**GENERIC REPOSITORY WITH DBFIRST APPROACH**

Generic Repository is a data access pattern that prompts a more loosely coupled approach to data access. We create a generic repository, which queries the data source for the data, maps the data from the data source to a business entity, and persists changes in the business entity to the data source.

**Packages:**

Microsoft.AspNetCore.Http.Abstractions

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.Design

Microsoft.EntityFrameworkCore.SqlServer

System.Data.Common

System.Data.SqlClient

**Appsettings.json:**

"ConnectionStrings": {

"DBConnection": "Server=XXX; Initial Catalog=XXX;Integrated Security = true;Encrypt=false"

}

**Creating Models:**

namespace Feature.Entity.Entities

{

[Table("userProfileTbl")]

public class UserProfile : BaseEntity

{

public int Id { get; set; }

public string UserId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string RefreshToken { get; set; }

}

}

**Generic Repository Implementation:**

**In IGenericRepository.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

using System.Threading.Tasks;

namespace Feature.Repository.Interface.Generic

{

public interface IGenericRepository<T> where T : class

{

IEnumerable<T> GetAll();

T GetById(int id);

void Insert(T obj);

void Update(T obj);

void Delete(object id);

void Save();

Task<T> Filter(Expression<Func<T, bool>> Predicate);

}

}

**In GenericRepository.cs:**

namespace Feature.Repository.DBFirst.Generic

{

public class GenericRepository<T> : IGenericRepository<T> where T : class

{

protected readonly ApplicationDbContext \_context;

private DbSet<T> \_entities = null;

string errorMessage = string.Empty;

public GenericRepository(ApplicationDbContext context)

{

\_context = context;

\_entities = \_context.Set<T>();

}

public IEnumerable<T> GetAll()

{

return \_context.Set<T>().ToList();

}

public T GetById(int id)

{

var results = \_context.Find<T>(id);

return results == null ? null : results;

}

public void Insert(T obj)

{

\_entities.Add(obj);

\_context.Entry(obj).State = EntityState.Added;

\_context.SaveChanges();

}

public void Update(T obj)

{

\_entities.Attach(obj);

\_context.Entry(obj).State = EntityState.Modified;

\_context.SaveChanges();

}

public void Delete(object id)

{

T existing = \_entities.Find(id);

\_entities.Remove(existing);

\_context.SaveChangesAsync();

}

public void Save()

{

\_context.SaveChanges();

}

public virtual async Task<T> Filter(Expression<Func<T,bool>>Predicate)

{

return await \_entities.FirstOrDefaultAsync(Predicate);

}

}

}

**Repository Implementation:**

**In IUserRepository:**

namespace Feature.Repository.Interface.Interfaces

{

public interface IUserRepository : IGenericRepository<UserProfile>

{

Task<LoginResponse> Login(UserLogins userLogins);

}

}

**In UserRepository.cs**

namespace Feature.Repository.DBFirst.Repositories

{

public class UserRepository : GenericRepository<UserProfile>, IUserRepository

{

private ApplicationDbContext \_context;

IOptions<Appsetting> \_settings;

private IJWTTokenConfig iJWTConfig;

public UserRepository(ApplicationDbContext context, IOptions<Appsetting> settings,

IJWTTokenConfig \_iJWTConfig) :base(context)

{

\_context = context;

\_settings = settings;

iJWTConfig = \_iJWTConfig;

}

public async Task<LoginResponse> Login(UserLogins userLogins)

{

try

{

var user = \_context.UserProfileTbl.Where(x => (x.UserId.Equals(userLogins.UserName)) && (x.Password.Equals(userLogins.Password))).FirstOrDefault();

//var user = \_context.UserProfileTbl.FirstOrDefault();

if (user != null)

{

var token = iJWTConfig.GenerateJwtToken(user.UserId);

if (token != null)

{

user.RefreshToken = iJWTConfig.GenerateRefreshToken();

//ToDO update refreshtoken in database for user record

await \_context.SaveChangesAsync();

LoginResponse loginResponse = new LoginResponse()

{

FirstName = user.FirstName,

LastName = user.LastName,

Token = token,

RefreshToken = user.RefreshToken,

};

return loginResponse;

}

}

return null;

}

catch (Exception ex)

{

throw;

}

}

}

}

**Service Implementation:**

**In IUserService.cs**

namespace Feature.Services.Abstract

{

public interface IUserService

{

Task<LoginResponse> Login(UserLogins userLogins);

IEnumerable<UserProfile> GetAll();

Task<UserProfile> GetById(int id);

void Insert(UserProfile profile);

void Update(UserProfile profile);

void Delete(object id);

Task<UserProfile> Filter(Expression<Func<UserProfile, bool>> Predicate);

}

}

**In UserService.cs**

namespace Feature.Services.Concrete

{

public class UserService : IUserService

{

private readonly IUserRepository \_userRepository;

public UserService(IUserRepository userRepository)

{

\_userRepository = userRepository;

}

public async Task<LoginResponse> Login(UserLogins userLogins)

{

return await \_userRepository.Login(userLogins);

}

public IEnumerable<UserProfile> GetAll()

{

return \_userRepository.GetAll();

}

public async Task<UserProfile> GetById(int id)

{

return \_userRepository.GetById(id);

}

public void Insert(UserProfile profile)

{

\_userRepository.Insert(profile);

}

public void Update(UserProfile profile)

{

\_userRepository.Update(profile);

}

public void Delete(object id)

{

\_userRepository.Delete(id);

}

public async Task<UserProfile> Filter(Expression<Func<UserProfile, bool>> Predicate)

{

return await \_userRepository.Filter(Predicate);

}

}

}

**In UserController.cs**

namespace Feature.API.Controllers

{

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

[ApiController]

public class UserController : ControllerBase

{

private readonly ILogger<UserController> \_logger;

private readonly ILogger<UserController> \_seriLogger;

private readonly ILoggerExtention \_nlogLogger;

private readonly IUserService \_userService;

public UserController(ILogger<UserController> logger, ILogger<UserController> seriLogger, ILoggerExtention nlogLogger, IUserService userService)

{

\_seriLogger = seriLogger;

\_nlogLogger = nlogLogger;

\_logger = logger;

\_userService = userService;

}

[Route("GetAllUsers")]

[HttpGet]

public IEnumerable<UserProfile> GetAll()

{

return \_userService.GetAll();

}

[Route("GetById")]

[HttpGet]

public Task<UserProfile> GetById(int id)

{

return \_userService.GetById(id);

}

[Route("Create User")]

[HttpPost]

public async Task<IActionResult> CreateUser(UserProfile userProfile)

{

\_userService.Insert(userProfile);

return Ok();

}

[Route("Update User")]

[HttpPost]

public async Task<IActionResult> UpdateUser(UserProfile userProfile)

{

\_userService.Update(userProfile);

return Ok();

}

[Route("Delete User")]

[HttpDelete]

public async Task<IActionResult> DeleteUser(int id)

{

\_userService.Delete(id);

return Ok();

}

[HttpGet]

public Task<UserProfile> Filter(Expression<Func<UserProfile, bool>> Predicate)

{

return \_userService.Filter(Predicate);

}

}

}

**In Program.cs**

builder.Services.AddDbContext<JWTDBContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection")));

builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection")));

builder.Services.AddTransient<IUserService, UserService>();

builder.Services.AddTransient<IUserRepository, UserRepository>();

builder.Services.AddTransient <IGenericRepository<UserProfile>, GenericRepository<UserProfile>>();