# Project

* Site-19-Api-Ng-UnitOfWork-OptQueries-CofirmService-MemMessage
  + Copied from: Site-18-Api-Ng-responsive nav
* For the “MySocialConnect-API”
  + dotnet restore
  + dotnet build
  + Go to project : MSC.WebApi
    - dotnet build : to build
    - dotnet run : to run the api

# New Resources

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/DB/UnitOfWork |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| MSC.Core/Extensions/AppServiceExtensions.cs |  |
| MSC.Core/Repositories/UserRepository |  |
| MSC.Core/Repositories/MessageRepository |  |
| MSC.Core/Repositories/SignalRRepository |  |
| MSC.Core/BusinessLogic/LikesBusinessLogic |  |
| MSC.Core/BusinessLogic/MessageBusinessLogic |  |
| MSC.Core/BusinessLogic/SignalRBusinessLogic |  |
| MSC.Core/BusinessLogic/UserBusinessLogic |  |

# Clear Database

* New migrations created
* Clear the users
* dotnet ef database drop
* dotnet ef database update

# Setting Up UnitOfWork

## MSC.Core/DB/UnitOfWork

### IUnitOfWork.cs

using System.Threading.Tasks;

using MSC.Core.Repositories;

namespace MSC.Core.DB.UnitOfWork;

public interface IUnitOfWork

{

    IUserRepository UserRepo {get;}

    ILikesRepository LikesRepo {get;}

    IMessageRepository MessageRepo {get;}

    ISignalRRepository SignalRRepo {get;}

    Task<bool> SaveChangesAsync();

    bool HasChanges();

}

### UnitOfWork.cs

using System.Threading.Tasks;

using MSC.Core.DB.Data;

using MSC.Core.Repositories;

namespace MSC.Core.DB.UnitOfWork;

public class UnitOfWork : IUnitOfWork

{

    private readonly DataContext \_context;

    private readonly IUserRepository \_userRepo;

    private readonly ILikesRepository \_likesRepo;

    private readonly IMessageRepository \_msgRepo;

    private readonly ISignalRRepository \_sigrRepo;

    public UnitOfWork(DataContext context,

                        IUserRepository userRepo, ILikesRepository likesRepo,

                        IMessageRepository msgRepo, ISignalRRepository sigrRepo)

    {

        \_context = context;

        \_userRepo = userRepo;

        \_likesRepo = likesRepo;

        \_msgRepo = msgRepo;

        \_sigrRepo = sigrRepo;

    }

    public IUserRepository UserRepo => \_userRepo;

    public ILikesRepository LikesRepo => \_likesRepo;

    public IMessageRepository MessageRepo => \_msgRepo;

    public ISignalRRepository SignalRRepo => \_sigrRepo;

    public async Task<bool> SaveChangesAsync()

    {

        return await \_context.SaveChangesAsync() > 0;

    }

    public bool HasChanges()

    {

        return \_context.ChangeTracker.HasChanges();

    }

}

## DI – UnitOFWork

### MSC.Core/Extensions/AppServiceExtensions.cs

#### AddServices Method

    public static IServiceCollection AddServices(this IServiceCollection services, IConfiguration config)

    {

        services.AddScoped<IUserRepository, UserRepository>();

        services.AddScoped<IUserBusinessLogic, UserBusinessLogic>();

        services.AddScoped<ILikesRepository, LikesRepository>();

        services.AddScoped<ILikesBusinessLogic, LikesBusinessLogic>();

        services.AddScoped<IMessageRepository, MessageRepository>();

        services.AddScoped<IMessageBusinessLogic, MessageBusinessLogic>();

        services.AddScoped<ITokenService, TokenService>();

        services.AddScoped<IPhotoService, PhotoService>();

        //services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies()); //when have single project/assembly

        services.AddAutoMapper(typeof(AutoMapperProfiles).Assembly);

        //adding the Cloudinary to read data from

        //check programs.cs for ref: builder.Services.Configure<EnvConfig>(configuration);

        services.Configure<CloudinaryConfig>(config.GetSection(ConfigKeyConstants.CloudinarySettingsKey));

