Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №2

з дисципліни <<Технології розробки програмного забезпечення>>

Тема << Діаграма варіантів використання>>

Виконав:
студент ІА-23
Содолиснький Вадим

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи

Хід роботи

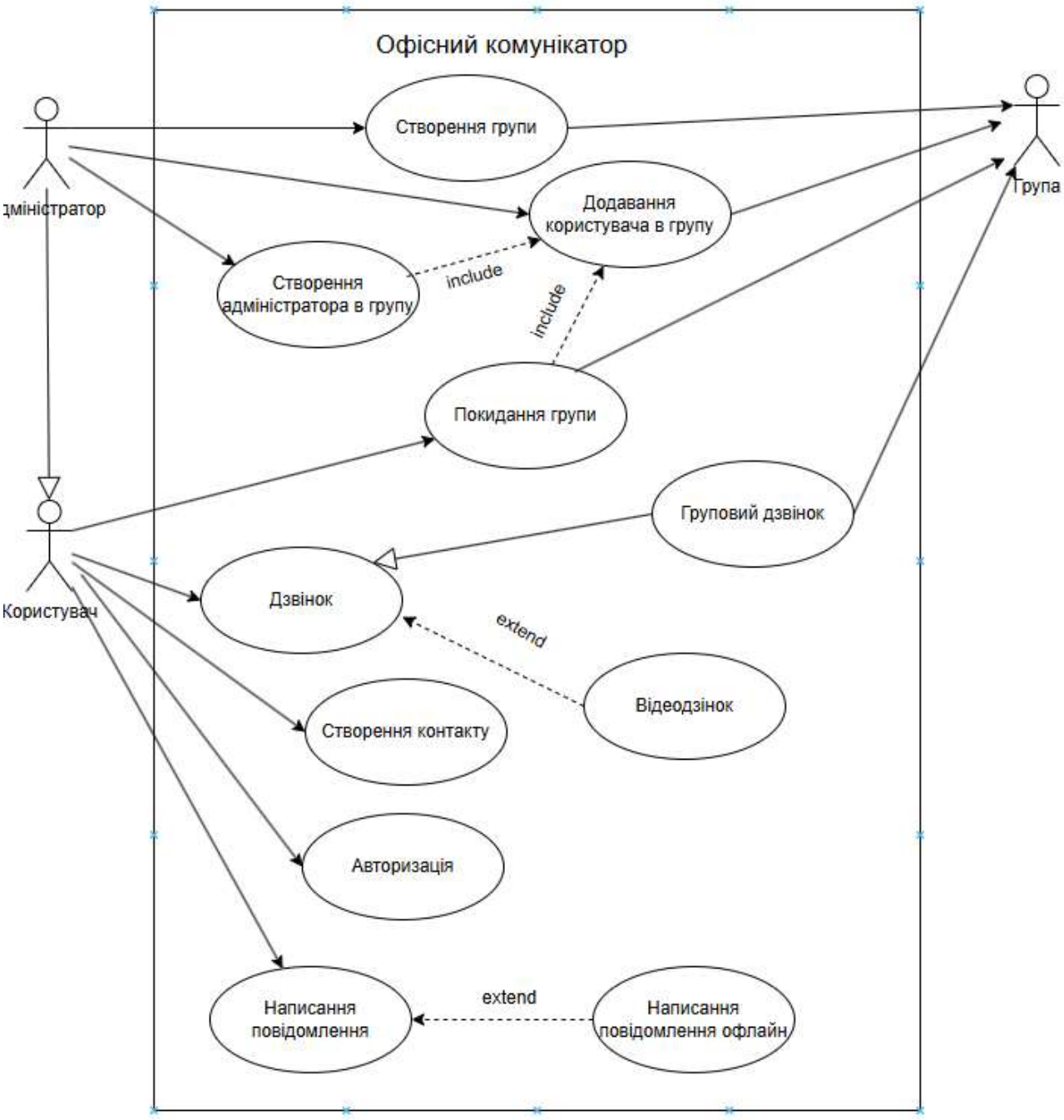

Рис. 1 – Діаграма прецендетів для теми офісний комунікатор

Прецедент 1: Відправка повідомлення.

Передумова: Користуввач зареєстрований.

Постумова: Якщо користувач онлайн, тоді повідомлення надіслане. Інакше система чекатиме поки з‘явиться мережа.

Взаємодіючі сторони: Користувач та група або два користувачі.

Короткий опис: Даний варіант використання описує надсилання повідомлення групі або користувачу.

