Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

**Лабораторна робота № 8**

З дисципліни: «Компоненти програмної інженерії - 2»

Реалізація сервісів бізнес-логіки проекту

|  |  |
| --- | --- |
| Виконав студент групи ІТ- 73  Паньків О. В.  Дата здачі  Захищено з балом \_\_\_\_\_\_\_\_\_\_\_ | Перевірено:  ас. кафедри АУТС  Галушко Д. О. |

Київ 2020

Для зменшення зв’язності компонентів нашого програмного продукту ми застосували впровадження залежностей за допомогою контейнера сервісів. Наші сервіси і є в даному випадку залежностями. Для початку створимо інтерфейси сервісів бізнес-логіки:

IOrderService

using BLL.DTO;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Services.Interfaces

{

public interface IOrderService

{

Task AddPerformerAsync(OrderDTO projectDTO, UserDTO performer);

Task ChangeDescriptionAsync(OrderDTO projectDTO, string description);

Task ChangeManagerAsync(OrderDTO projectDTO, UserDTO manager);

Task CreateOrderAsync(OrderDTO projectDTO);

Task DeleteOrderAsync(OrderDTO projectDTO, UserDTO manager);

Task<IEnumerable<OrderDTO>> GetByManagerAsync(string name);

Task<OrderDTO> GetOrderByIdAsync(int id);

}

}

IOrderTaskService

using BLL.DTO;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Services.Interfaces

{

public interface IOrderTaskService

{

Task AssignPerformerAsync(OrderTaskDTO projectTaskDTO, UserDTO performer);

Task CreateTaskAsync(OrderTaskDTO projectTaskDTO);

Task ChangeDesriptionAsync(OrderTaskDTO projectTaskDTO, string description);

}

}

IUserService  
using BLL.DTO;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Services.Interfaces

{

public interface IUserService

{

Task CreateUserAsync(UserDTO userDTO);

Task<UserDTO> GetUserByNameAsync(string name);

Task AddRefreshTokenAsync(string refreshToken, string IdentityId);

Task ExchangeRefreshTokenAsync(string oldToken, string newToken, string IdentityId);

}

}

Далі ми реалізовуємо наші сервіси та клас BaseService, який вони наслідуватимуть:

BaseService

ing System;

using System.Collections.Generic;

using System.Text;

using AutoMapper;

using DAL.Base;

using DAL.UnitOfWork;

namespace BLL.Services.Base

{

public abstract class BaseService

{

protected readonly IUnitOfWork \_unitOfWork;

protected readonly IMapper \_mapper;

public BaseService(IUnitOfWork unitOfWork, IMapper mapper)

{

\_unitOfWork = unitOfWork;

\_mapper = mapper;

}

}

}

OrderService

using AutoMapper;

using BLL.DTO;

using BLL.Exceptions;

using BLL.Services.Base;

using BLL.Services.Interfaces;

using DAL.Entities;

using DAL.Entities.Enums;

using DAL.UnitOfWork;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace BLL.Services

{

//TODO: add XML annotations to non-trivial methods

public class OrderService : BaseService, IOrderService

{

public OrderService(IUnitOfWork unitOfWork, IMapper mapper)

: base(unitOfWork, mapper) { }

public async Task AddPerformerAsync(OrderDTO orderDTO, UserDTO performer)

{

var searchedOrder = await \_unitOfWork.OrderRepository

.GetByIdAsync(orderDTO.Id);

if (searchedOrder == null)

throw new NotFoundException(searchedOrder.Title);

var searchedPerformer = await \_unitOfWork.UserRepository

.GetByIdAsync(performer.Id);

if (searchedPerformer == null)

throw new NotFoundException(searchedPerformer.Name);

searchedOrder.Users.Add(searchedPerformer);

searchedPerformer.Order = searchedOrder;

await \_unitOfWork.OrderRepository.UpdateAsync(searchedOrder);

await \_unitOfWork.UserRepository.UpdateAsync(searchedPerformer);

}

public async Task ChangeDescriptionAsync(OrderDTO orderDTO, string description)

{

var searchedOrder = await \_unitOfWork.OrderRepository

.GetByIdAsync(orderDTO.Id);

if (searchedOrder == null)

throw new NotFoundException(searchedOrder.Title);

searchedOrder.Description = description;

await \_unitOfWork.OrderRepository.UpdateAsync(searchedOrder);

}

public async Task ChangeManagerAsync(OrderDTO orderDTO, UserDTO manager)

{

var searchedOrder = await \_unitOfWork.OrderRepository

.GetByIdAsync(orderDTO.Id);

if (searchedOrder == null)

throw new NotFoundException(searchedOrder.Title);

var searchedUser = await \_unitOfWork.UserRepository

.GetByIdAsync(manager.Id);

if (searchedUser == null)

throw new NotFoundException(searchedUser.Name);

searchedUser.Order = searchedOrder;

searchedOrder.OrderChef = searchedUser;

await \_unitOfWork.UserRepository.UpdateAsync(searchedUser);

await \_unitOfWork.OrderRepository.UpdateAsync(searchedOrder);

}

public async Task CreateOrderAsync(OrderDTO orderDTO)

{

var mappedOrder = \_mapper.Map<Order>(orderDTO);

await \_unitOfWork.OrderRepository.AddAsync(mappedOrder);

}

public async Task DeleteOrderAsync(OrderDTO orderDTO, UserDTO manager)

{

var searchedUser = await \_unitOfWork.UserRepository

.GetByIdAsync(manager.Id);

if (searchedUser == null)

throw new NotFoundException(searchedUser.Name);

var searchedOrder = await \_unitOfWork.OrderRepository

.GetByIdAsync(orderDTO.Id);

if (searchedOrder == null)

throw new NotFoundException(searchedOrder.Title);

if (searchedUser.Role != UserRole.Chef)

