T>(string storedProcName, DynamicParameters parameters)

public async Task<List<T>> DapperMultipleRows<T>(string storedProcName, DynamicParameters parameters)

{

using (var connection = \_connection)

{

await EnsureConnection();

var resultData = await \_connection.QueryAsync<T>(storedProcName,parameters,commandType: CommandType.StoredProcedure);

return resultData.ToList();

}

}

// USING : Get Single Value that may be int or string.

public async Task<T> DapperGetSingleValue<T>(string storedProcName, DynamicParameters parameters)

{

using (var connection = \_connection)

{

await EnsureConnection();

var resutl = await \_connection.QueryFirstOrDefaultAsync<T>(storedProcName, parameters, commandType: CommandType.StoredProcedure);

return resutl;

}

}

// using dataset:

//public async Task<DataSet> DapperMultiDataTableResult(string[] storedProcNames, DynamicParameters parameters)

//{

// var dataset = new DataSet();

// using (var connection = \_connection)

// {

// await EnsureConnection();

// var tasks = storedProcNames.Select(async (procName) =>

// {

// using (var multisetResult = await \_connection.QueryMultipleAsync(procName, parameters, commandType: CommandType.StoredProcedure))

// {

// var dataTable = new DataTable();

// var result = (await Task.Run(() => multisetResult.Read().ToList())).Cast<object>().ToList();

// if (result.Any())

// {

// var properties = result.GetType().GetProperties();

// dataTable.Columns.AddRange(properties.Select(prop => new DataColumn(prop.Name, prop.PropertyType)).ToArray());

// var rows = result.Select(row => properties.Select(prop => prop.GetValue(row)).ToArray()).ToArray();

// rows.Select(row => dataTable.Rows.Add(row)).ToList();

// dataset.Tables.Add(dataTable);

// }

// }

// });

// await Task.WhenAll(tasks);

// }

// return dataset;

//}

}

}

CustomerDal.cs:

using JWTRoleAuthentication.CommonLayer.Models;

using Microsoft.AspNetCore.Mvc.TagHelpers.Cache;

using Microsoft.Extensions.Options;

using System.Data;

using System.Data.SqlClient;

using System.Runtime.InteropServices;

namespace JWTRoleAuthentication.JWTDAL

{

public class CustomerDAL

{

private readonly IOptions<ConnectionStrings> \_options;

private readonly SqlHelpers \_sqlHelpers;

public CustomerDAL(IOptions<ConnectionStrings> options)

{

\_options = options;

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

}

//public async Task<DataRow> GetCustomerDetailsById(int id)

//{

// var storedProcName = "GetCustomerById";

// return await \_sqlHelpers.GetSingleRow(storedProcName, new SqlParameter("@Id", id));

//}

//public async Task<DataSet> GetMultipleTablesData()

//{

// string[] storedProcNames =

// {

// "GetUserDetails",

// "GetCustomers"

// };

// return await \_sqlHelpers.GetMultipleTablesData(storedProcNames, null);

//}

// create mapcustomer model for retrive the data using IDataRecord:

private Task<CustomerModel> MapCustomer(IDataRecord record)

{

var model = new CustomerModel

{

Id = Convert.ToInt32(record["Id"]),

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

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

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

Address = record["Address"].ToString()

};

Console.WriteLine(model);

return Task.FromResult(model);

}

// create map customers list using IDataRecord

private async Task<List<CustomerModel>> MapCustomerList(IDataRecord record)

{

var customerList = new List<CustomerModel>();

customerList.Add(new CustomerModel

{

Id = Convert.ToInt32(record["Id"]),

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

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

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

Address = record["Address"].ToString()

});

return await Task.FromResult(customerList);

}

public async Task<CustomerModel> GetCustomerDetailsById(int id)

{

try

{

var storedProcName = "GetCustomerById";

var result = await \_sqlHelpers.GetSingleRow(storedProcName, new SqlParameter("@Id", id));

var customer = new CustomerModel

{

Id = Convert.ToInt32(result["Id"]),

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

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

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

Address = result["Address"].ToString()

};