Основний потік подій

Даний варіант використання починає виконуватись коли користувач хоче надіслати повідомлення.

1. Користувач вводить повідомлення або файл та надсилає.
2. Якщо користувач знаходиться в мережі то повідомлення надіслані. В іншому випадку виключення №1.

Виключення

Виключення №1

Користувач знаходиться не в мережі і повідомлення не надіслано до групи або іншого користувача. Додаток чекає поки з'явиться мережа для надсилання повідомлення.

Прецедент 2: Голосовий дзвінок

Передумова: Користувач зареєстрований.

Постумова: Користувач знаходиться в мережі та здійснюється дзвінок.
Взаємодіючи сторони: Два користувачі або користувач та група.

Короткий опис: Даний варіант описує здійснення дзвінка.

Основний потік подій.

Даний варіант починає виконуватись, коли користувач хоче здійснити дзвінок.

1. Користувач обирає групу або іншого користувача для здійснення дзвінка.

2. Якщо обидва користувачі знаходяться в мережі, то здійснюється відеодзвінок. Якщо один з користувачів не знаходиться в мережі, тоді виключення №1.

Виключення.

Виключення №1.

Один з користувачів знаходиться не в мережі і немає змоги провести дзвінок.

Прецедент 3: Додавання користувача в групу

Передумова: Користувач, який хоче додати іншого користувача в групу, повинен бути адміністратором цієї групи.

Постмова: Користувач доданий до групи.

Взаємодіючи сторони: Два користувачі та група.

Короткий опис: Даний варіант використання описує додавання користувача у групу.

Основний потік подій.

Даний варіант починає виконуватись коли користувач хоче додати іншого користувача в групу.

1. Адміністратор обирає групу та користувача якого хоче додати.
2. Система додає користувача в дану групу, якщо він в ній ще не перебуває.