        //add the action filter as a service, it wil get applied to the abse controller

        services.AddScoped<LogUserActivityFilter>();

        services.AddSignalR();

        services.AddSingleton<PresenceTrackerMemory>();

        services.AddScoped<ISignalRRepository, SignalRRepository>();

        services.AddScoped<ISignalRBusinessLogic, SignalRBusinessLogic>();

        services.AddScoped<IUnitOfWork, UnitOfWork>();

        return services;

    }

# CleanUp Repos - MSC.Core/Repositories

## UserRepository

### IUserRepository.cs

Comment or remove SaveAllSync()

Change RegisterUserAsync to RegisterUSer

    /\*changed the signature after UnitOfWork Implementation\*/

    //Task<bool> RegisterUserAsync(AppUser user);

    void RegisterUser(AppUser appUser);

### UserRepository.cs

Comment or remove SaveAllSync()

Change RegisterUserAsync to RegisterUSer

    /\*changed the signature after UnitOfWork Implementation

    public async Task<bool> RegisterUserAsync(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(appUser);

        var isSave = await SaveAllAsync();

        return isSave;

    }

    \*/

    public void RegisterUser(AppUser appUser)

    {

        if (appUser == null)

            throw new ValidationException("Invalid user");

        \_context.Users.Add(appUser);

    }

## MessageRepository

### IMessageRepository.cs

Comment or remove SaveAllSync()

### MessageRepository.cs

Comment or remove SaveAllSync()

Go to method GetMessageThread and remove line \_context.SaveChangesAsync

Go to BusinessLogic and do the save via UnitOfWork

## SignalRRepository

### ISignalRRepository.cs

Comment or remove SaveAllSync()

### SignalRRepository.cs

Comment or remove SaveAllSync()

# Refactory MSC.Core/BusinessLogic – Use UnitOfWork

## LikesBusinessLogic.cs

1. Remove likesRepo and userRepo injected via constructor.
2. Inject unitOFWork instead
3. And make other changes to fix the issues

    private readonly IUnitOfWork \_uow;

    public LikesBusinessLogic(IUnitOfWork uow)

    {

        \_uow = uow;

    }

And then fix all errors.

In the following methods use UnitOfWork SaveChangesAsync

1. AddLike

## MessageBusinessLogic.cs

1. Remove msgRepo and userRepo injected via constructor
2. Inject unitOFWork instead
3. And make other changes to fix the issues

    private readonly IUnitOfWork \_uow;

    private readonly IMapper \_mapper;

    public MessageBusinessLogic(IUnitOfWork uow, IMapper mapper)

    {

        \_uow = uow;

        \_mapper = mapper;

    }

And then fix all errors.

In the following methods use UnitOfWork SaveChangesAsync

1. AddMessageHandle
2. DeleteMessage

Go to GetMessageThread method and save the changes via UnitOfWork. Remember as we have removed the saving code from the repo method

    public async Task<IEnumerable<MessageDto>> GetMessageThread(int currentUserId, int receipientId)

    {

        var messages = await \_uow.MessageRepo.GetMessageThread(currentUserId, receipientId);

        if(messages == null)

            return null;

        if(\_uow.HasChanges())

            await \_uow.SaveChangesAsync();

        var messagesDto = \_mapper.Map<IEnumerable<MessageDto>>(messages);

        return messagesDto;

    }

## SignalRBusinessLogic.cs

1. Remove srRepo injected via constructor
2. Inject unitOFWork instead
3. And make other changes to fix the issues

    private readonly IUnitOfWork \_uow;

    public SignalRBusinessLogic(IUnitOfWork uow)

    {

        \_uow = uow;

    }

Change method name SaveAllSync to SaveAllAsync on both interface and the class

And then fix all errors.

In the following methods use UnitOfWork SaveChangesAsync

1. SaveAllAsync

## UserBusinessLogc.cs

1. Remove userRepo injected via constructor
2. Inject unitOFWork instead
3. And make other changes to fix the issues

    private readonly UserManager<AppUser> \_userManager;

    private readonly RoleManager<AppRole> \_roleManager;

    private readonly ITokenService \_tokenService;

    private readonly IMapper \_mapper;

    private readonly IPhotoService \_photoService;

    private readonly IUnitOfWork \_uow;

    public UserBusinessLogic(UserManager<AppUser> userManager, RoleManager<AppRole> roleManager,

                            ITokenService tokenService, IMapper mapper,

                            IPhotoService photoService, IUnitOfWork uow)

    {

        \_userManager = userManager;

        \_roleManager = roleManager;

        \_tokenService = tokenService;

        \_mapper = mapper;

        \_photoService = photoService;

        \_uow = uow;

    }

And then fix all errors.

