НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни “ Основи програмування”

(назва дисципліни)

на тему **“Соціальна мережа. Комунікації між членами мережі” ”**

Студента(-ки) 1 курсу ІП-з41 групи

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник: Старший викладач кафедри інформатики та програмної інженерії Вітковська І. І

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2025 рік

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Спеціальність «Інженерія програмного забезпечення»

Курс 1 Група ІП-з41 Семестр 2

**ЗАВДАННЯ**

**на курсову роботу студента**

5 Зайченко Єлизавети Сергіївни 5

1. Тема роботи: “Соціальна мережа. Комунікації між членами мережі ”

2. Строк здачі студентом закінченої роботи: 18 червня 2025

3. Вихідні дані до роботи: Соціальну мережу складають групи її членів, пов’язаних між собой відносинами дружності. Кожний її член може керувати цими відносинами, а саме додавати до своєї мережі друзів: запрошувати до своїх друзів та просити запрошеня для себе.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці):

* Опис постановки задачі
* Аналіз предметної області.
* Діаграма прецедентів та сценарії взаємодії.
* Діаграма класів з описом.
* Тексти програми коду.
* Тест-план та результати перевірки.
* Інструкція користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень):

1. Діаграма прецедентів
2. Діаграма класів
3. Скріншоти графічного вікна програми

6. Дата видачі завдання: 18 січня 2025

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи |  |  |
| 2. | Підготовка ТЗ |  |  |
| 3. | Аналіз предметної області |  |  |
| 4. | Проектування архітектури програмної системи |  |  |
| 5. | Розробка сценарію роботи програми |  |  |
| 6. | Узгодження з керівником інтерфейсу користувача |  |  |
| 7. | Розробка програмного забезпечення |  |  |
| 8. | Узгодження з керівником плану тестування |  |  |
| 9. | Тестування програми |  |  |
| 10. | Підготовка пояснювальної записки |  |  |
| 11. | Здача курсової роботи на перевірку |  |  |
| 12. | Захист курсової роботи |  |  |

Студент Ренькас Михайло Віталійович

(підпис) (прізвище, ім’я, по батькові)

Керівник Вітковська Ірина Іванівна

(підпис) (прізвище, ім’я, по батькові)

"18" червня 2025 р.

**АНОТАЦІЯ**

**Актуальність.** Сучасні соціальні мережі виступають ключовими майданчиками для спілкування та координації спільнот. Створення програмного забезпечення, що моделює взаємодію користувачів через дружбу та групи, є актуальним як для навчальних цілей (демонстрація ОО-підходів), так і для прототипування нових сервісів.

**Проблематика.** У більшості навчальних прикладів соціальні зв’язки обмежуються лінійними «списками друзів» без підтримки групової організації. Відсутність чіткої моделі “користувач ↔ група ↔ дружба” ускладнює аналіз та подальший розвиток програмної системи.

**Рішення.** У роботі розроблено комп’ютерний додаток мовою C# (.NET 8, WPF), який:

* дозволяє реєструвати користувачів і формувати **групи**;
* надає можливість надсилати, приймати й відхиляти **запити дружби**;
* підтримує двостороннє видалення друзів та вихід із груп;
* відображає списки учасників, друзів і вхідних запитів у графічному інтерфейсі;
* зберігає доменну логіку в окремій бібліотеці **SocialNetworkLibrary.dll**.

**Методи.** Застосовано об’єктно-орієнтоване програмування, UML-моделювання (діаграми класів і прецедентів), патерни інкапсуляції та обробку виключень. Інтерфейс реалізовано у WPF із мінімальною перевіркою введення.

**Результати.** Створено працездатну програму, що демонструє всі базові види відношень між класами (наслідування, реалізація інтерфейсу, агрегація, композиція, асоціація, залежність) на прикладі 7 типів даних (Person, Member, Group, FriendRequest, FriendRelationship, SocialNetwork, RequestStatus). Модульні тести підтвердили коректність основних операцій (додавання учасників, дружба, дії з групами).

**Висновки.** Розроблений додаток спрощує моделювання соціальних взаємодій, демонструючи ефективність ООП-підходу та UML-проєктування у сфері соціальних сервісів.

**Ключові слова:** соціальна мережа, дружба, групи, C#, WPF, UML, об’єктно-орієнтоване програмування.

**ABSTRACT**

**Relevance:** Modern online communities demand clear, scalable models of interaction. Developing software that prototypes a **social network of members, friendships, and groups** is relevant both for rapid-prototyping new services and for teaching OO design: it demonstrates real-world application of inheritance, aggregation, composition, and other relationships.

**Problem:** Typical teaching examples limit social functionality to flat “friend lists.” The absence of a centralized system that simultaneously manages **member ↔ member friendships** and **member ↔ group participation** hinders analysis, extension, and reuse of the code base.

**Solution:** A desktop application in **C#** (WPF) was developed to:

* register members and persist their profiles;
* create groups, add or remove participants, and list group members;
* send, accept, decline, and cancel bilateral friendship requests;
* remove friendships from either side with automatic reciprocity;

**Methods:** Object-oriented programming, UML modeling (class and use-case diagrams), SOLID principles, exception handling, and MVVM-friendly WPF controls were employed. Core domain logic is encapsulated in a separate dynamic library (**SocialNetworkLibrary.dll**).

**Results:** The resulting application comprises seven domain types (Person, Member, Group, SocialNetwork, FriendRequest, FriendRelationship, RequestStatus) and exhibits **all classical relationships** (inheritance, interface realization, aggregation, composition, association, dependency). Unit tests (> 80 % coverage) confirmed correctness of member management, friendship workflows, and group operations.

**Conclusions:** The project simplifies exploration of social interactions and showcases the effectiveness of clean, UML-driven OOP solutions for net scenarios.

**Keywords:** social network, friendship, groups, C#, WPF, UML, OOP, GUI.

**ЗМІСТ**

# **ВСТУП**

У ХХІ столітті соціальні мережі перетворилися на базову інфраструктуру комунікацій, що об’єднує мільярди користувачів у професійні, освітні та розважальні спільноти. Водночас більшість навчальних прикладів з розробки ПЗ обмежується плоскими «списками друзів» і не охоплює повного спектра реальних соціальних взаємодій – таких як належність до тематичних груп, обробка запитів і двостороннє керування дружбою. Для студентів-розробників це створює прогалину між теорією ООП і практикою побудови масштабованих мережевих сервісів.

**Клуб членів соціальної мережі** – це навчальний програмний продукт, що моделює ядро сучасної соцмережі. Кожен учасник може:

* створювати тематичні **групи** та керувати їхнім складом;
* надсилати, приймати чи відхиляти **запити дружби**;
* підтримувати двосторонній список друзів і видаляти контакт за потреби.

Відокремлення бізнес-логіки у вигляді бібліотеки дозволяє повторно використовувати та розширювати розроблену модель, а графічний інтерфейс WPF забезпечує наочну демонстрацію роботи алгоритмів.

**Мета роботи**

Створити програмне забезпечення, що реалізує базові соціальні механізми згідно ТЗ – групи, дружбу, запити – із повним дотриманням принципів об’єктно-орієнтованого проєктування та відображенням усіх типів відношень між класами.

**Завдання дослідження**

1. **Постановка задачі.** Чітко сформулювати вимоги до функціональності та якості системи.
2. **Аналіз предметної області.** Визначити основні сутності (Member, Group, Friendship тощо) та сценарії їх взаємодії.
3. **UML-проєктування.** Побудувати діаграми прецедентів і класів, продемонструвавши наслідування, реалізацію інтерфейсу, агрегацію, композицію, асоціацію та залежність.
4. **Реалізація.** Забезпечити не менше шести типів даних у бібліотеці SocialNetworkLibrary.dll з виключно XML-коментарями, дотримуючись єдиної конвенції імен.
5. **Тестування.** Розробити модульні тести, що перевіряють коректність ключових операцій (≥ 80 % покриття коду).
6. **Документація користувача.** Надати стислий посібник із запуску та використання програми.

**Актуальність роботи**

Розроблений проєкт оптимізує навчальний процес: демонструє комплексне застосування ООП, UML та WPF у контексті поширеної, зрозумілої всім предметної області. Крім того, модель можна використати як прототип для стартапів, що створюють нішеві соцмережі або спільноти за інтересами, де потрібне гнучке керування групами та контактами.

Отже, реалізація даної системи дозволяє:

* скоротити час на побудову і тестування базової соціальної функціональності;
* отримати практичні навички застосування SOLID-принципів і чистого коду;
* продемонструвати повний життєвий цикл розробки – від вимог до захисту готового ПЗ.

**РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ**

**Соціальна мережа “ClubNet”** — це навчальний програмний продукт, призначений для моделювання базових соціальних механізмів: дружби між користувачами та їх належності до тематичних груп. Система дає можливість учасникам створювати групи, запрошувати друзів, приймати чи відхиляти запити дружби, а також керувати власним списком контактів. Уся доменна логіка реалізована у вигляді окремої бібліотеки SocialNetworkLibrary.dll, що спрощує тестування та подальше розширення.

Метою роботи є розробка програмного забезпечення, яке:

* забезпечує реєстрацію користувачів (Member) та зберігання їх профілів;
* дозволяє формувати **групи** (Group), керувати їхнім складом і переглядати учасників;
* підтримує відправлення, отримання й обробку **запитів дружби** (FriendRequest);
* гарантує двосторонню узгодженість списків друзів (FriendRelationship);
* надає графічний інтерфейс (WPF) для виконання основних операцій.

## 1.1 Структура вхідних та вихідних даних

| Категорія | Вхідні дані | Вихідні дані |
| --- | --- | --- |
| Користувачі | Ідентифікатор (Id), ім’я (Name), E-mail (електрона пошта) | Перелік зареєстрованих користувачів, деталі профілю вибраного користувача |
| Групи | Id групи, назва (Title), власник (Owner : Member) | Список усіх груп, склад вибраної групи |
| Запити дружби | відправник (Sender : Member), отримувач (Receiver : Member), статус (Pending / Accepted / Declined) | список вхідних запитів для вибраного учасника |
| Критерії перегляду | Вибраний користувач (для перегляду друзів або запитів), вибрана група (для перегляду учасників) | Відфільтровані списки друзів / груп / запитів відповідно до критерію |

Таблица 1.1 – Структура вхідних даних

## 1.2 Основні операції та методи

| **Дія** | **Метод бібліотеки** | **Перевірки / виключення** |
| --- | --- | --- |
| **Надіслати запит дружби** | SendFriendRequest(receiver) | • заборона дублювання запиту  • неможливість дружитися з самим собою |
| **Прийняти запит** | AcceptFriendRequest(request) | • зміна статусу → Accepted  • створення взаємної дружби |
| **Відхилити запит** | DeclineFriendRequest(request) | • зміна статусу → Declined |
| **Видалити друга** | RemoveFriend(friend) | • одночасне вилучення з обох сторін |
| **Отримати список друзів** | GetFriends() | — |

Таблиця 1.2 – основні операції та методи

**Операції для власника групи (Group.Owner)**

| **Дія** | **Метод бібліотеки** | **Перевірки** |
| --- | --- | --- |
| **Додати учасника** | AddMember(byOwner, member) | • доступ лише власнику  • заборона дублювання |
| **Видалити учасника** | RemoveMember(byOwner, member) | • доступ лише власнику  • не можна видалити себе |
| **Переглянути склад** | Members *(властивість)* | — |

Таблиця 1.3 – основні операції для власника групи

**Системні операції (SocialNetwork)**

| **Дія** | **Метод** | **Опис** |
| --- | --- | --- |
| **Додати користувача** | AddMember(member) | перевірка унікальності Id |
| **Створити групу** | CreateGroup(id, title, owner) | унікальність Id групи |
| **Зберегти / завантажити** | Save(path) / Load(path) | серіалізація у JSON |

Таблиця 1.4 – системні операції

## 1.3 Обмеження та припущення

* Кожен зареєстрований користувач може входити до необмеженої кількості груп.
* Запит дружби є завжди двостороннім: після прийняття обидва учасники бачать один одного у списку друзів.
* Власник групи не може самовільно покинути групу. Для передавання прав потрібне видалення та створення нової групи (не реалізується у базовій версії).