// iDataReader

//var iDataReader = await \_sqlHelpers.IDataReaderAsync(storedProcName, new SqlParameter("@Id", id));

//if(iDataReader.Read())

//{

// var model = new CustomerModel

// {

// Id = Convert.ToInt32(iDataReader["Id"]),

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

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

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

// Address = iDataReader["Address"].ToString()

// };

// Console.WriteLine(model);

// // IDataRecord:

// await MapCustomer((IDataRecord)iDataReader);

//}

// using dapper:

var parameters = new DynamicParameters();

parameters.Add("@Id", id, DbType.Int32);

var getCustomerDetails = await \_sqlHelpers.DapperQuerySingleRow<CustomerModel>(storedProcName, parameters);

// return getCustomerDetails;

return customer;

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<DatasetResponse> GetMultipleTablesData()

{

DatasetResponse datasetResponse = null;

datasetResponse = new DatasetResponse();

string[] storedProcNames =

{

"GetUserDetails",

"GetCustomers"

};

var dataset = await \_sqlHelpers.GetMultipleTablesData(storedProcNames, null);

try

{

var userTable = dataset.Tables[0];

var customerTable = dataset.Tables[1];

datasetResponse.users = (from DataRow r in userTable.Rows

select new Register

{

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

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

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

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

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

DateOfBirth = Convert.ToDateTime(r["DateOfBirth"]),

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

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

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

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

}

).ToList();

datasetResponse.Customer = (from DataRow dt in customerTable.Rows select new CustomerModel

{

Id = Convert.ToInt32(dt["Id"]),

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

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

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

Address = dt["Address"].ToString()

}).ToList();

datasetResponse.StatusCode = 200;

datasetResponse.StatusMessage= "OK";

}catch (Exception ex)

{

datasetResponse.StatusCode = 500;

datasetResponse.StatusMessage = "Something went wrong.";

}

return datasetResponse;

}

public async Task<string> GetCustomerEmail(int id)

{

var storedProcName = "GetCustomerEmail";

//using (var cmd = \_sqlHelpers.CreateStoredProcedureCommand(storedProcName))

//{

// cmd.Parameters.AddWithValue("@Id", id);

// string email = await \_sqlHelpers.ExecuteScalarString(cmd);

// return email;

//}

// USING DAPPER:

var parameter = new DynamicParameters();

parameter.Add("@Id",id,DbType.Int32);

var getEmail = await \_sqlHelpers.DapperGetSingleValue<string>(storedProcName,parameter);

return getEmail;

}

public async Task<List<CustomerModel>> GetCustomers()

{

try

{

var customers = new List<CustomerModel>();

var listCustomers = new List<CustomerModel>();

var storedProcName = "GetCustomers";

// using multiple Rows:

//var customerTable = await \_sqlHelpers.GetMultipleRows(storedProcName, null);

var customerTable = await \_sqlHelpers.GetMultipleDataRows(storedProcName, null);

customers = (from DataRow dr in customerTable

select new CustomerModel

{

Id = Convert.ToInt32(dr["Id"]),

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

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

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

Address = dr["Address"].ToString()

}).ToList();

// IDataReader for multiple data rows:

//var iDataReader = await \_sqlHelpers.IDataReaderAsync(storedProcName, null);

//while (iDataReader.Read())

//{

// listCustomers.Add(new CustomerModel

// {

// Id = Convert.ToInt32(iDataReader["Id"]),

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

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

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

// Address = iDataReader["Address"].ToString()

// });

// //await MapCustomerList((IDataRecord)iDataReader);

//}

// var iDataReader = await \_sqlHelpers.IDataReaderMultiRows(storedProcName, null);

//listCustomers = (from DataRow dr in iDataReader

// select new CustomerModel

// {

// Id = Convert.ToInt32(dr["Id"]),

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

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

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

// Address = dr["Address"].ToString()

// }

// ).ToList();

// var IDataRecordsData = await \_sqlHelpers.IDataRecordsData(storedProcName, null);

// IDataRecord for list of customer

// var records = IDataRecordsData.Select(async record => await MapCustomerList(record)).ToList();