In the following methods use UnitOfWork SaveChangesAsync

1. UpdateUserAsync
2. AddPhotoAsync
3. DeletePhotoAsync
4. SetPhotoMainAsync
5. LogUserActivityAsync

# Optimizing Query - GetMessageThread

## MSC.Core/Repositories/MessageRepository

### IMessageRepository.cs

GetMessageThread will return MessageDto

    /\*Method is now returning MessageDto\*/

    //Task<IEnumerable<UserMessage>> GetMessageThread(int currentUserId, int receipientId);

    Task<IEnumerable<MessageDto>> GetMessageThread(int currentUserId, int receipientId);

### MessageRepository.cs

Comment the current method and rewrite it to use projects. This means we don’t have to do include now

    public async Task<IEnumerable<MessageDto>> GetMessageThread(int currentUserId, int receipientId)

    {

        var query = \_context.Messages

                            .Where(m =>

                                (m.RecipientId == currentUserId && m.SenderId == receipientId && !m.RecipientDeleted) ||

                                (m.RecipientId == receipientId && m.SenderId == currentUserId && !m.SenderDeleted)

                            )

                            .OrderBy(m => m.DateMessageSent)

                            .AsQueryable();

        var unreadMessages = query.Where(m => m.DateMessageRead == null && m.Recipient.Id == currentUserId).ToList();

        if(unreadMessages != null && unreadMessages.Any())

        {

            //update the date

            unreadMessages.ForEach(x => {x.DateMessageRead = DateTime.UtcNow;});

        }

        var messages = await query.ProjectTo<MessageDto>(\_mapper.ConfigurationProvider).ToListAsync();

        return messages;

    }

## MSC.Core/BusinessLogic/MessageBusinessLogic.cs

Update the GetMessageThread since it is returning the MessageDto

    public async Task<IEnumerable<MessageDto>> GetMessageThread(int currentUserId, int receipientId)

    {

        //now return MessageDto so no need to do mapping below

        var messages = await \_uow.MessageRepo.GetMessageThread(currentUserId, receipientId);

        if(messages == null)

            return null;

        if(\_uow.HasChanges())

            await \_uow.SaveChangesAsync();

        //var messagesDto = \_mapper.Map<IEnumerable<MessageDto>>(messages);

        //return messagesDto;

        return messages;

    }

# Optimizing Query – Get Members

## MSC.Core/Repositories/UserRepository

### IUserRepository.cs

    Task<string> GetUserGenderAsync(Guid guid);

### UserRepository.cs

    public async Task<string> GetUserGenderAsync(Guid guid)

    {

        var gender = await \_context.Users.Where(x => x.Guid == guid).Select(x => x.Gender).FirstOrDefaultAsync();

        return gender;

    }

## MSC.Core/BusinessLogic/UserBusinessLogic

### IUSerBusinessLogic.cs

    Task<string> GetUserGenderAsync(Guid guid);

### UserBusinessLogic.cs

    public async Task<string> GetUserGenderAsync(Guid guid)

    {

        var gender = await \_uow.UserRepo.GetUserGenderAsync(guid);

        return gender;

    }

## MSC.WebApi/Controllers/UsersController.cs

    public async Task<ActionResult<PagedList<UserDto>>> GetUsers([FromQuery]UsersSearchParamDto userParams)

    {

        var userClaims = User.GetUserClaims();

        if(userClaims == null)

            return BadRequest("User issue");

        //get the current user Gender

        var gender = await \_userBusinessLogic.GetUserGenderAsync(userClaims.Guid);

        //var users = await \_userBusinessLogic.GetUsersAsync();

        if(string.IsNullOrWhiteSpace(userParams.Gender))

            userParams.Gender = gender.ToLower() == "male" ? "female" : "male";

        var users = await \_userBusinessLogic.GetUsersAMQEAsync(userParams, userClaims.Guid);

        if (users == null || !users.Any())

        {

            return NotFound("No users found!");

        }

        //write pagination header

        Response.AddPaginationHeader(users.CurrentPage, users.PageSize, users.TotalCount, users.TotalPages);

        return Ok(users);

    }