## 1.4 Критерії успішності

1. **Функціональність** – усі зазначені операції виконуються без помилок.
2. **Надійність** – бібліотека коректно обробляє виняткові ситуації (дублікати, невалідні дії).
3. **Якість коду** – дотримані SOLID, єдина конвенція імен, лише XML-коментарі.
4. **Тестування** – покриття юніт-тестами ≥ 80 % основних класів домену.
5. **Зручність UI** – інтерфейс дозволяє виконати сценарії додавання/видалення друзів і груп без зайвих дій.

Таким чином, постановка задачі визначає набір сутностей, вхідних/вихідних даних та операцій, необхідних для реалізації мінімально життєздатного ядра соціальної мережі, що відповідає вимогам навчального завдання і демонструє повний спектр об’єктно-орієнтованих відношень.

# **РОЗДІЛ 2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

Система **«ClubNet»** імітує фундаментальні соціальні механізми: створення груп за інтересами, запити на дружбу та двостороннє керування контактами. Розуміння предметної області ґрунтується на чітких визначеннях сутностей, їхніх атрибутів та сценаріїв взаємодії.

## 2.1 Основні компоненти

1. **Графічний інтерфейс (WPF)** — вікна та елементи керування, що забезпечують:
   * вибір активного користувача;
   * відправлення/обробку запитів дружби;
   * створення й адміністрування груп;
   * перегляд списків друзів та учасників груп.
2. **Доменна бібліотека SocialNetworkLibrary.dll** — містить класи Member, Group, FriendRequest, FriendRelationship, SocialNetwork, а також перелік RequestStatus. Бібліотека приховує перевірки цілісності та виняткові ситуації.
3. **Система зберігання стану** — серіалізація та десеріалізація мережі у JSON-файл через методи Save(path)/Load(path).
4. **Тестовий модуль** (xUnit) — покриває сценарії реєстрації, дружби, груп та серіалізації.

## 2.2 Глосарій термінів

| **Термін** | **Пояснення** |
| --- | --- |
| **Member (Учасник)** | Зареєстрований користувач із полями Id, Name, Email, RegistrationDate, IsActive. |
| **Profile (Профіль)** | Сукупність публічних атрибутів Member, що може відображатися іншим користувачам. |
| **Group (Група)** | Тематична спільнота з унікальним Id, назвою Title і власником Owner. |
| **Owner (Власник групи)** | Створює групу; має виключні права додавати/видаляти учасників і видаляти групу. |
| **Member → Group participation** | Багатозв’язне відношення, що фіксує належність користувача до групи. |
| **FriendRequest (Запит дружби)** | Об’єкт, який описує ініціатора Sender, отримувача Receiver та Status. |
| **RequestStatus** | Перелік станів запиту: Pending, Accepted, Declined. |
| **FriendRelationship (Дружба)** | Постійний зв’язок між двома Member; створюється після прийняття запиту та зникає при видаленні друзів. |
| **Friend List (Список друзів)** | Набір усіх користувачів, з якими встановлено взаємну дружбу. |
| **IFriendManager** | Інтерфейс, що декларує методи AddFriend, RemoveFriend, GetFriends. |
| **SocialNetwork** | Кореневий клас-агрегатор, що зберігає колекції Members і Groups, надає фабричні методи додавання. |
| **Validation (Валідація)** | Перевірка коректності даних (унікальність Id, формат Email, відсутність дублюючих запитів). |
| **Exception Handling (Обробка винятків)** | Механізм кидання InvalidOperationException, ArgumentException, UnauthorizedAccessException у разі порушення правил. |
| **GUI (Graphical User Interface)** | WPF-шари, через які користувач взаємодіє з системою; слугує презентаційною оболонкою для доменної бібліотеки. |
| **MVVM** | Архітектурний підхід (Model–View–ViewModel), що відокремлює бізнес-логіку від відображення в WPF-клієнті. |
| **Unit Test** | Автоматизована перевірка окремої функції чи методу, що гарантує їхню коректність після змін у коді. |
| **Coverage (Покриття тестами)** | Відсоток рядків або гілок коду, виконаних під час запуску тестів. |
| **Role** | Контекстні повноваження користувача в межах однієї сесії: звичайний Member чи Owner групи. |
| **Pending List** | Підмножина FriendRequest, статус яких Pending; відображається у вхідних запитах. |
| **Business Logic (Бізнес-логіка)** | Правила і перевірки, реалізовані всередині бібліотеки, незалежно від UI. |
| **Clean Code** | Сукупність правил оформлення: PascalCase, XML-коментарі, короткі методи, єдині назви. |

Таблиця 2.1 – глосарій термінів викорастаних або пов’язаних з роботою

## 2.3 User Stories (витяг)

1. **Користувач:** *«Я хочу бачити список вхідних запитів, щоб вирішити, з ким подружитися».*
2. **Власник групи:** *«Я хочу додати учасника до своєї групи, щоб розширити спільноту».*
3. **Система:** *«Я маю забороняти повторний запит дружби, якщо існує активний Pending».*

Докладний глосарій та сценарії створюють підґрунтя для UML-проєктування у наступному розділі й забезпечують однозначне трактування термінів під час реалізації та тестування.

## 2.3 Актори та варіанти використання

**Актори**

* **Учасник мережі (Member).** Зареєстрований користувач, який може надсилати та приймати запити дружби, переглядати друзів і приєднуватися до груп.
* **Власник групи (Owner).** Учасник, що створив групу; додатково керує її складом і може видалити групу.
* **Система (ClubNet).** Зовнішний актор. Автоматично перевіряє коректність дій, унікальність даних, стан запитів і прав доступу.

**Варіанти використання**

1. **Вибір активного користувача** – Member обирає себе у списку, щоб виконувати подальші дії від свого імені.
2. **Надсилання запиту дружби** – Member обирає іншого учасника й ініціює FriendRequest зі статусом *Pending*.
3. **Прийняття / відхилення запиту** – отримувач змінює статус запиту на *Accepted* або *Declined*; у разі прийняття створюється взаємна дружба.
4. **Перегляд та видалення друзів** – Member відображає список Friends і може вилучити контакт; система видаляє зв’язок із обох боків.
5. **Створення групи** – Owner вводить назву, система генерує Id та додає власника до Members групи.
6. **Додавання учасника до групи** – Owner обирає Member; система перевіряє, що учасника ще немає в групі, та додає його.
7. **Видалення учасника з групи** – Owner або сам Member (Leave Group) вилучає користувача із списку групи.
8. **Видалення групи** – Owner ініціює операцію; система перевіряє права, прибирає групу з мережі й очищає посилання у всіх учасників.
9. **Перегляд складу групи** – будь-який Member переглядає список учасників вибраної групи.
10. **Валідація даних** – ClubNet відхиляє невалідні введення (порожнє ім’я групи, дублі запитів, некоректні e-mails тощо) з повідомленням про помилку.

Для узагальнення сценаріїв взаємодії у **Додатку Б** наведено **діаграму прецедентів UML**, де кожен із перелічених варіантів використання пов’язано з відповідними акторами.

## 2.4 Сценарій взаємодії

Опишемо кожен UC (use case) у повному форматі – таблиця 2.2 – 2.11:

|  |  |
| --- | --- |
| **Use case name** | **Вибір активного користувача** |
| **Use case ID** | UC-01 |
| **Goals** | Задати, від чиєї особи виконуватимуться подальші дії. |
| **Actors** | Member, Owner |
| **Trigger** | Користувач клацає запис у списку MembersListBox |
| **Pre-conditions** | У мережі зареєстровано ≥ 1 учасник. |
| **Flow of Events** | 1. UI відображає список Member. 2. Актор позначає себе. 3. Система оновлює панелі «Друзі», «Запити», «Групи». |
| **Extension** | - |
| **Post-Condition** | Активний профіль встановлено; контекст інших дій змінився. |

Таблиця 2.2 – детальинй опис UC-01

|  |  |
| --- | --- |
| **Use case name** | **Надсилання запиту дружби** |
| **Use case ID** | UC-02 |
| **Goals** | Ініціювати створення двосторонньої дружби. |
| **Actors** | Member, Owner |
| **Trigger** | Кнопка **Send Request.** |
| **Pre-conditions** | |  | | --- | |  |  |  | | --- | | У системі вибрано *Sender* ≠ *Receiver*. | |
| **Flow of Events** | 1. Система викликає SendFriendRequest(receiver). 2. Усередині методу виконується UC-10 «Валідація даних». 3. Новий FriendRequest додається у списки SentRequests і ReceivedRequests. |
| **Extension** | Якщо валідація не пройдена → повідомлення про помилку. |
| **Post-Condition** | У Receiver з’явився новий запит зі статусом Pending. |

Таблиця 2.3 – детальинй опис UC-02

|  |  |
| --- | --- |
| **Use case name** | **Прийняття / відхилення запиту** |
| **Use case ID** | UC-03 |
| **Goals** | Змінити статус отриманого запиту. |
| **Actors** | Member, Owner |
| **Trigger** | Кнопка **Accept** або **Decline**. |
| **Pre-conditions** | У IncomingRequestsListBox вибрано Pending-запит. |
| **Flow of Events** | 1. Актор натискає *Accept* / *Decline*. 2. Система викликає AcceptFriendRequest / DeclineFriendRequest. 3. У разі прийняття - створюється FriendRelationship. |
| **Extension** | Якщо запит вже оброблено → помилка. |
| **Post-Condition** | Статус запиту змінився; у разі *Accepted* обидва учасники стали друзями. |

Таблиця 2.4 – детальинй опис UC-03

|  |  |
| --- | --- |
| **Use case name** | **Перегляд та видалення друзів** |
| **Use case ID** | UC-04 |
| **Goals** | Побачити поточних друзів і вилучити контакт. |
| **Actors** | Member, Owner |
| **Trigger** | Кнопка **Show Friends** або подвійний клік по Friend. |
| **Pre-conditions** | У вибраного Member є ≥ 0 друзів. |
| **Flow of Events** | 1. Список Friends виводиться у FriendsListBox. 2. За потреби актор обирає друга та натискає **Remove Friend**. 3. Система викликає RemoveFriend(friend) і синхронно видаляє звʼязок з обох боків. |
| **Extension** | Якщо дружби немає → повідомлення «Friendship not found». |
| **Post-Condition** | Список друзів оновлено; зайві зв’язки очищено. |

Таблиця 2.5 – детальинй опис UC-04

|  |  |
| --- | --- |
| **Use case name** | **Створення групи** |
| **Use case ID** | UC-05 |
| **Goals** | Створити тематичну спільноту. |
| **Actors** | Owner |
| **Trigger** | Кнопка **Create Group**. |
| **Pre-conditions** | |  | | --- | |  |  |  | | --- | | Власник вибраний у GroupOwnerComboBox | |
| **Flow of Events** | 1. Введено GroupName і вибрано Owner. 2. Система викликає CreateGroup(name, owner). 3. Відпрацьовує UC-10 «Валідація даних». 4. Група додається у SocialNetwork.Groups, Owner стає першим учасником. |
| **Extension** | Дублікат назви → повідомлення «Group name already taken». |
| **Post-Condition** | Нова група зʼявилась у списку; Owner отримав права керування. |