//using (var cmd = \_sqlHelpers.CreateStoredProcedureCommand(storedProcName))

//{

// // using adapter:

// var customerTable = await \_sqlHelpers.ExecuteDataTableAsync(cmd);

// // using reader:

// // var customerTable = await \_sqlHelpers.ReadDataTable(cmd);

// customers = (from DataRow dr in customerTable.Rows

// select new CustomerModel

// {

// Id = Convert.ToInt32(dr["Id"]),

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

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

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

// Address = dr["Address"].ToString()

// }).ToList();

//}

// using dapper get customer list data:

var customersList = await \_sqlHelpers.DapperMultipleRows<CustomerModel>(storedProcName,null);

return customersList;

//return customers;

}

catch (Exception ex)

{

throw ex;

}

}

public async Task<CustomerResponse> UpdateCustomer(int id, CustomerModel model)

{

CustomerResponse customerResponse = new CustomerResponse();

try

{

var storedProcName = "UpdateCustomer";

// using sqlparameter:

//var parameters = new SqlParameter[]

//{

// new SqlParameter ("@Id",id),

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

// new SqlParameter("@Email",model.Email),

// new SqlParameter("@MobileNumber",model.MobileNumber),

// new SqlParameter("@Address",model.Address),

//};

// sql

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

// using Dapper

var parameters = new DynamicParameters();

parameters.Add("@Id", id, DbType.Int32);

parameters.Add("@UserName", model.UserName, DbType.String, size: 100);

parameters.Add("@Email", model.Email, DbType.String, size: 100);

parameters.Add("@MobileNumber", model.MobileNumber, DbType.String, size: 100);

parameters.Add("@Address", model.Address, DbType.String, size: 100);

//var rowsAffected = await \_sqlHelpers.DapperExecuteNonQueryAsync(storedProcName, parameters);

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

if (rowsAffected > 0)

{

customerResponse.Customer = new CustomerModel();

customerResponse.Customer.Id = id;

customerResponse.Customer.UserName = model.UserName;

customerResponse.Customer.Email = model.Email;

customerResponse.Customer.MobileNumber = model.MobileNumber;

customerResponse.Customer.Address = model.Address;

customerResponse.StatusCode = 200;

customerResponse.StatusMessage = "Customer has been updated successfully.";

}

}

catch (Exception ex)

{

customerResponse.StatusCode = 500;

customerResponse.StatusMessage = "Something went wrong.";

}

return customerResponse;

}

public async Task<CustomerResponse> AddCustomer(CustomerModel model)

{

CustomerResponse customerResponse = new CustomerResponse();

try

{

var storedProcName = "InsertCustomer";

// USING SQL COMMANDS:

// var parameters = new SqlParameter[]

// {

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

// new SqlParameter("@Email",model.Email),

// new SqlParameter("@MobileNumber",model.MobileNumber),

// new SqlParameter("@Address",model.Address),

// };

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

// USING DAPPER:

var parameters = new DynamicParameters();

parameters.Add("@UserName", model.UserName, DbType.String, size:100);

parameters.Add("@Email", model.Email, DbType.String, size: 100);

parameters.Add("@MobileNumber", model.MobileNumber, DbType.String, size: 100);

parameters.Add("@Address", model.Address, DbType.String, size: 100);

var affectedRow = await \_sqlHelpers.DapperExecuteNonQueryAsync(storedProcName,parameters);

if(affectedRow > 0)

{

customerResponse.Customer = model;

customerResponse.StatusCode = 200;

customerResponse.StatusMessage = "Customer details has been add successfully.";

}

else

{

// customerResponse.Customer = null;

customerResponse.StatusCode = 400;

customerResponse.StatusMessage = "Insert of customer details has been failed.";

}

}catch(Exception ex)

{

customerResponse.StatusCode = 500;

customerResponse.StatusMessage = "Something went wrong. Please try again later.";

}

return customerResponse;