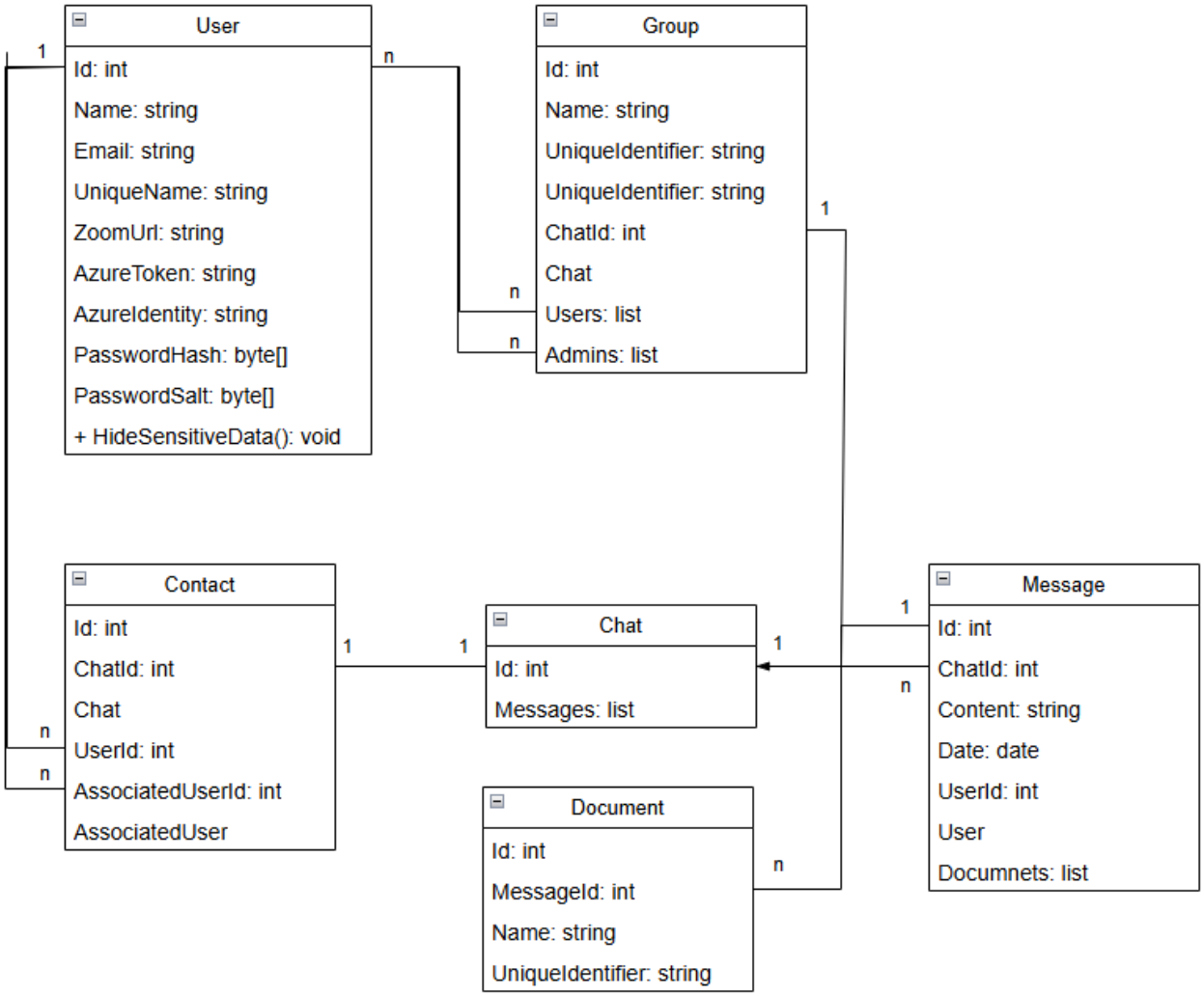

Рис. 2 – Діаграма класів

```
1 reference
public interface IRepository<T, in TDto, in TUpdateDto>
{
    4 references
    Task<T?> GetByIdWithIncludeAsync(int id);
    1 reference
    Task<T?> GetByIdAsync(int id);
    2 references
    Task<List<T>> GetAllAsync();
    2 references
    Task<T> AddAsync(TDto entity);
    3 references
    Task<bool> UpdateAsync(TUpdateDto entity);
    1 reference
    Task<bool> DeleteAsync(int id);
}
```

Рис. 3 – Інтерфейс IRepository

Даний інтерфейс є generic і містить три параметри типу: T, TDto та TUpdateDto. Перший параметр T відповідає за основний тип сутності, з якою працює репозиторій. Другий параметр TDto використовується для операцій додавання, а третій TUpdateDto – для операцій оновлення.

Інтерфейс містить кілька методів для роботи з даними. Метод GetByIdWithIncludeAsync асинхронно отримує сутність типу T за її ідентифікатором, включаючи пов'язані дані. Метод GetByIdAsync також повертає сутність за ідентифікатором, але без додаткових пов'язаних даних. Інший метод, GetAllAsync, дозволяє отримати список усіх сутностей типу T.

Операції додавання та оновлення реалізовані через методи AddAsync та UpdateAsync. Метод AddAsync додає нову сутність, використовуючи об'єкт типу TDto, тоді як UpdateAsync оновлює існуючу сутність за допомогою об'єкта типу TUpdateDto. Крім того, інтерфейс містить метод DeleteAsync, який видаляє сутність за її ідентифікатором і повертає результат виконання у вигляді булевого значення.

```
public class UserRepository : IRepository<User, UserDto, UserUpdateDto>
{
    private readonly OfficeDbContext _dbContext;
    private readonly AuthHelper _authHelper;
    private readonly IMapper _mapper;

    3 references
    public UserRepository(OfficeDbContext dbContext, IMapper mapper, AuthHelper authHelper)
    {
        _dbContext = dbContext;
        _mapper = mapper;
        _authHelper = authHelper;
    }

    2 references
    public async Task<List<User>> GetAllAsync()
    {
        return await _dbContext.Users.ToListAsync();
    }

    1 reference
    public async Task<User?> GetByIdAsync(int id)
    {
        return await _dbContext.Users.FindAsync(id);
    }


    4 references
    public async Task<User?> GetByIdWithIncludeAsync(int id)
    {
        return await _dbContext.Users
            .Include(u => u.Groups)
            .Include(u => u.Contacts)
            .ThenInclude(c => c.AssociatedUser)
            .FirstOrDefaultAsync(u => u.Id == id);
    }
```

Рис. 4 – Уналідування класу UserRepository від IRepository

Висновок: Я створив діаграму прецедентів для теми офісний комунікатор щоб знати, яким вимогам повинен відповідати застосунок. Описав прецеденти щоб розуміти, яким чином повинна відбуватись взаємодія користувача із застосунком та який розвиток подій очікувати. Для уніфікації роботи застосунку з базою даних створив інтерфейс IRepository, який описує основні методи взаємодії.