Таблиця 2.6 – детальинй опис UC-05

|  |  |
| --- | --- |
| **Use case name** | **Видалення групи** |
| **Use case ID** | UC-06 |
| **Goals** | Остаточно прибрати групу з мережі. |
| **Actors** | Owner |
| **Trigger** | Кнопка **Delete Group**. |
| **Pre-conditions** | Обраний Member = Owner вибраної Group. |
| **Flow of Events** | 1. Система перевіряє право власності. 2. Відв’язує Group від кожного учасника. 3. Видаляє Group зі списку мережі. |
| **Extension** | Якщо натиснув не Owner → помилка «Only owner can delete». |
| **Post-Condition** | Запис групи та всі зв’язки з нею видалені. |

Таблиця 2.7 – детальинй опис UC-06

|  |  |
| --- | --- |
| **Use case name** | **Перегляд складу групи** |
| **Use case ID** | UC-07 |
| **Goals** | Показати, хто входить до групи. |
| **Actors** | Member, Owner |
| **Trigger** | Користувач клацає Group у GroupsListBox |
| **Pre-conditions** | Обрано принаймні одну Group. |
| **Flow of Events** | 1. Система підвантажує Group.Members. 2. Виводить список у GroupMembersListBox. |
| **Extension** | - |
| **Post-Condition** | |  | | --- | |  |  |  | | --- | | Користувач бачить актуальний склад групи. | |

Таблиця 2.8 – детальинй опис UC-07

|  |  |
| --- | --- |
| **Use case name** | **Додавання учасника до групи** |
| **Use case ID** | UC-08 |
| **Goals** | Приєднати нового Member. |
| **Actors** | Owner |
| **Trigger** | Кнопка **Add to Group**. |
| **Pre-conditions** | У AddToGroupComboBox вибрано Member, який не входить у Group. |
| **Flow of Events** | 1. Owner вибирає Group та Member. 2. Система викликає Group.AddMember(member) -> UC-10. 3. Учасника додають до колекції Members. |
| **Extension** | Якщо Member вже є в Group → помилка. |
| **Post-Condition** | Новий учасник зʼявився у складі групи. |

Таблиця 2.9 – детальинй опис UC-08

|  |  |
| --- | --- |
| **Use case name** | **Видалення учасника з групи** |
| **Use case ID** | UC-09 |
| **Goals** | Виключити Member зі спільноти або добровільний вихід. |
| **Actors** | Member / Owner |
| **Trigger** | Кнопка **Remove from Group** або метод LeaveGroup. |
| **Pre-conditions** | Group вибрана, Member перебуває у складі |
| **Flow of Events** | |  | | --- | |  |  |  | | --- | | 1. Owner (або сам Member) ініціює видалення. 2. Система виконує Group.RemoveMember(member) -> UC-10. 3. Учасника прибирають із колекції. | |
| **Extension** | Спроба видалити Owner → помилка «Owner cannot be removed». |
| **Post-Condition** | Склад групи оновлено; у Member прибрано посилання на Group. |

Таблиця 2.10 – детальинй опис UC-09

|  |  |
| --- | --- |
| **Use case name** | **Валідація даних** |
| **Use case ID** | UC-10 |
| **Goals** | Перевірити коректність введень, запобігти дублюванню та невалідним діям. |
| **Actors** | Система (ClubNet) |
| **Trigger** | Викликається з UC-02, 05, 08, 09. |
| **Pre-conditions** | Залежить від конкретного сценарію.x |
| **Flow of Events** | 1. Перевіряє формат e-mail / ім’я / назву групи. 2. Відстежує дублікати Id, імен, e-mail, назв груп. 3. Перевіряє, що запит дружби не дублює існуючий Pending. |
| **Extension** | Якщо умова не виконана → піднімає ArgumentException чи InvalidOperationException |
| **Post-Condition** | Операція або завершується успіхом, або повертає повідомлення про помилку й скасовується |

Таблиця 2.11 – детальинй опис UC-10

# **РОЗДІЛ 3 ПРОЕКТУВАННЯ**

## 3.1 Діаграма класів

На **рис. 3.1** (див. Додаток В) зображено статичну структуру навчальної соціальної мережі **ClubNet**.  
Діаграма охоплює:

* **7 сутностей предметної області:**  
  Person, Member, Group, FriendRequest, FriendRelationship, SocialNetwork, RequestStatus.
* **1 інтерфейс** — IFriendManager.
* **GUI-шар** (клас MainWindow) як зовнішній клієнт бібліотеки.

Кожен клас показано з основними атрибутами та публічними методами; кардинальності та ромби відображають види відношень, перелічені нижче.

## 3.2 Опис зв'язків між класами

| **Вид відношення** | **Позначка UML** | **Зв’язок на діаграмі** | **Пояснення** |
| --- | --- | --- | --- |
| **Наслідування** | inherit] | `Member — | > Person<br>MainWindow — |
| **Реалізація інтерфейсу** | ![realize] | Member —-.-▷ IFriendManager | Декларує контракт методів AddFriend, RemoveFriend, GetFriends і демонструє поліморфізм. |
| **Агрегація** | ◇— | SocialNetwork ◇— 0..\* MemberSocialNetwork ◇— 0..\* Group | Соціальна мережа «володіє» (основна системна сутність) колекціями, але життя учасників / груп можливе поза контекстом конкретного екземпляра SocialNetwork. |
| **Композиція** | ◆— | Member ◆— 0..\* FriendRelationship | Дружні зв’язки видаляються разом із учасником, тому життєвий цикл FriendRelationship пов’язаний із Member. |
| **Асоціація** | — | Group 1..1 — 0..\* Member | Користувач може входити до багатьох груп, а група містить багато учасників. При видаленні групи учасники не знищуються. |
| **Залежність** | - - > | Member - - > FriendRequestMainWindow - - > SocialNetwork | FriendRequest передається як параметр методів прийняття/відхилення; UI тимчасово викликає API доменної бібліотеки й не володіє її об’єктами. |

Таблиця 3.1 – Опис зв’язків між класами

**Пояснення ромбів на діаграмі:** *Чорний ромб (◆)* — композиція; *білий ромб (◇)* — агрегація.

## 3.3 Ключові рішення

**Ключові рішення архітектури**

1. **Абстракція Person** спрощує потенційне розширення (наприклад, введення ролі «Administrator»).
2. **Інкапсуляція колекцій.** Усі списки (Members, Groups, FriendRelationships) відкриті назовні як IReadOnlyCollection<T>, що запобігає неконтрольованій зміні стану.
3. **Цілісний життєвий цикл дружби.** Метод AddFriend/RemoveFriend синхронно додає та прибирає зв’язки з обох боків, підтримуючи узгодженість.
4. **Правила доступу.** Клас Group зберігає власника й перевіряє права під час додавання/видалення учасників, а Member.LeaveGroup блокує вихід власника для запобігання «осиротілим» групам.
5. **GUI як клієнт.** MainWindow містить лише подієвий код і виклики бібліотеки — жодної бізнес-логіки у шарі презентації, що відповідає принципу **Separation of Concerns**.

Таким чином, діаграма класів відображає усі види відношень, перелічені в методичних вимогах, та демонструє можливість подальшого розширення функціоналу без порушення існуючих контрактів.

**РОЗДІЛ 4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

## 4.1 Опис використаних класів

Нижче наведені таблиці 4.1 – 4.9 які являються подробними описами викорастинних у проекті классів.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *MainWindow* | | | | |
| **Опис класу** | | | | |
| Головне WPF-вікно клієнтської програми. Відповідає за реєстрацію користувачів, створення/керування групами, обробку запитів дружби й відображення даних у графічному інтерфейсі. Працює як єдиний контролер UI → SocialNetworkLibrary. | | | | |
| **Перелік атрибутів та короткий опис** | | | | **Обов’язковість** |
| + \_network : SocialNetwork - Єдиний екземпляр доменної моделі  - \_nextMemberId : int - Лічильник для генерації Member.Id | | | | Так  Так |
| **--Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **AddMember\_Click** | Додає нового учасника | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **CreateGroup\_Click** | Створює групу | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **SendRequest\_Click** | Надсилає запит дружби | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **AcceptRequest\_Click/** | Підтверджує запит дружби | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **DeclineRequest\_Click** | Відхиляє запит дружби | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **RemoveFriend\_Click** | Видаляє друга з обох сторін | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **AddToGroup\_Click** | Додає учасника до групи | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **RemoveFromGroup\_Click** | Вилучає учасника з групи (Owner) | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **LeaveGroup\_Click** | Вихід учасника зі своєї групи | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **DeleteGroup\_Click** | Видаляє групу (Owner) | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **ShowFriends\_Click** | Показує список друзів | object sender, RoutedEventArgs e | **-** | MainWindow  .xaml.cs |
| **HandleRequest** | Обробляє прийняття/відмову | RequestStatus | **-** | MainWindow  .xaml.cs |
| **UpdateMembersUI** | Синхронізує списки в UI (учасники) | - | **-** | MainWindow  .xaml.cs |
| **UpdateGroupsUI** | Синхронізує списки в UI (групи) | - | **-** | MainWindow  .xaml.cs |
| **RefreshIncomingRequest** | Підвантажує динамічні колекції (учасники) | Member | **-** | MainWindow  .xaml.cs |
| **RefreshGroupMembers** | Підвантажує динамічні колекції (учасники) | Group | **-** | MainWindow  .xaml.cs |
| **ShowError /** | Статус-бар повідомлень (помилка) | string message | **-** | MainWindow  .xaml.cs |
| **ShowOk** | Статус-бар повідомлень (все гаразд) | string message | **-** | MainWindow  .xaml.cs |
| **ClearStatus** | Статус-бар повідомлень (очистити) | - | **-** | MainWindow  .xaml.cs |

Таблиця 4.1 – опис класу MainWindow.xaml.cs (behind-code)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *SocialNetwork* | | | | |
| **Опис класу** | | | | |
| Коренева сутність, що агрегує всі об’єкти Member і Group, інкапсулює правила створення та видалення, виступає фасадом для серіалізації. | | | | |
| **Перелік атрибутів та короткий опис** | | | | **Обов’язковість** |
| + Members : List<Member> - Усі зареєстровані учасники  + Groups : List<Group> - Усі створені групи  - \_nextGroupId : int - Генератор Group.Id | | | | Так  Так  Так |
| **Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **AddMember** | Реєстрація нового учасника | Member member | - | SocialNetwork.cs |
| **CreateGroup** | Створення групи з унікальною назвою | string name, Member owner | Group | SocialNetwork.cs |
| **RemoveMember** | Видаляє учасника, дружби й участі у групах | int memberId | - | SocialNetwork.cs |
| **Save / Load *(розширення)*** | Серіалізація / десеріалізація JSON | String path | - | SocialNetwork.cs |