}

Helper.cs: for refreshtoken

using JWTRoleAuthentication.JWTDAL;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using System.Data.SqlClient;

using System.Data;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Security.Cryptography;

using System.Text;

namespace JWTRoleAuthentication.CommonLayer.Models

{

public class Helpers

{

private readonly IOptions<JWT> \_jwt;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IOptions<AppSettings> \_appsettings;

private readonly IConfiguration \_configuration;

private readonly IOptions<ConnectionStrings> \_config;

private readonly IServiceProvider \_serviceProvider;

private readonly SqlHelpers \_sqlHelpers;

public Helpers(

IOptions<JWT> jwt,

IServiceProvider serviceProvider,

IHttpContextAccessor httpContextAccessor,

IOptions<AppSettings> appsettings,

IConfiguration configuration,

IOptions<ConnectionStrings> config

)

{

\_jwt = jwt;

\_serviceProvider = serviceProvider;

\_httpContextAccessor = httpContextAccessor;

\_appsettings = appsettings;

\_configuration = configuration;

\_config = config;

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

}

// encrypted password

public string EncryptedPassword(string password)

{

if (string.IsNullOrEmpty(password))

{

return "";

}

else

{

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

string encryptPassword = Convert.ToBase64String(passwordBase64);

return encryptPassword;

}

}

// decrypted password

public string DecryptedPassword(string password)

{

if (string.IsNullOrEmpty(password))

{

return "";

}

else

{

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

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

return DecryptedPassword;

}

}

// sha256 password:

public string EncryptSha256Password(string password)

{

using var sha256 = SHA256.Create();

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

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

return Convert.ToBase64String(EncryptPasswordstorage);

}

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

//{

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

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

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

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

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

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

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

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

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

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: ValidAudience,

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

// claims: claims,

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

// );

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

//}

//public string GenerateJwtAccessToken(string username)

//{

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

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

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

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

// var authClaims = new List<Claim>

// {

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

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

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

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

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

// };

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

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

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

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

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

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: validAudience,

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

// claims: authClaims,

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

// );

// return tokenHandler.WriteToken(token);

//}

//public JwtSecurityToken GenerateJwtToken(string username)

//{

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

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

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

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

// var authClaims = new List<Claim>

// {

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

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

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

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

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

// };

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

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

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

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

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

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: validAudience,

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

// claims: authClaims,

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

// );

// return token;

//}

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

// {

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

// var authClaims = new List<Claim>

//{

// new Claim(ClaimTypes.Name, userName),

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

// new Claim(ClaimTypes.Role,role.ToString()),

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

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

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

//};

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

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

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

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

// var token = new JwtSecurityToken(

// issuer: validIssuer,

// audience: validAudience,

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

// claims: authClaims,

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

// );

// return tokenHandler.WriteToken(token);

// }

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

{

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

var authClaims = new List<Claim>

{

new Claim(ClaimTypes.Name, userName),

new Claim(ClaimTypes.Email, email),

new Claim(ClaimTypes.Role, role),

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

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

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

};

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

var token = new JwtSecurityToken(

\_jwt.Value.ValidIssuer,

\_jwt.Value.ValidAudience,

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

claims: authClaims,

signingCredentials: new SigningCredentials(key, SecurityAlgorithms.HmacSha256)

);

return tokenHandler.WriteToken(token);

}

//public string GenerateRefreshToken()

//{

// var randomNumber = new byte[640];

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

// generator.GetBytes(randomNumber);

// return Convert.ToBase64String(randomNumber);

//}

public ClaimsPrincipal GetPrincipalFromExpiredToken(string token)

{

var secret = \_jwt.Value.Secret;

var tokenValidationParameters = new TokenValidationParameters

{

ValidateAudience = false,

ValidateIssuer = false,

ValidateIssuerSigningKey = true,

ValidateLifetime = false,

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

};

var tokenHandler = new JwtSecurityTokenHandler();

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

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

throw new SecurityTokenException("Invalid Token");

return principal;

}

// public string GenerateRefreshToken()

//{

// var randomNumber = new byte[32];

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

// {

// rng.GetBytes(randomNumber);

// return Convert.ToBase64String(randomNumber);

// }

//}

public string GenerateRefreshToken()

{

var randomNumber = new byte[32];

using (var randomNum = RandomNumberGenerator.Create())

