**AuthRepository**

**using CityManagerApi.Models;**

namespace CityManagerApi.Data

{

public interface IAuthRepository

{

Task<User> Register(User user, string password);

Task<User> Login(string username, string password);

Task<bool> UserExists(string username);

}

}

**====================================================**

using CityManagerApi.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

namespace CityManagerApi.Data

{

public class AuthRepository : IAuthRepository

{

private DataContext \_context;

public async Task<User> Login(string username, string password)

{

var user = await \_context.Users.FirstOrDefaultAsync(x => x.Username == username);

if (user == null)

return null;

if (!VerifyPasswordHash(password, user.PasswordHash, user.PasswordSalt))

{

return null;

}

return user;

}

private bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)

{

using (var hmac = new System.Security.Cryptography.HMACSHA512(passwordSalt))

{

var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

for (int i = 0; i < computedHash.Length; i++)

{

if (computedHash[i] != passwordHash[i])

return false;

}

return true;

}

}

public async Task<User> Register(User user, string password)

{

byte[] passwordHash, passwordSalt;

CreatePasswordHash(password, out passwordHash, out passwordSalt);

user.PasswordHash = passwordHash;

user.PasswordSalt = passwordSalt;

await \_context.Users.AddAsync(user);

await \_context.SaveChangesAsync();

return user;

}

private void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)

{

using (var hmac = new System.Security.Cryptography.HMACSHA512())

{

passwordSalt = hmac.Key;

passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

}

}

public async Task<bool> UserExists(string username)

{

return await \_context.Users.AnyAsync(u => u.Username == username);

}

}

}

**=================================================**

**Authentication JWT**

var key = Encoding.ASCII.GetBytes(builder.Configuration.GetSection("AppSettings:Token").Value);

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters()

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(key),

ValidateIssuer = false,

ValidateAudience = false

};

});

app.UseAuthentication();

app.UseAuthorization();

**Login JWT Token**

[HttpPost]

public async Task<IActionResult> Login([FromBody] UserForLoginDto dto)

{

var user = await \_authRepository.Login(dto.Username, dto.Password);

if (user == null)

{

return Unauthorized();

}

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_configuration.GetSection("AppSettings:Token").Value);

var tokenDescriptor = new SecurityTokenDescriptor()

{

Subject = new ClaimsIdentity(

new Claim[]

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Name, user.Username)

}),

Expires = DateTime.Now.AddDays(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha512)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

var tokenString = tokenHandler.WriteToken(token);

return Ok(tokenString);

}