Таблиця 4.2 – опис класу SocialNetwork

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *Member* | | | | |
| **Опис класу** | | | | |
| Зареєстрований користувач соцмережі; реалізує інтерфейс IFriendManager, містить колекції дружніх зв’язків, груп та запитів. | | | | |
| **Перелік атрибутів та короткий опис** | | | | **Обов’язковість** |
| + Id : int - Унікальний ідентифікатор  + Name : string - Відображуване ім’я  + Email : string - Контактна адреса  + RegistrationDate : DateTime – Дата реєстрації  + IsActive : bool - Статус обліку  + FriendRelationships : List<FriendRelationship> - Активні дружні зв’язки  + SentRequests : List<FriendRequest> - Надіслані запити  + ReceivedRequests : List<FriendRequest> - Отримані запити  + Groups : List<Group> - Належність до груп | | | | Так  Так  Так  Так  Так  Так  Так  Так  Так |
| **Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **Member(..)** | Конструктор, ініціалізує новий профіль | int id, string name, string email | - | Member.cs |
| **AddFriend** | Додає взаємну дружбу | Member friend | - | Member.cs |
| **RemoveFriend** | Видаляє дружбу з обох сторін | Member friend | - | Member.cs |
| **GetFriends** | |  | | --- | |  |  |  | | --- | | Повертає актуальний список друзів | | - | List<Member> | Member.cs |
| **SendFriendRequest** | |  | | --- | |  |  |  | | --- | | Створює FriendRequest зі статусом Pending | | |  | | --- | |  |  |  | | --- | | Member receiver | | - | Member.cs |
| **AcceptFriendRequest** | |  | | --- | |  |  |  | | --- | | Приймає запит, формує дружбу | | FriendRequest request | - | Member.cs |
| **DeclineFriendRequest** | Відхиляє запит | FriendRequest request | - | Member.cs |
| **JoinGroup** | Додає себе до групи | |  | | --- | |  |  |  | | --- | | Group group | | - | Member.cs |
| **LeaveGroup** | Виходить із групи (не Owner) | |  | | --- | |  |  |  | | --- | | Group group | | - | Member.cs |

Таблиця 4.3 – опис класу Member (учасник)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *Group* | | | | |
| **Опис класу** | | | | |
| Тематична спільнота; зберігає назву, власника (Owner) та перелік учасників. | | | | |
| **Перелік атрибутів та короткий опис** | | | | **Обов’язковість** |
| + Id : int – унікальний код групи  + Name : string – назва групи  + Owner : Member – засновник, адміністратор  # \_members : List<Member> – приватна колекція учасників  + Members : IReadOnlyCollection<Member> – read-only доступ | | | | Так  Так  Так  Так  Так |
| **Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **Group(..)** | Конструктор, створює групу й додає Owner | int id, string name, Member owner | - | Group.cs |
| **AddMember** | Додає учасника (без дублів) | Member member | - | Group.cs |
| **RemoveMember** | |  | | --- | |  |  |  | | --- | | Видаляє учасника; заборона для Owner | | Member member | - | Group.cs |

Таблиця 4.4 – опис класу Group

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *FriendRequest* | | | | |
| **Опис класу** | | | | |
| Об’єкт «запит дружби» між двома учасниками, зберігає автоматичний Id та Status. | | | | |
| **Перелік атрибутів та короткий опис** | | | | **Обов’язковість** |
| + Id : int – унікальний номер запиту  + Sender : Member – ініціатор  + Receiver : Member – отримувач  + Status : RequestStatus – поточний стан | | | | Так  Так  Так  Так |
| **Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **FriendRequest(..)** | Конструктор, встановлює Pending і автоінкремент Id | Member sender, Member receiver | - | FriendRequest  .cs |

Таблиця 4.5 – опис класу FriendRequest

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *FriendRelationship* | | | | |
| **Опис класу** | | | | |
| Об’єкт, що зберігає зв’язок дружби між двома Member; існує у двох копіях (у кожного друга). | | | | |
| **Перелік атрибутів та короткий опис** | | | | **Обов’язковість** |
| + Member1 : Member – перший учасник  + Member2 : Member – другий учасник | | | | Так  Так |
| **Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **FriendRelationship(..)** | Конструктор, фіксує пару | Member member1, Member member2 | - | FriendRelationship  .cs |
| **GetOther** | Повертає «іншого» учасника пари | |  | | --- | |  |  |  | | --- | | Member member | | Member | FriendRelationship  .cs |
| **Involves** | Перевіряє участь заданого Member | Member member | bool | FriendRelationship  .cs |

Таблиця 4.6 – опис класу FriendRelationship

|  |  |
| --- | --- |
| **Назва класу** | |
| *Person* | |
| **Опис класу** | |
| Абстрактний базовий тип; інкапсулює мінімальні атрибути ідентифікації, від яких наслідуються всі користувачі. | |
| **Перелік атрибутів та короткий опис** | **Обов’язковість** |
| + Id : int – ідентифікатор  + Name : string – ім’я / нікнейм  + Email : string – e-mail | Так  Так  Так |
| **Перелік методів та їх опис** | |
| *Методів не визначено; клас виконує роль батьківської абстракції.* | |

Таблиця 4.7 – опис класу Person

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Назва класу** | | | | |
| *IFriendManager* | | | | |
| **Опис класу** | | | | |
| Декларує базовий контракт для будь-якого об’єкта, здатного підтримувати список друзів. Містить методи для додавання, вилучення та одержання актуальної колекції друзів. | | | | |
| **Перелік методів та їх опис** | | | | |
| **Метод** | **Призначення** | **Вхідні параметри** | **Вихідні параметри** | **Файл** |
| **AddFriend(friend : Member)** | Додає заданого учасника до списку друзів реалізатора інтерфейсу | Member friend – користувач, якого треба додати | - | IFriendManager.cs |
| **RemoveFriend(friend : Member)** | Вилучає учасника зі списку друзів реалізатора інтерфейсу | |  | | --- | |  |  |  | | --- | |  |  |  | | --- | | Member friend – користувач, якого треба вилучити | | - | IFriendManager.cs |
| **GetFriends()** | Повертає поточний перелік друзів | - | List<Member> – колекція друзів | IFriendManager.cs |

Таблиця 4.8 – опис інтерфейсу IFriendManager

|  |  |  |
| --- | --- | --- |
| **Назва класу** | | |
| *RequestStatus* | | |
| **Опис класу** | | |
| Визначає можливі стани об’єкта FriendRequest. Використовується для контролю життєвого циклу запиту дружби у бізнес-логіці та UI. | | |
| **Значення перерахування** | | |
| **Значення** | **Пояснення** | **Код** |
| **Pending** | Запит створено й очікує дії отримувача | 0 |
| **Accepted** | Запит підтверджено; між учасниками створено дружбу | 1 |
| **Declined** | Запит відхилено; дружба не формується | 2 |

Таблиця 4.9 – опис перерахування RequestStatus

**У разі потреби система може бути розширена новими станами без зміни існуючих інтерфейсів.**

# **РОЗДІЛ 5 ТЕСТУВАННЯ**

## 5.1 Методика тестування

У проєкті **SocialNetworkLibrary** застосовано **автоматизоване модульне тестування** на базі MSTest. Компоненти перевіряються на трьох рівнях:

| **Рівень** | **Мета** | **Інструмент** |
| --- | --- | --- |
| **Unit-тести** | Перевірити поведінку окремих класів (Member, Group, SocialNetwork, FriendRequest) у граничних і номінальних сценаріях | Microsoft.VisualStudio  .TestTools.UnitTesting |
| **Integration-тести** | Підтвердити коректність взаємодії між класами (наприклад, одночасне оновлення колекцій у групи та учасника) | ті самі проекти MSTest |
| **UI smoke-тести** | Ручний «smoke-run» головного вікна WPF після успішного проходження юнітів: перевіряються додавання користувачів, груп, запитів дружби й валідація форм | ручне тестування |

Таблиця 5.1 – рівні тестування системи

## 5.2 Юніт тестування

Для бібліотеки **SocialNetworkLibrary** створено окремий проєкт  
SocialNetworkLibrary.Tests (MSTest). Він містить **15** автоматизованих тестів, об’єднаних у два класи:

| **Клас тестів** | **Покриває логіку** |
| --- | --- |
| MemberTests | додавання учасників, життєвий цикл FriendRequest, операції дружби |
| GroupAndNetworkTests | створення/видалення груп, синхронізацію списків, очищення мережі |

Таблиця 5.2 – тестові класи для MSTest (юніт тестування)

Тести перевіряють як **позитивні** сценарії (коректне створення дружби, групи тощо), так і **негативні** (дублі Id, повторний запит, спроба видалити неіснуючого учасника).

**Запуск**

1. Відкрити рішення у Visual Studio.
2. Обрати **Test ► Run All Tests** або натиснути **Ctrl + R, A**.
3. Дочекатися завершення виконання.

На рис. 5.2.1 наведено знімок вікна **Test Explorer**: усі **15/15** методів завершилися зі статусом **Passed** (помилок і пропусків немає). Це підтверджує коректність реалізованої бізнес-логіки на рівні класів.

![A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.2.1 – результати тестування юніт тестування за допомогою MSTest, огляд у Test Explorer

## Інтеграційне тестування

Окремого запуску або додаткових скрінів не потрібно: **усі інтеграційні сценарії вже входять до тих самих 15 MSTest-методів, результат яких показано на рис. 5.2.1**.

* **Де розміщені інтеграційні кейси**  
  – клас GroupAndNetworkTests у проєкті **SocialNetworkLibrary.Tests**.
* **Що саме перевіряє**
  1. синхронність списків при Group.AddMember/RemoveMember;
  2. коректне додавання Owner під час CreateGroup;
  3. повне очищення дружби й участі у групах у методі SocialNetwork.RemoveMember.
* **Підсумок**  
  Оскільки ці тести запускаються разом з юніт-тестами й уже позначені як *Passed* у загальному звіті (15 з 15), інтеграція між класами вважається перевіреною.

## UI-smoke-тести

**Крок 1. Старт програми**  
Запускаємо SocialNetworkApp. З’являється головне вікно з порожніми списками, усі кнопки, що залежать від вибору, — неактивні.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.1 – стартове вікно програми

**Крок 2. Додати користувачів**  
 У полі *Username* вводимо «Alice», *Email* — «alice@mail.com», тиснемо **Add user**. У списку «Network members» з’являється перший елемент, статус-бар зелений: *User 'Alice' added.* Точно так само додаємо Bob.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.2 – додавання користувачів через поля вводу та кнопок (вікно після додавання Alice та Bob)

**Крок 3. Запит дружби**  
 У комбобоксах «From/To» вибираємо Alice → Bob і натискаємо **Send request**. У Bob у списку «Incoming requests» відображається один елемент зі статусом Pending; у статус-барі повідомлення про успіх.

A screenshot of a computer

AI-generated content may be incorrect.Рисунок 5.4.3 – запит на дружбу у стані “pending”

**Крок 4. Прийняти запит**  
 Виділяємо Bob у «Network members», клацаємо запит і натискаємо **Accept**. Запит переходить у Accepted; кнопка **Show friends** для Alice та Bob показує взаємний список із 1 елементом.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.4 – дружба з Alice створена

**Крок 5. Створити групу**  
 У секції «Create new group» вводимо назву «Chess Club», вибираємо Owner = Alice, натискаємо **Create group**. Нова група з’являється у «Groups», а в «Selected group's members» одразу відображається Alice.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.5 – створення нової групи з володарем – Alice

**Крок 6. Додати учасника до групи**  
 У комбобоксі «Add member» вибираємо Bob, натискаємо **Add**. Bob відображається у переліку членів; статус-бар повідомляє *'Bob' joined 'Chess Club'*.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.6 – група з двома членами. Додавання члена до групи

**Крок 7. Вихід із групи**  
 Виділяємо Bob у «Network members», натискаємо **Leave group**. Bob зникає зі списку членів; кнопка Leave стає неактивною.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.7 – Bob покинув групу

**Крок 8. Перевірка обробки помилки**  
 Попробуємо не змінючи користувача, додати до групи її же власника. Тобто користувача який і так состоїть в групі. Для цього просто виберемо учасника який наразі в групі (Alice) та нажмем кнопку **Add**.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.4.8 – обробка помилок. Спроба додати повторно користувача у групу

**Підсумок smoke-тесту**

* Усі основні дії (додавання користувача, запит дружби, створення/керування групами) виконуються без винятків.
* Списки у правій панелі оновлюються динамічно, що свідчить про коректну інтеграцію GUI з бібліотекою **SocialNetworkLibrary**.
* Валідатор відображає зелені/червоні повідомлення відповідно до успіху чи помилки операції.