{

randomNum.GetBytes(randomNumber);

}

var randomValue = Convert.ToBase64String(randomNumber);

var tokenHandler = new JwtSecurityTokenHandler();

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

// Create an array of claims

var claims = new[]

{

new Claim("randomValue", randomValue)

};

var tokenDescriptor = new SecurityTokenDescriptor

{

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

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

Subject = new ClaimsIdentity(claims)

};

var securityToken = tokenHandler.CreateToken(tokenDescriptor);

var refreshToken = tokenHandler.WriteToken(securityToken);

return refreshToken;

}

public bool IsTokenExpired(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

return jwtToken.ValidTo < DateTime.UtcNow;

}

public bool IsRefreshTokenExpired(string refreshToken)

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtRefreshToken = tokenHandler.ReadToken(refreshToken);

return jwtRefreshToken.ValidTo < DateTime.UtcNow;

}

public async Task<int> ValidateDBToken(string token)

{

var storedProcName = "ValidateDBToken";

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

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

if (tokenExist > 0)

{

return 1;

}

return 0;

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

//{

// await conn.OpenAsync();

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

// command.CommandType = CommandType.StoredProcedure;

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

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

// await conn.CloseAsync();

// if(tokenExists > 0)

// {

// return 1;

// }

// return 0;

//}

}

public async Task<TokenModel> GetDBTokenDetailsByToken(string token)

{

var storedProcName = "GetDBTokensByToken";

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

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

if (user != null)

{

return new TokenModel

{

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

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

};

}

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

//{

// await connection.OpenAsync();

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

// command.CommandType = CommandType.StoredProcedure;

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

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

// {

// if (await reader.ReadAsync())

// {

// return new TokenModel

// {

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

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

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

// };

// }

// }

//}

return null;

}

//private bool IsTokenExpired(string token)

//{

// var tokenHandler = new JwtSecurityTokenHandler();

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

// return jwtToken.ValidTo < DateTime.UtcNow;

//}

}

}

UserMiddleware for RefreshToken:

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);

//}

//public async Task Invoke(HttpContext context, IServiceProvider serviceProvider)

//{

// // for Account controller

// //var path = context.Request.Path;

// //if ( path.StartsWithSegments("/api/Account") )

// //{

// // await \_next(context);

// // return;

// //}

// 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 (IsTokenExpired(token) || IsTokenExpired(tokenCookie))

// {

// 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();

// // tokenResponse.TokenModel.Token = newAccessToken;

// //tokenResponse.TokenModel.RefreshToken = newRefreshToken;

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

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

// {

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

// //Secure = true,

// //IsEssential = true,

// //SameSite = SameSiteMode.None

// 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 successully..." });

// 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 async Task Invoke(HttpContext context, IServiceProvider serviceProvider)

{

using var scope = serviceProvider.CreateScope();

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

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

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

int tokenExist = 0;

string dbRefreshToken = null;

string dbToken = null;

if (token != null)

{

tokenExist = await \_helpers.ValidateDBToken(token);

}

if (tokenExist > 0)

{

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

dbRefreshToken = getTokens.RefreshToken;

dbToken = getTokens.Token;

}

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

{

if (IsEnabledUnathourizedRoute(context))

{

await \_next(context);

}

else

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

}

return;

}

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

{

var principal = \_helpers.GetPrincipalFromExpiredToken(token);

string userName = principal.Identity.Name;

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

if (IsRefreshTokenExpired(dbRefreshToken))

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

return;

}

if (IsTokenExpired(token))

{

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

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

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

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

return;

}

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

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

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

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

var newRefreshToken = dbRefreshToken;

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

var tokenModel = new TokenModel

{

Token = newAccessToken,

RefreshToken = newRefreshToken,

};

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

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

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

// await \_next(context);

//return

}

await \_next(context);

}

}

}

}

StartUp.cs:

using JWTRoleAuthentication.CommonLayer.Models;

using JWTRoleAuthentication.JWTBLL;

using JWTRoleAuthentication.JWTDAL;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

namespace JWTRoleAuthentication

{

//public class StartUp

//{

// //private readonly IConfiguration \_configuration;