{

//exception

}

var orders = await \_unitOfWork.OrderRepository

.GetAllAsync();

//prbbly should change it

bool isCurrentOrderManager = orders

.Exists(x => x.OrderChef.Id == manager.Id);

if (isCurrentOrderManager)

{

await \_unitOfWork.OrderRepository.DeleteAsync(searchedOrder);

}

else

{

//exception or result

}

}

public async Task<IEnumerable<OrderDTO>> GetByManagerAsync(string name)

{

var searchedUser = await \_unitOfWork.UserRepository

.GetByCriteriaAsync(x => x.Name == name);

if (searchedUser == null)

throw new NotFoundException(searchedUser.Name);

var orders = await \_unitOfWork.OrderRepository

.GetAllAsync();

var ordersByManager = orders

.Where(x => x.OrderChef.Name == name).ToList();

if (ordersByManager == null || ordersByManager.Count == 0)

throw new NotFoundException("");

return \_mapper.ProjectTo<OrderDTO>(ordersByManager as IQueryable);

}

public async Task<OrderDTO> GetOrderByIdAsync(int id)

{

var searchedOrder = await \_unitOfWork.OrderRepository

.GetByIdAsync(id);

if (searchedOrder == null)

throw new NotFoundException(searchedOrder.Title);

return \_mapper.Map<OrderDTO>(searchedOrder);

}

}

}

OrderTaskService

using AutoMapper;

using BLL.DTO;

using BLL.Exceptions;

using BLL.Services.Base;

using BLL.Services.Interfaces;

using DAL.Entities;

using DAL.UnitOfWork;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Services

{

internal class OrderTaskService : BaseService, IOrderTaskService

{

public OrderTaskService(IUnitOfWork unitOfWork, IMapper mapper)

: base(unitOfWork, mapper) { }

public async Task AssignPerformerAsync(OrderTaskDTO projectTaskDTO, UserDTO performer)

{

var searchedTask = await \_unitOfWork.OrderTaskRepository

.GetByIdAsync(projectTaskDTO.Id);

if (searchedTask == null)

throw new NotFoundException(searchedTask.Title);

var searchedUser = await \_unitOfWork.UserRepository

.GetByIdAsync(performer.Id);

if (searchedUser == null)

throw new NotFoundException(searchedUser.Name);

searchedTask.Performer = searchedUser;

searchedUser.Tasks.Add(searchedTask);

await \_unitOfWork.OrderTaskRepository.

UpdateAsync(searchedTask);

await \_unitOfWork.UserRepository.

UpdateAsync(searchedUser);

}

public async Task CreateTaskAsync(OrderTaskDTO projectTaskDTO)

{

var taskExist = await \_unitOfWork.OrderTaskRepository

.GetByCriteriaAsync(x => x.Title == projectTaskDTO.Title) != null;

if (taskExist) throw new Exception("already exists");//gotta change

var mappedTask = \_mapper.Map<OrderTask>(projectTaskDTO);

await \_unitOfWork.OrderTaskRepository

.AddAsync(mappedTask);

}

public async Task ChangeDesriptionAsync(OrderTaskDTO orderTaskDTO, string description)

{

var searchedTask = await \_unitOfWork.OrderTaskRepository

.GetByIdAsync(orderTaskDTO.Id);

if (searchedTask == null)

throw new NotFoundException(searchedTask.Title);

searchedTask.Description = description;

await \_unitOfWork.OrderTaskRepository

.UpdateAsync(searchedTask);

}

}

}

UserService

using AutoMapper;

using BLL.DTO;

using BLL.Exceptions;

using BLL.Services.Base;

using BLL.Services.Interfaces;

using DAL.Entities;

using DAL.UnitOfWork;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

namespace BLL.Services

{

internal class UserService : BaseService, IUserService

{

public UserService(IUnitOfWork unitOfWork, IMapper mapper)

: base(unitOfWork, mapper) { }

public async Task CreateUserAsync(UserDTO userDTO)

{

var mappedUser = \_mapper.Map<OrderUser>(userDTO);

await \_unitOfWork.UserRepository.AddAsync(mappedUser);

}

public async Task<UserDTO> GetUserByNameAsync(string name)

{

var searchedUser = await \_unitOfWork.UserRepository.GetByCriteriaAsync(u => u.Name == name);

if (searchedUser == null) throw new NotFoundException("");

var mappedUser = \_mapper.Map<UserDTO>(searchedUser);

return mappedUser;

}

public async Task ExchangeRefreshTokenAsync(string oldToken, string newToken, string identityId)

{

var searchedUser = await \_unitOfWork.UserRepository

.GetByCriteriaAsync(u => u.ApplicationUserId == identityId);

if (searchedUser == null) throw new NotFoundException("");

if (!searchedUser.HasValidRefreshToken(oldToken))

throw new Exception(""); //create own ex

searchedUser.RemoveRefreshToken(oldToken);

searchedUser.AddRefreshToken(newToken, identityId);

await \_unitOfWork.UserRepository.UpdateAsync(searchedUser);

}

public async Task AddRefreshTokenAsync(string refreshToken, string identityId)

{

var searchedUser = await \_unitOfWork.UserRepository

.GetByCriteriaAsync(u => u.ApplicationUserId == identityId);

if (searchedUser == null) throw new NotFoundException("");

searchedUser.AddRefreshToken(refreshToken, identityId);

await \_unitOfWork.UserRepository.UpdateAsync(searchedUser);

}

}

}