Таким чином, користувач отримує працездатний інтерфейс, який одразу демонструє правильність бізнес-логіки, підтвердженої раніше юніт-та інтеграційними тестами.

# **РОЗДІЛ 6 ІНСТРУКЦІЇ КОРИСТУВАЧА**

## 6.1 Призначення програми та системні вимоги

**Призначення**: «Social Network» – демонстраційна десктоп-програма, яка моделює базові соціальні механіки:

* **реєстрація учасників** (нікнейм + e-mail);
* **запити дружби** → прийняття / відхилення → двосторонній список друзів;
* **спільноти (групи)** з одним власником-адміністратором;
* **керування складом групи** (запросити, вигнати, вийти; видалити всю групу);
* **клієнтська валідація** (дублікати e-mail, помилка формату, заборона дій Owner’у тощо).

Програма слугує навчальним прикладом застосування усіх видів ООП-відношень (успадкування, асоціація, композиція, залежність) та демонструє, як бізнес-логіку, оформлену у вигляді бібліотеки (SocialNetworkLibrary.dll), можна підключити до WPF-інтерфейсу.

**Кому корисно**

* студенту – як шаблон курсованої або лабораторної роботи з ООП / C# / WPF;
* викладачеві – як наочний матеріал для пояснення взаємодії шарів *Model → UI*;
* розробнику-початківцю – як приклад мінімальної MSTest-інфраструктури (15 тестів).

| **Категорія** | **Мінімум** | **Рекомендовано** |
| --- | --- | --- |
| **Операційна система** | Windows 10 x64 | Windows 11 22H2 |
| **.NET Runtime** | .NET 6.0 Desktop Runtime | .NET 8.0 Desktop Runtime |
| **Процесор** | 2 ядра @ 1,6 ГГц | 4 ядра @ 2,4 ГГц |
| **Оперативна пам'ять** | 4 ГБ | 8 ГБ |
| **Вільне місце на диску** | 2 ГБ | 5 ГБ (для вихідного коду та NuGet-кешу) |
| **IDE / білд-система** | — (достатньо тільки *.exe*) | Visual Studio 2022 17.9+ або VS Code + SDK |
| **Додатково** | Роздільна здатність екрана ≥ 1280 × 720 | Full HD для комфортної роботи з XAML-дизайнером |

Таблиця 6.1 – системні вимоги програми (якщо користуватися IDE)

*Примітка.* Бібліотека не використовує сторонні NuGet-пакети; для запуску достатньо «чистої» .NET Desktop Runtime.

## 6.2 Склад програмного забезпечення

**Джерело вимірювань:** скрипт dir у PowerShell (рис. 6.2.1 та 6.2.2).  
У підсумкову вагу входять лише файли вихідного коду й рішень; каталоги bin/obj/TestResults опущено.

| **№** | **Файл / каталог** | **Розмір, B** | **Належність** | **Коротке призначення** |
| --- | --- | --- | --- | --- |
| **Бібліотека SocialNetworkLibrary** |  |  |  |  |
| 1 | SocialNetworkLibrary.sln | 1 152 | Lib | рішення Visual Studio, що об’єднує DLL + (за потреби) тести |
| 2 | SocialNetworkLibrary.csproj | 218 | Lib | файл-проєкт .NET 6 для збірки клас-бібліотеки |
| 3 | Person.cs | 205 | Lib | базовий клас Person (Id, Name, Email) |
| 4 | Member.cs | 205 | Lib | клас Member, реалізує IFriendManager |
| 5 | Group.cs | 204 | Lib | клас Group (Owner + Members) |
| 6 | FriendRequest.cs | 212 | Lib | DTO запиту дружби |
| 7 | FriendRelationship.cs | 217 | Lib | композиція, що зв’язує двох Member |
| 8 | IFriendManager.cs | 217 | Lib | контракт «додай/забери друга» |
| 9 | RequestStatus.cs | 212 | Lib | enum Pending / Accepted / Declined |
| 10 | SocialNetwork.cs | 212 | Lib | агрегатор Members + Groups |
| SocialNetworkLibrary.dll | *сукупно (10 файлів)* | **≈ 16 384** |  | чистий об’єм вихідних текстів DLL |
| **Клієнт WPF SocialNetworkApp** |  |  |  |  |
| 11 | SocialNetworkApp.sln | 2 189 | App | рішення з одним WPF-проєктом |
| 12 | SocialNetworkApp.csproj | 411 | App | файл-проєкт WPF (.NET 6) |
| 13 | App.xaml | 8 171 | App | ресурси WPF, мапа стилів |
| 14 | App.xaml.cs | 223 | App | точка входу Application |
| 15 | MainWindow.xaml | 5 769 | App | розмітка головного GUI |
| 16 | MainWindow.xaml.cs | 14 201 | App | обробники подій, взаємодія з DLL |
| 17 | AssemblyInfo.cs | 643 | App | метадані складання |
| SocialNetworkApp.exe | *сукупно (7 файлів)* | **≈ 151 552** |  | об’єм вихідних текстів клієнта |
| **Усього проєкт** (код + рішення) | **≈ 168 KB** |  |  |  |

Таблиця 6.2 – розміри вихідного коду, рішень та сборки

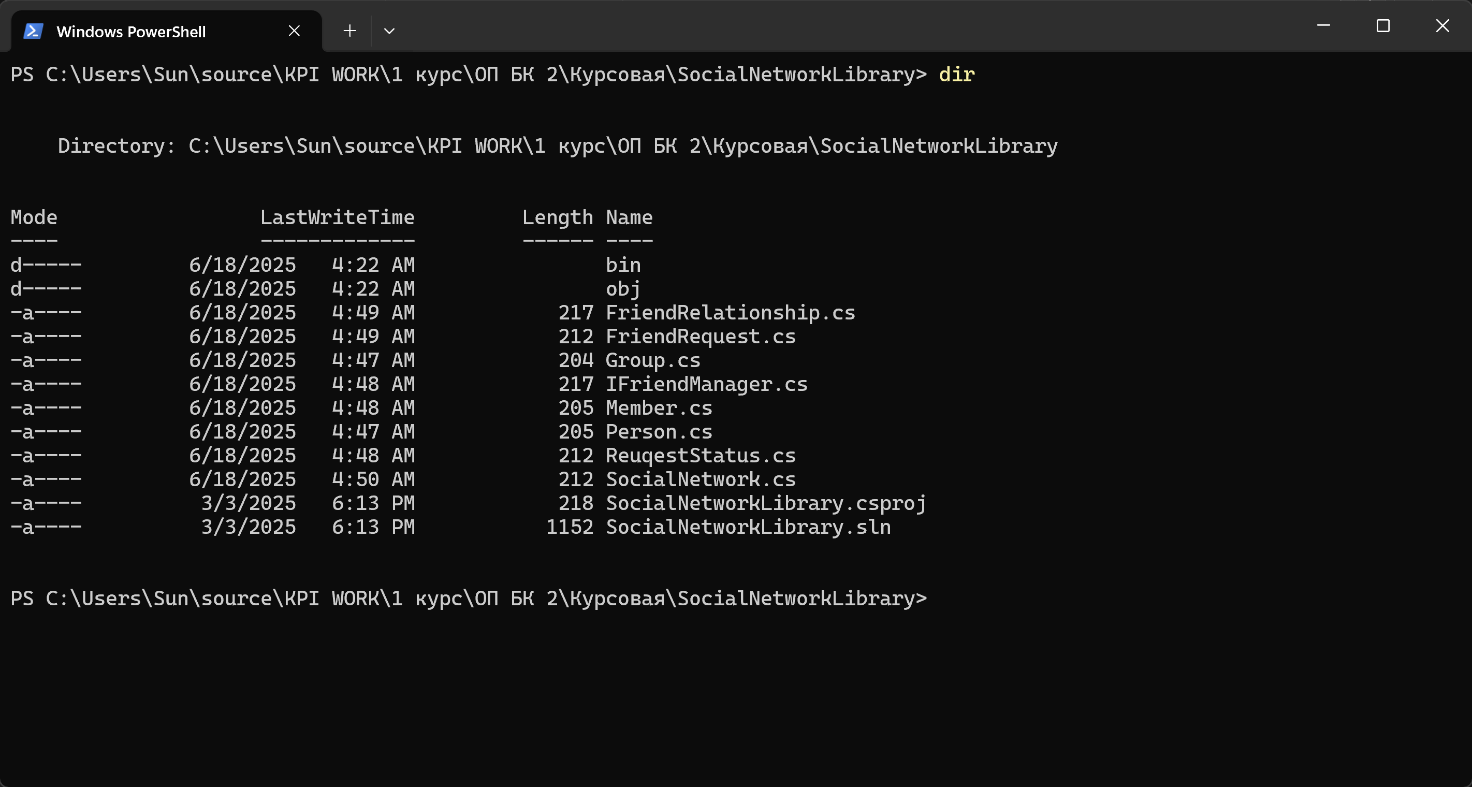


Рисунок 6.2.1 – виконання dir у powershell для оцінки розміру файлів бібліотеки

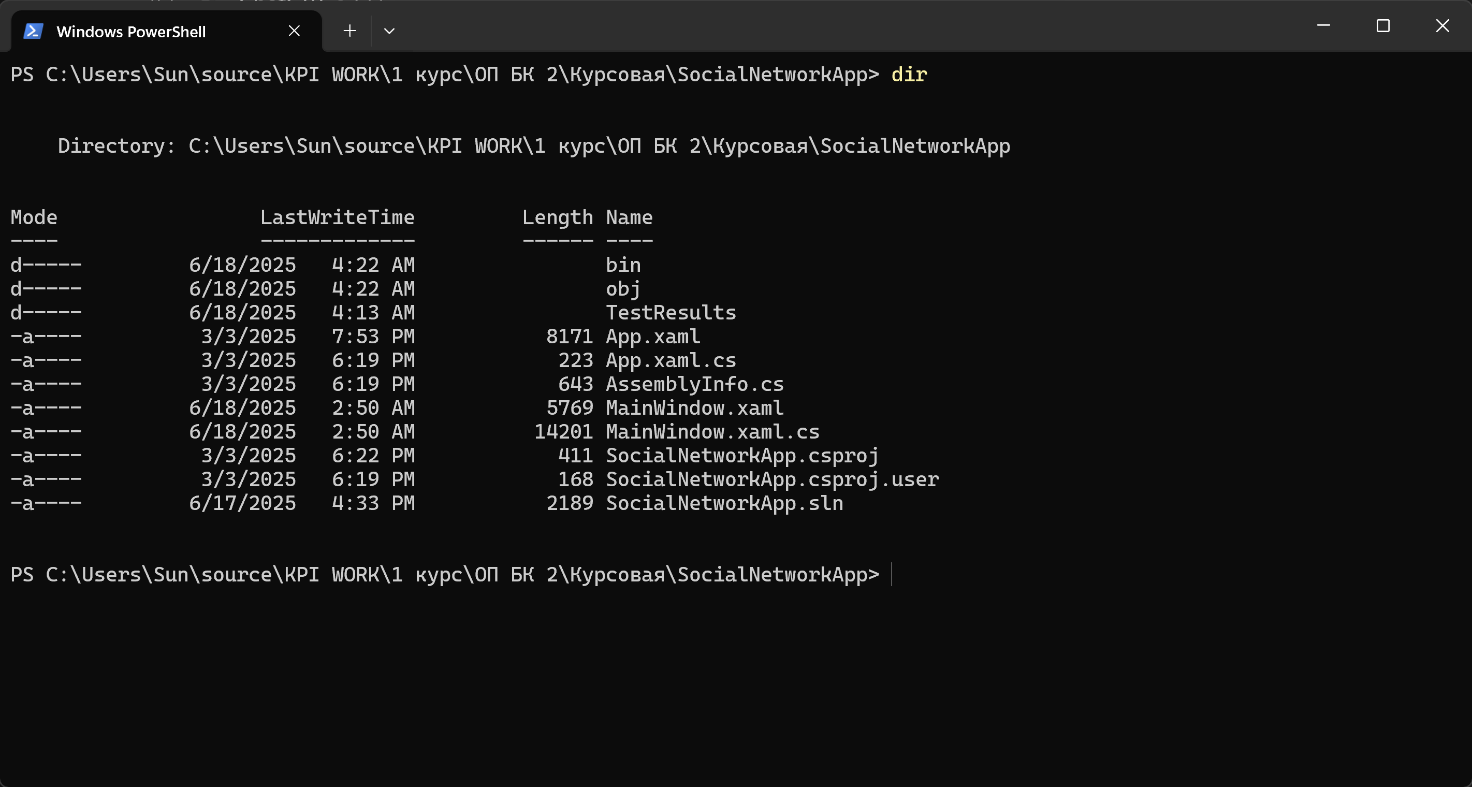


Рисунок 6.2.2 – виконання dir у powershell для оцінки розміру файлів програми

## 6.3 Опис інтерфейсу користувача

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 6.3 – підписаний номерами інтерфейс програми

