Role Base Authentication

loginAuthRepo.cs:

public Response GetByEmail(string email)

{

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

using (SqlConnection conn = new SqlConnection(connection))

{

Response response = new Response();

conn.Open();

// check email:

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

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

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

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

if( emailid != email )

{

response.StatusCode = 404;

response.StatusMessage = "Invalid email id";

}

else

{

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

SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd);

dataAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;

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

DataTable dataTable = new DataTable();

dataAdapter.Fill(dataTable);

var register = new RegisterModel();

conn.Close();

if (dataTable.Rows.Count > 0)

{

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

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

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

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

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

response.StatusMessage = "Ok";

response.StatusCode = 200;

response.Register = register;

}

else

{

return null;

}

}

// return await Task.FromResult(register);

// return register;

return response;

}

}

public Response Login(LoginModel login)

{

Response response = new Response();

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

using (SqlConnection conn = new SqlConnection(connection))

{

conn.Open();

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

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

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

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

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

conn.Close();

if (Variable == 1)

{

// check email

var getbyemail = GetByEmail(login.Email);

Console.WriteLine(login);

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

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

var issuer = \_jwt.Value.ValidIssuer;

var audience = \_jwt.Value.ValidAudience;

var tokenValidityInMinutes = \_jwt.Value.TokenValidityInMinutes;

var refreshTokenValidityInMinutes = \_jwt.Value.RefreshTokenValidityInMinutes;

var tokenDescriptor = new Microsoft.IdentityModel.Tokens.SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

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

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

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

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

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

new Claim(JwtRegisteredClaimNames.Jti,

Guid.NewGuid().ToString())

}),

Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

Issuer = issuer,

Audience = audience,

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

};

var token = tokenHandler.CreateToken(tokenDescriptor);

var refreshToken = RefreshTokenGenerator();

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

var Token = tokenHandler.WriteToken(token);

TokenModel model = new TokenModel();

model.RefreshToken = refreshToken;

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

model.AccessToken = Token;

response.Token = model;

response.Login = login;

response.StatusCode = 200;

response.StatusMessage = "Login Successful";

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

{

Expires = DateTime.UtcNow.AddMinutes(tokenValidityInMinutes),

//HttpOnly = true,

//Secure = true,

//IsEssential = true,

//SameSite = SameSiteMode.None

HttpOnly = true,

SameSite = SameSiteMode.Strict

});

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

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

}

else

{

response.StatusCode = 400;

response.StatusMessage = "Invalid UserName or Password";

}

return response;

}

}

public async Task<IActionResult> GetSecret()

{

string secretName = "sai/test/key";

string region = "ap-south-1";

IAmazonSecretsManager client = new AmazonSecretsManagerClient(RegionEndpoint.GetBySystemName(region));

GetSecretValueRequest request = new GetSecretValueRequest

{

SecretId = secretName,

VersionStage = "AWSCURRENT", // VersionStage defaults to AWSCURRENT if unspecified.

};

GetSecretValueResponse response;

try

{

response = await client.GetSecretValueAsync(request);

}

catch (Exception e)

{

// For a list of the exceptions thrown, see

// https://docs.aws.amazon.com/secretsmanager/latest/apireference/API\_GetSecretValue.html

throw e;

}

string secret = response.SecretString.ToString();

// SecretModel member = JsonSerializer.Deserialize<SecretModel>(secret.ToString());

// SecretModel model = new SecretModel();

// string json = JsonSerializer.Serialize(decrept3);

// SecretModel model = JsonConvert.DeserializeObject<SecretModel>(secret.ToString());

var userSecretProtector = \_dataProtectionProvider.CreateProtector(secretName);

byte[] secret4ByteInfo = userSecretProtector.Protect(System.Text.Encoding.UTF8.GetBytes((secret)));

var encryptedData = Convert.ToBase64String(secret4ByteInfo);

// decrypt:

byte[] encryptDataByte = Convert.FromBase64String(encryptedData);

var decrept1Base64 = userSecretProtector.Unprotect(encryptDataByte);

string decrept = System.Text.Encoding.UTF8.GetString(decrept1Base64);

// SecretModel member = JsonSerializer.Deserialize<SecretModel>(decrept);

// SecretModel model = new SecretModel();

// string json = JsonSerializer.Serialize(decrept3);

SecretModel model = JsonConvert.DeserializeObject<SecretModel>(decrept);

Console.WriteLine(model);

return Ok(new {model,encryptedData});

// Your code goes here

}

[Authorize(Roles = "User")]

[HttpGet]

[Route("StoreInsession")]

public IActionResult StoreInsession()

{

var studentData = \_studentService.GetAllStudent();

HttpContext.Session.SetString("Student",JsonConvert.SerializeObject(studentData));

// var name = HttpContext.Session.GetString("Student");

return Ok();

}

[Authorize(Roles ="User")]

[HttpGet]

[Route("GetSessionData")]

public IActionResult GetSessionData()

{

//if (HttpContext.Session != null)

//{

// var data = HttpContext.Session.GetString("Student");

// return Ok(data);

//}

//else

//{

// return BadRequest();

//}

var data = HttpContext.Session.GetString("Student");

if(string.IsNullOrEmpty(data))

{

return BadRequest("Expired session");

}

return Ok(data);

}

[Authorize(Roles ="Admin,User")]

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

[ApiController]

public class PaginationController : ControllerBase

{

private readonly IPaginationRepo \_paginationRepo;

private readonly IPaginationService \_paginationService;

public PaginationController(

IPaginationRepo paginationRepo,

IPaginationService paginationService

)

{

\_paginationRepo = paginationRepo;

\_paginationService = paginationService;

}

[HttpGet]

[Route("GetStudentsData")]

public async Task<IActionResult> GetStudents( string? SearchTerm, string? SortColumn, string? SortDirection, string? MultiNames, int Page = 1, int Limit = 5)

{

try

{

var result = await \_paginationService.GetStudentsRecords(Page,Limit,SearchTerm,SortColumn,SortDirection,MultiNames);

return Ok(result);

}

catch(Exception ex)

{

return StatusCode(StatusCodes.Status500InternalServerError,new StatusModel

{

StatusCode= 500,

Message = ex.Message,

});

}

}

}

Session storage:

Program.cs:

services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(1);//We set Time here

options.Cookie.HttpOnly = true;

options.Cookie.IsEssential = true;

});

// use session

app.UseSession();