// public IConfiguration configRoot { get; set; }

// // constructor

// public StartUp(IConfiguration configuration)

// {

// configRoot = configuration;

// }

// // configure services

// public void ConfigureServices(IServiceCollection services)

// {

// // ConfigJsonFiles

// // dependeny injections

// services.AddScoped<IAuthRepo, AuthRepo>();

// services.AddScoped<IAuthService, AuthService>();

// services.AddScoped<ITokenService, TokenService>();

// services.AddScoped<Helpers>();

// // configure

// services.AddOptions();

// services.Configure<AppSettings>(configRoot.GetSection("AppSettings"));

// services.Configure<ConnectionStrings>(configRoot.GetSection("ConnectionStrings"));

// services.Configure<JWT>(configRoot.GetSection("JWT"));

// // login auth

// // wcf service

// // services.AddScoped<IService,ServiceClient>();

// // add error handling exceptions here for method3

// services.AddCors(options =>

// {

// options.AddDefaultPolicy(

// policy =>

// {

// policy.AllowAnyOrigin()

// .AllowAnyHeader()

// .AllowAnyMethod();

// });

// });

// // add data protection for encrypt and decrypt:

// services.AddDataProtection();

// // jwt authentication:

// // default:

// 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>();

// // JWT Configuration

// services.AddAuthentication(options =>

// {

// options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

// options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

// })

// .AddJwtBearer(options =>

// {

// options.RequireHttpsMetadata = false; // Change this to true in production

// options.SaveToken = true;

// options.TokenValidationParameters = new TokenValidationParameters

// {

// ValidateIssuerSigningKey = true,

// IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(configRoot["JWT:Secret"])),

// ValidateIssuer = true, // Ensure these values are set appropriately

// ValidateAudience = true,

// ValidIssuer = configRoot["JWT:ValidAudience"],

// ValidAudience = configRoot["JWT:ValidAudience"]

// };

// });

// // Add authorization policies

// services.AddAuthorization(options =>

// {

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

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

// // Add more policies as needed

// });

// services.AddSwaggerGen(options =>

// {

// options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

// {

// In = ParameterLocation.Header,

// Description = "Please enter Bearer Jwt Token",

// Name = "Authorization",

// Type = SecuritySchemeType.ApiKey

// });

// var scheme = new OpenApiSecurityScheme

// {

// Reference = new OpenApiReference

// {

// Type = ReferenceType.SecurityScheme,

// Id = "Bearer"

// }

// };

// options.AddSecurityRequirement(new OpenApiSecurityRequirement { { scheme, Array.Empty<string>() } });

// });

// services.AddMvc();

// services.AddControllers();

// services.AddSwaggerGen();

// services.AddHttpContextAccessor();

// // add cors

// //services.AddCors(options =>

// //{

// // options.AddDefaultPolicy(

// // policy =>

// // {

// // policy.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod();

// // // policy.WithOrigins("http://localhost:3000/").AllowAnyHeader().AllowAnyMethod();

// // });

// //});

// // session storage:

// // services.AddMvc().AddSessionStateTempDataProvider();

// services.AddDistributedMemoryCache();

// services.AddSession(options =>

// {

// options.IdleTimeout = TimeSpan.FromSeconds(60);

// options.Cookie.HttpOnly = true;

// options.Cookie.IsEssential = true;

// });

// // add service add memory cache

// // configure http request pipeline

// }

// public void Configure(WebApplication app, IWebHostEnvironment env)

// {

// if (app.Environment.IsDevelopment())

// {

// app.UseSwagger();

// app.UseSwaggerUI();

// }

// // add cors

// //app.UseCors(c => c.AllowAnyHeader().AllowAnyOrigin().AllowAnyMethod());

// app.UseHttpsRedirection();

// // we have to must use for JWt authentication app.UseAuthentication();

// app.UseAuthentication();

// app.UseAuthorization();

// app.MapControllers();

// app.Run();

// }

//}

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>();

services.AddScoped<CustomerDAL>();

services.AddScoped<CustomerService>();

// 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();

});

}

}

}