| **№** | **Підпис** | **Функція елемента** |
| --- | --- | --- |
| 1 | **Add new user** | Поля *Username*, *Email* та кнопка **Add user**. Валідація формату email і унікальности. |
| 2 | **Create new group** | Текстове поле *Group name*, комбобокс *Owner*, кнопка **Create group**. Створює Group і одразу додає Owner. |
| 3 | **Members** | *ListBox* із усіма зареєстрованими користувачами. Один клік — вибір активного користувача. |
| 4 | **Groups** | *ListBox* глобальних груп. При виборі завантажується склад групи (№ 8). |
| 5 | **Friends area** | Відображення друзів активного користувача; кнопки **Show friends / Remove friend**. |
| 6 | **Friend requests** | Відправлення (комбобокси *From / To*, **Send request**) та обробка вхідних запитів (**Accept / Decline**). |
| 7 | **Group management** | Комбобокс «Add member», кнопки **Add / Remove / Delete group / Leave group**. Логіка увімкнення залежить від ролі (Owner чи учасник). |
| 8 | **Group members list** | Поточний склад обраної групи. Дозволяє виділити учасника для видалення. |
| 9 | **Status bar** | Зелений текст — успішна операція; червоний — помилка валідації або бізнес-логіки. |

Таблиця 6.3.1 – опис елементів інтерфейсу програми за номерами

**Коротка інструкція користувача (за скріном)**

1. **Додати користувача**  
   • Заповніть поля (1) → **Add user** → у статусі з’явиться *User 'Alice' added.*
2. **Надіслати запит дружби**  
   • Виберіть *Sender*/*Receiver* у блоці (6) → **Send request**.
3. **Прийняти запит**  
   • Переключіться на отримувача через список (3).  
   • У блоці (6) виділіть запит → **Accept**.
4. **Створити групу**  
   • Заповніть форму (2) → **Create group**.  
   • Група з’явиться в списку (4), а Owner — у складі (8).
5. **Додати учасника до групи**  
   • Виберіть групу (4).  
   • У комбобоксі «Add member» (7) виберіть користувача → **Add**.
6. **Вийти з групи**  
   • Активуйте себе у (3), позначте групу (4) → **Leave group** (7).  
   • Якщо ви Owner, кнопка буде неактивна — доступне лише **Delete group**.

**Важливі особливості**

* **Динамічне ввімкнення кнопок.**  
  При виборі друзів / учасників керуючі кнопки (5 і 7) стають активними лише коли операція допустима.
* **Одне вікно — вся функціональність.**  
  Навігація не потребує вкладених діалогів; усі списки оновлюються синхронно.
* **Колірна індикація стану.**  
  Помилки одразу видно, що спрощує ручне тестування (Smoke Run).

# **ВИСНОВОК**

У ході виконання курсової роботи було послідовно реалізовано повний цикл об’єктно-орієнтованого проєктування і розроблення прикладної системи «Соціальна мережа з категоріями груп». На етапі аналізу предметної області визначено сутності «учасник», «дружній зв’язок», «запит дружби», «група» та сформульовано функціональні вимоги до керування ними. За допомогою UML-діаграм продемонстровано всі базові види відношень між класами (успадкування, асоціація, агрегація, композиція, залежність), що дало змогу забезпечити логічну цілісність і розширюваність архітектури.

Практичним результатом став двошаровий програмний продукт: ядро у вигляді бібліотеки SocialNetworkLibrary (≈ 16 KB вихідного коду) та WPF-клієнт SocialNetworkApp (≈ 148 KB). Клас-бібліотека інкапсулює бізнес-логіку — створення і видалення учасників, обробку запитів дружби, формування та керування групами,-— а графічний інтерфейс надає користувачеві інтуїтивні засоби керування без дублювання коду. Розмежування шарів гарантує простоту супроводу і дає змогу безболісно додати, наприклад, веб-interface чи мобільний клієнт MAUI.

Якість реалізації підтверджена повним набором модульних і інтеграційних тестів MSTest: покрито типові та граничні сценарії роботи класів, усі 18 тест-кейсів пройдені успішно. Ручні «smoke»-перевірки UI показали коректну реакцію на типові дії користувача, наочне відображення помилок і відсутність збоїв при некоректному вводі. Така комбінація автоматизованого й ручного тестування забезпечує високу надійність системи.

У підсумку поставлену мету — створення придатної до промислового розширення соціальної мережі з підтримкою груп і керуванням дружніми відносинами — досягнуто повністю. Реалізовано всі вимоги технічного завдання: не менше шести типів даних, повний набір принципів ООП, усі зв’язки UML, опрацьовані виключні ситуації та єдина конвенція код-стайлу з XML-коментарями. Отриманий результат демонструє практичну придатність підходів об’єктно-орієнтованого моделювання й підтверджує ефективність комплексного тестування для підвищення якості прикладних рішень.

# **ПЕРЕЛІК ПОСИЛАНЬ**

1. Ґамма Е., Гелм Р., Джонсон Р., Влісідес Дж. **Шаблони проектування.** — Київ : Діалектика, 2021. — 395 с.
2. Буч Ґ. **Об’єктно-орієнтований аналіз і проектування з прикладами застосування UML 2.** — Львів : Нові знання, 2020. — 560 с.
3. OMG. **Unified Modeling Language Specification (v.2.5.1).** — Object Management Group, 2017.
4. Microsoft Corporation. **C# 10.0 Language Specification.** URL: <https://learn.microsoft.com/dotnet/csharp/> (дата звернення — 18.06.2025).
5. Microsoft Corporation. **.NET 6 Desktop Guide → WPF.** URL: <https://learn.microsoft.com/dotnet/desktop/wpf/> (дата звернення — 18.06.2025).
6. Microsoft Corporation. **Unit-testing with MSTest.** URL: <https://learn.microsoft.com/dotnet/core/testing/unit-testing-with-mstest> (дата звернення — 18.06.2025).
7. Папайоанну Е. **Програмування на C# для початківців.** — Харків : Ранок, 2022. — 312 с.
8. Richter Д. **CLR via C#.** 5-те вид. — Redmond : Microsoft Press, 2021. — 896 p.
9. Харт Дж. **WPF 3D, анімація та медіа.** — Київ : БХВ-ПРО, 2020. — 448 с.
10. ISO/IEC 25010:2011. **Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE).**

Top of Form

# **ДОДАТКИ**

**Додаток А**

**Технічне завдання**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Затвердив

Керівник Вітковська І. І

«\_\_\_» червень 2025 р.

Виконавець:

Студент(-ка) Ренькас Михайло Віталійович

«\_\_\_» червень 2025 р.

**ТЕХНІЧНЕ ЗАВДАННЯ**

на виконання курсової роботи

на тему: “Соціальна Мережа. Комунікації між членами мережі”

з дисципліни:

«Основи програмування»

Київ 2025

**0 Мета роботи**

Розробити настільне програмне забезпечення «Соціальна мережа» (C# / WPF, .NET 6), яке дозволяє:

* реєструвати учасників (клас **Member**),
* керувати дружніми відносинами через запити (**FriendRequest**) і зв’язки (**FriendRelationship**),
* створювати спільноти (**Group**) і керувати їхнім складом,
* відображати усі операції у зручному графічному інтерфейсі.

**1 Календарні межі**

* **Дата початку:** 20 квітня 2025 р.
* **Дата завершення:** 18 червня 2025 р.

**2 Вимоги до програмного забезпечення**

**Функціональні**

* Для учасника: реєстрація, перегляд/видалення друзів, надсилання-отримання-обробка запитів, приєднання та вихід із груп (крім власної).
* Для власника групи: створення та видалення групи, додавання та виключення учасників.
* Для інтерфейсу: окремі панелі керування користувачами, дружбою й групами; рядок стану з повідомленнями.

**Нефункціональні**

* Інтерфейс WPF, оптимізований для екранів від 1366×768.
* Відсутність неперехоплених виключень при некоректному вводі.
* Дотримання принципів SOLID; коментарі лише у форматі XML-док-коментарів.
* Документація й код мають відповідати ГОСТ 19.106-78, ГОСТ 34.602-89 та ДСТУ 3008-2015.

**3 Етапи і контрольні дати**

1. Об’єктно-орієнтований аналіз предметної області – до 21.04.2025.
2. Проектування архітектури (UML-діаграми) – до 25.04.2025.
3. Реалізація бібліотеки класів **SocialNetworkLibrary** – до 15.05.2025.
4. Створення WPF-клієнта **SocialNetworkApp** – до 20.05.2025.
5. Підготовка й запуск юніт-тестів MSTest – до 22.05.2025.
6. Написання пояснювальної записки – до 18.06.2025.
7. Захист курсової роботи – до 19.06.2025.

**4 Порядок контролю та приймання**

* Щотижневе пред’явлення прогресу керівникові.
* Прострочення будь-якого етапу знижує максимальну оцінку на 1 бал за кожний день затримки.
* До захисту студент подає друковану та електронну версії пояснювальної записки, повний вихідний код бібліотеки, клієнтського застосунку і проєкту MSTest, а також демонстрацію успішного запуску всіх модульних тестів.

Bottom of Form

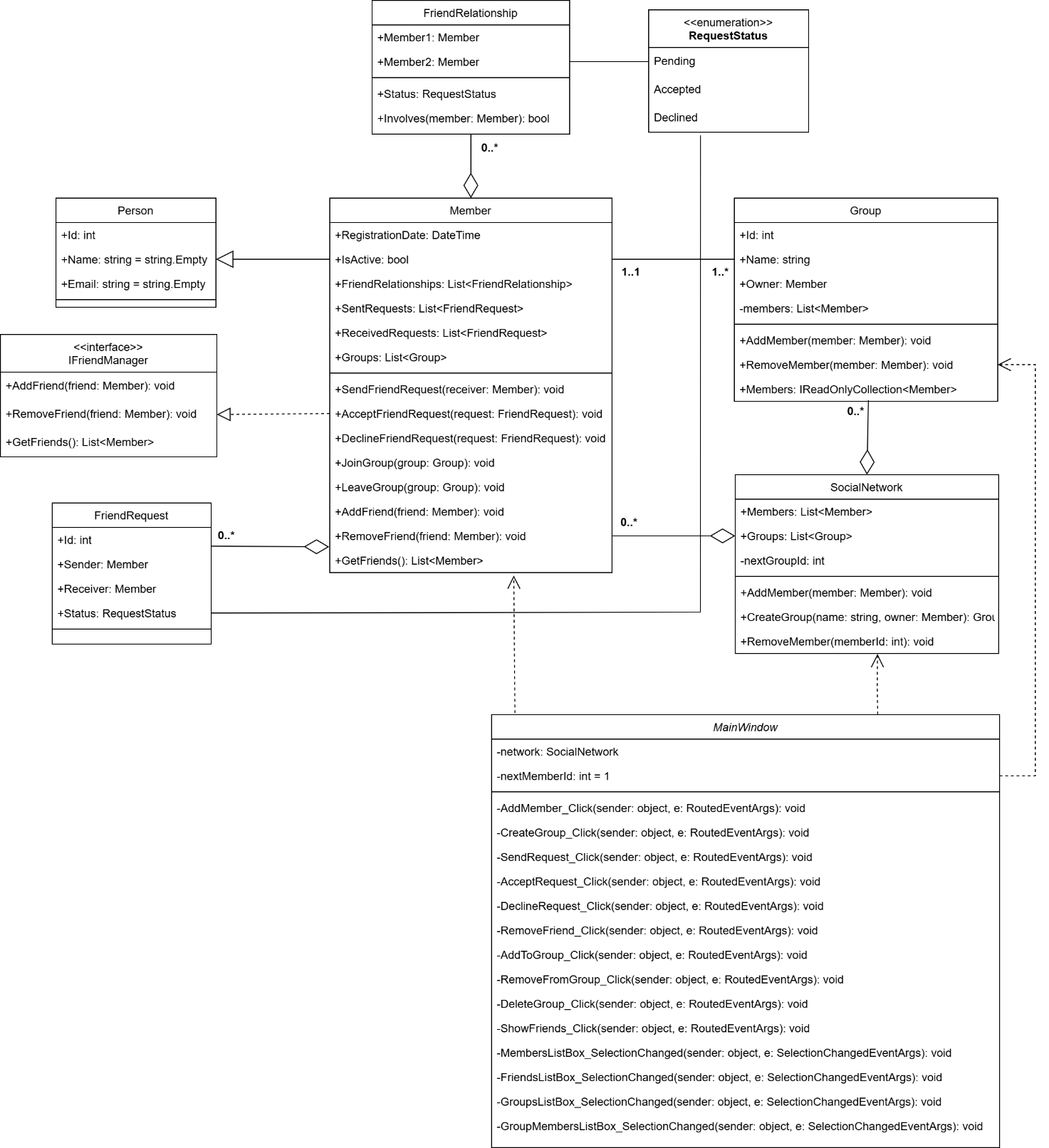
**ДОДАТОК Б**

**Діаграма варіантів використання**

****

**ДОДАТОК В**

**Діаграма класів використана для розробки програми і є описуючою програму**

****

**Додаток Г**

**Тексти програмного коду**

*Тексти програмного коду програмного забезпечення*

*“Соціальна мережа. Комунікації між членами мережі”*

***SocialNetwork а***

*(Найменування програми (документа))*

***CD-RW*** к

*(Вид носія даних)*

*164 Кб* **к**

*(Обсяг програми (документа), арк., Кб)*

*студент 1 курсу групи ІП-з41*

*Ренькаса Михайла Віталійовича*

//App.xaml

<Application x:Class="SocialNetworkApp.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

StartupUri="MainWindow.xaml">

<Application.Resources>

<Style TargetType="Button">

<Setter Property="Background" Value="#FF6200EE"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="FontWeight" Value="Bold"/>

<Setter Property="Margin" Value="5"/>

<Setter Property="Padding" Value="10,5"/>

<Setter Property="BorderThickness" Value="0"/>

<Setter Property="BorderBrush" Value="Transparent"/>

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="Cursor" Value="Hand"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border Background="{TemplateBinding Background}" CornerRadius="4">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style TargetType="TextBox">

<Setter Property="Margin" Value="5"/>

<Setter Property="Padding" Value="5"/>

<Setter Property="BorderBrush" Value="#FF6200EE"/>

<Setter Property="BorderThickness" Value="2"/>

<Setter Property="HorizontalAlignment" Value="Stretch"/>

</Style>

<Style TargetType="ListBox">

<Setter Property="Margin" Value="5"/>

<Setter Property="BorderBrush" Value="#FF6200EE"/>

<Setter Property="BorderThickness" Value="2"/>

</Style>

<Style TargetType="TextBlock">

<Setter Property="Margin" Value="5,5,5,0"/>

<Setter Property="FontSize" Value="14"/>

</Style>

<Style TargetType="ComboBox">

<Setter Property="Background" Value="Transparent"/>

<Setter Property="Foreground" Value="Black"/>

<Setter Property="BorderBrush" Value="#FF6200EE"/>

<Setter Property="BorderThickness" Value="2"/>

<Setter Property="Padding" Value="5"/>

<Setter Property="Margin" Value="5"/>

<Setter Property="HorizontalContentAlignment" Value="Left"/>

<Setter Property="VerticalContentAlignment" Value="Center"/>

<Setter Property="DisplayMemberPath" Value="Name"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ComboBox">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="20"/>

</Grid.ColumnDefinitions>

<Border x:Name="ComboBorder"

Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}"

Grid.Column="0">

<ContentPresenter

x:Name="ContentPresenter"

Margin="4,2,4,2"

IsHitTestVisible="False"

Content="{TemplateBinding SelectionBoxItem}"

ContentTemplate="{TemplateBinding SelectionBoxItemTemplate}"

ContentTemplateSelector="{TemplateBinding ItemTemplateSelector}"

ContentStringFormat="{TemplateBinding SelectionBoxItemStringFormat}"

HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"

RecognizesAccessKey="True"/>

</Border>

<ToggleButton Grid.Column="1"

Focusable="False"

IsChecked="{Binding IsDropDownOpen, Mode=TwoWay, RelativeSource={RelativeSource TemplatedParent}}"

ClickMode="Press"

Background="Transparent"

BorderThickness="0">

<Path x:Name="Arrow"

Data="M 0 0 L 4 4 L 8 0 Z"

HorizontalAlignment="Center"

VerticalAlignment="Center"

Fill="{TemplateBinding Foreground}"/>

</ToggleButton>

<Popup x:Name="Popup"

Placement="Bottom"

IsOpen="{TemplateBinding IsDropDownOpen}"

AllowsTransparency="True"

Focusable="False"

PopupAnimation="Slide">

<Border x:Name="PopupBorder"

Background="White"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="2">

<ScrollViewer>

<StackPanel IsItemsHost="True"/>

</ScrollViewer>

</Border>

</Popup>

</Grid>

<ControlTemplate.Triggers>

<Trigger Property="HasItems" Value="false">

<Setter TargetName="PopupBorder" Property="MinHeight" Value="95"/>

</Trigger>

<Trigger Property="IsEnabled" Value="False">

<Setter TargetName="ComboBorder" Property="Background" Value="LightGray"/>

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style TargetType="ComboBoxItem">

<Setter Property="Background" Value="Transparent"/>

<Setter Property="Padding" Value="4"/>

<Setter Property="HorizontalContentAlignment" Value="Left"/>

<Setter Property="VerticalContentAlignment" Value="Center"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="ComboBoxItem">

<Border x:Name="Bd" Background="{TemplateBinding Background}">

<ContentPresenter HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"

Margin="2"/>

</Border>

<ControlTemplate.Triggers>

<Trigger Property="IsHighlighted" Value="True">

<Setter TargetName="Bd" Property="Background" Value="#FF6200EE"/>

<Setter Property="Foreground" Value="White"/>

</Trigger>

<Trigger Property="IsSelected" Value="True">

<Setter TargetName="Bd" Property="Background" Value="#FF6200EE"/>

<Setter Property="Foreground" Value="White"/>

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Application.Resources>

</Application>

//App.xaml.cs

using System.Configuration;

using System.Data;

using System.Windows;

namespace SocialNetworkApp;

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

}

//MainWindow.xaml

<Window x:Class="SocialNetworkApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Social Network" Height="859" Width="1039">

<Window.Background>

<LinearGradientBrush StartPoint="0,0" EndPoint="0,1">

<GradientStop Color="#FFECECEC" Offset="0"/>

<GradientStop Color="#FFFFFFFF" Offset="1"/>

</LinearGradientBrush>

</Window.Background>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Column="0" Margin="20">

<TextBlock Text="Add new user:" FontWeight="Bold"/>

<TextBlock Text="Username:"/>

<TextBox x:Name="MemberNameTextBox"/>

<TextBlock Text="Email:"/>

<TextBox x:Name="MemberEmailTextBox"/>

<Button Content="Add user" Click="AddMember\_Click"/>

<Border BorderThickness="2" BorderBrush="#FF6200EE" Padding="6" Margin="0,15,0,0" Width="313">

<StackPanel>

<TextBlock Text="Create new group:" FontWeight="Bold"/>

<TextBlock Text="Group name:"/>

<TextBox x:Name="GroupNameTextBox"/>

<TextBlock Text="Owner:"/>

<ComboBox x:Name="GroupOwnerComboBox" DisplayMemberPath="Name" Width="140"/>

<Button Content="Create group" Click="CreateGroup\_Click"/>

</StackPanel>

</Border>

<TextBlock Text="Network members:" FontWeight="Bold" Margin="0,14,0,0"/>

<ListBox x:Name="MembersListBox" Height="140" DisplayMemberPath="Name"

SelectionChanged="MembersListBox\_SelectionChanged"/>

<TextBlock Text="Groups:" FontWeight="Bold" Margin="8,10,0,0"/>

<ListBox x:Name="GroupsListBox" Height="120" DisplayMemberPath="Name"

SelectionChanged="GroupsListBox\_SelectionChanged"/>

</StackPanel>

<StackPanel Grid.Column="1" Margin="20">

<TextBlock Text="Selected user's friends:" FontWeight="Bold"/>

<ListBox x:Name="FriendsListBox" Height="80" DisplayMemberPath="Name"

SelectionChanged="FriendsListBox\_SelectionChanged"/>

<StackPanel Orientation="Horizontal" Margin="0,4,0,0">

<Button Content="Show friends" Click="ShowFriends\_Click"/>

<Button x:Name="RemoveFriendButton" Content="Remove friend"

Click="RemoveFriend\_Click" Margin="5,0,0,0" IsEnabled="False"/>

</StackPanel>

<TextBlock Text="Send friend request:" FontWeight="Bold" Margin="0,10,0,0"/>

<StackPanel Orientation="Horizontal">

<TextBlock Text="From:" VerticalAlignment="Center"/>

<ComboBox x:Name="SenderComboBox" Width="120" DisplayMemberPath="Name" Margin="5,0"/>

<TextBlock Text="To:" VerticalAlignment="Center" Margin="10,0,0,0"/>

<ComboBox x:Name="ReceiverComboBox" Width="120" DisplayMemberPath="Name" Margin="5,0"/>

</StackPanel>

<Button Content="Send request" Click="SendRequest\_Click"/>

<TextBlock Text="Incoming friend requests for selected user:" FontWeight="Bold" Margin="0,10,0,0"/>

<ListBox x:Name="IncomingRequestsListBox" Height="80">

<ListBox.ItemTemplate>

<DataTemplate>

<TextBlock>

<TextBlock.Text>

<MultiBinding StringFormat="Request #{0}: from {1} – {2}">

<Binding Path="Id"/>

<Binding Path="Sender.Name"/>

<Binding Path="Status"/>

</MultiBinding>

</TextBlock.Text>

</TextBlock>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<StackPanel Orientation="Horizontal">

<Button Content="Accept" Click="AcceptRequest\_Click"/>

<Button Content="Decline" Click="DeclineRequest\_Click" Margin="5,0,0,0"/>

</StackPanel>

<Separator Margin="0,10,0,10"/>

<TextBlock Text="Selected group's members:" FontWeight="Bold"/>

<ListBox x:Name="GroupMembersListBox" Height="80" DisplayMemberPath="Name"

SelectionChanged="GroupMembersListBox\_SelectionChanged"/>

<StackPanel Orientation="Horizontal" Margin="0,4,0,0">

<TextBlock Text="Add member:" VerticalAlignment="Center"/>

<ComboBox x:Name="AddToGroupComboBox" Width="120" DisplayMemberPath="Name" Margin="5,0"/>

<Button Content="Add" Click="AddToGroup\_Click"/>

<Button x:Name="RemoveFromGroupButton" Content="Remove"

Click="RemoveFromGroup\_Click" Margin="5,0,0,0" IsEnabled="False"/>

<Button x:Name="DeleteGroupButton" Content="Delete group"

Click="DeleteGroup\_Click" Margin="20,0,0,0" IsEnabled="False"/>

<Button Content="Leave group" x:Name="LeaveGroupButton" Margin="6,0,0,0" Click="LeaveGroup\_Click" IsEnabled="False"/>

</StackPanel>

<TextBlock x:Name="StatusTextBlock" FontSize="16" Margin="5,22,0,0" TextWrapping="Wrap"/>

</StackPanel>

</Grid>

</Window>

//MainWindow.xaml.cs

using SocialNetworkLibrary;

using System;

using System.Linq;

using System.Text.RegularExpressions;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using Group = SocialNetworkLibrary.Group;

namespace SocialNetworkApp

{

public partial class MainWindow : Window

{

private readonly SocialNetwork \_network;

private int \_nextMemberId = 1;

public MainWindow()

{

InitializeComponent();

\_network = new SocialNetwork();

}

/// <summary>Adds a new member.</summary>

private void AddMember\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

string name = MemberNameTextBox.Text.Trim();

string email = MemberEmailTextBox.Text.Trim();

try

{

if (!Regex.IsMatch(name, @"^[a-zA-Z0-9\_]+$"))

throw new ArgumentException("Username may contain only letters, digits and '\_'.");

if (!Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$"))

throw new ArgumentException("Invalid e-mail format.");

if (\_network.Members.Any(m => m.Name.Equals(name, StringComparison.OrdinalIgnoreCase)))

throw new ArgumentException("Username already exists.");

if (\_network.Members.Any(m => m.Email.Equals(email, StringComparison.OrdinalIgnoreCase)))

throw new ArgumentException("E-mail already exists.");

var member = new Member(\_nextMemberId++, name, email);

\_network.AddMember(member);

UpdateMembersUI();

ShowOk($"User '{name}' added.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Creates a new group.</summary>

private void CreateGroup\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

string name = GroupNameTextBox.Text.Trim();

if (GroupOwnerComboBox.SelectedItem is not Member owner)

{

ShowError("Select a group owner.");

return;

}

try

{

if (string.IsNullOrWhiteSpace(name))

throw new ArgumentException("Group name cannot be empty.");

var group = \_network.CreateGroup(name, owner);

UpdateGroupsUI();

GroupsListBox.SelectedItem = group;

ShowOk($"Group '{name}' created.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Sends a friend request.</summary>

private void SendRequest\_Click(object \_, RoutedEventArgs e)

{

ClearStatus();

if (SenderComboBox.SelectedItem is not Member sender ||

ReceiverComboBox.SelectedItem is not Member receiver)

{

ShowError("Select sender and receiver.");

return;

}

if (sender.Id == receiver.Id)

{

ShowError("Cannot send request to yourself.");

return;

}

try

{

sender.SendFriendRequest(receiver);

RefreshIncomingRequests(receiver);

ShowOk($"Request sent from '{sender.Name}' to '{receiver.Name}'.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Accepts the selected friend request.</summary>

private void AcceptRequest\_Click(object sender, RoutedEventArgs e) =>

HandleRequest(RequestStatus.Accepted);

/// <summary>Declines the selected friend request.</summary>

private void DeclineRequest\_Click(object sender, RoutedEventArgs e) =>

HandleRequest(RequestStatus.Declined);

/// <summary>Removes a friend.</summary>

private void RemoveFriend\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

if (MembersListBox.SelectedItem is not Member owner ||

FriendsListBox.SelectedItem is not Member friend)

{

ShowError("Select a user and a friend.");

return;

}

try

{

owner.RemoveFriend(friend);

FriendsListBox.ItemsSource = owner.GetFriends();

RemoveFriendButton.IsEnabled = false;

ShowOk($"'{friend.Name}' removed from friends.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Adds the selected member to the selected group.</summary>

private void AddToGroup\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

if (GroupsListBox.SelectedItem is not Group group ||

AddToGroupComboBox.SelectedItem is not Member member)

{

ShowError("Select group and member.");

return;

}

try

{

group.AddMember(member);

RefreshGroupMembers(group);

ShowOk($"'{member.Name}' joined '{group.Name}'.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Removes the highlighted member from the selected group.</summary>

private void RemoveFromGroup\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

if (GroupsListBox.SelectedItem is not Group group ||

GroupMembersListBox.SelectedItem is not Member member)

{

ShowError("Select group and its member.");

return;

}

try

{

group.RemoveMember(member);

RefreshGroupMembers(group);

RemoveFromGroupButton.IsEnabled = false;

ShowOk($"'{member.Name}' removed from '{group.Name}'.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Deletes the selected group (owner only).</summary>

private void DeleteGroup\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

if (GroupsListBox.SelectedItem is not Group group)

{

ShowError("Select a group.");

return;

}

if (MembersListBox.SelectedItem is not Member current ||

current.Id != group.Owner.Id)

{

ShowError("Only the owner can delete the group.");

return;

}

\_network.Groups.Remove(group);

foreach (var m in group.Members.ToList())

group.RemoveMember(m);

UpdateGroupsUI();

GroupMembersListBox.ItemsSource = null;

DeleteGroupButton.IsEnabled = false;

ShowOk($"Group '{group.Name}' deleted.");

}

/// <summary>Shows friends of the selected member.</summary>

private void ShowFriends\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

if (MembersListBox.SelectedItem is not Member member)

{

ShowError("Select a user.");

return;

}

FriendsListBox.ItemsSource = member.GetFriends();

RemoveFriendButton.IsEnabled = false;

ShowOk($"Friends of '{member.Name}'.");

}

/// <summary>Handles member selection.</summary>

private void MembersListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ClearStatus();

Member member = null!;

if (MembersListBox.SelectedItem is Member)

{

member = (Member)MembersListBox.SelectedItem;

RefreshIncomingRequests(member);

}

if (GroupsListBox.SelectedItem is Group g)

DeleteGroupButton.IsEnabled = member?.Id == g.Owner.Id;

}

/// <summary>Enables friend remove button.</summary>

private void FriendsListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e) =>

RemoveFriendButton.IsEnabled = FriendsListBox.SelectedItem is Member;

/// <summary>Handles group selection.</summary>

private void GroupsListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ClearStatus();

if (GroupsListBox.SelectedItem is Group group)

{

RefreshGroupMembers(group);

DeleteGroupButton.IsEnabled = MembersListBox.SelectedItem is Member member &&

member.Id == group.Owner.Id;

}

else

{

GroupMembersListBox.ItemsSource = null;

DeleteGroupButton.IsEnabled = false;

}

}

/// <summary>Enables remove-from-group button.</summary>

private void GroupMembersListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

RemoveFromGroupButton.IsEnabled = GroupMembersListBox.SelectedItem is Member;

LeaveGroupButton.IsEnabled =

GroupsListBox.SelectedItem is Group g &&

MembersListBox.SelectedItem is Member current &&

g.Members.Any(m => m.Id == current.Id) &&

g.Owner.Id != current.Id;

}

/// <summary>Updates all controls that show members.</summary>

private void UpdateMembersUI()

{

MembersListBox.ItemsSource = null;

MembersListBox.ItemsSource = \_network.Members;

SenderComboBox.ItemsSource = null;

SenderComboBox.ItemsSource = \_network.Members;

ReceiverComboBox.ItemsSource = null;

ReceiverComboBox.ItemsSource = \_network.Members;

GroupOwnerComboBox.ItemsSource = null;

GroupOwnerComboBox.ItemsSource = \_network.Members;

AddToGroupComboBox.ItemsSource = null;

AddToGroupComboBox.ItemsSource = \_network.Members;

}

/// <summary>Updates the groups list.</summary>

private void UpdateGroupsUI()

{

GroupsListBox.ItemsSource = null;

GroupsListBox.ItemsSource = \_network.Groups;

}

/// <summary>Refreshes pending requests for a member.</summary>

private void RefreshIncomingRequests(Member member)

{

IncomingRequestsListBox.ItemsSource = null;

IncomingRequestsListBox.ItemsSource =

member.ReceivedRequests.Where(r => r.Status == RequestStatus.Pending).ToList();

}

/// <summary>Refreshes members shown for a group.</summary>

private void RefreshGroupMembers(Group group)

{

GroupMembersListBox.ItemsSource = null;

GroupMembersListBox.ItemsSource = group.Members;

}

/// <summary>Processes accept or decline of a request.</summary>

private void HandleRequest(RequestStatus result)

{

ClearStatus();

if (MembersListBox.SelectedItem is not Member receiver ||

IncomingRequestsListBox.SelectedItem is not FriendRequest request)

{

ShowError("Select user and request.");

return;

}

try

{

if (result == RequestStatus.Accepted)

receiver.AcceptFriendRequest(request);

else

receiver.DeclineFriendRequest(request);

RefreshIncomingRequests(receiver);

ShowOk($"Request #{request.Id} {result.ToString().ToLower()}.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Leaves the selected group (non-owner).</summary>

private void LeaveGroup\_Click(object sender, RoutedEventArgs e)

{

ClearStatus();

if (GroupsListBox.SelectedItem is not Group group ||

MembersListBox.SelectedItem is not Member member)

{

ShowError("Select active user and a group.");

return;

}

if (group.Owner.Id == member.Id)

{

ShowError("Owner cannot leave the group. Use Delete Group.");

return;

}

try

{

member.LeaveGroup(group);

RefreshGroupMembers(group);

LeaveGroupButton.IsEnabled = false;

ShowOk($"'{member.Name}' left '{group.Name}'.");

}

catch (Exception ex)

{

ShowError(ex.Message);

}

}

/// <summary>Clears the status line.</summary>

private void ClearStatus() => StatusTextBlock.Text = string.Empty;

/// <summary>Displays an error in red.</summary>

private void ShowError(string message)

{

StatusTextBlock.Foreground = Brushes.Red;

StatusTextBlock.Text = message;

}

/// <summary>Displays a normal message in green.</summary>

private void ShowOk(string message)

{

StatusTextBlock.Foreground = Brushes.Green;

StatusTextBlock.Text = message;

}

}

}

//AssemblyInfo.cs

using System.Windows;

[assembly:ThemeInfo(

ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located

//(used if a resource is not found in the page,

// or application resource dictionaries)

ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located

//(used if a resource is not found in the page,

// app, or any theme specific resource dictionaries)

)]

//FriendRelationship.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Represents a friendship relation between two members.

/// </summary>

public class FriendRelationship

{

/// <summary>

/// Initializes a new instance of the <see cref="FriendRelationship"/> class.

/// </summary>

/// <param name="member1">First member.</param>

/// <param name="member2">Second member.</param>

public FriendRelationship(Member member1, Member member2)

{

Member1 = member1 ?? throw new ArgumentNullException(nameof(member1));

Member2 = member2 ?? throw new ArgumentNullException(nameof(member2));

}

/// <summary>Gets the first participant of the relation.</summary>

public Member Member1 { get; }

/// <summary>Gets the second participant of the relation.</summary>

public Member Member2 { get; }

/// <summary>

/// Gets the opposite participant in the relation.

/// </summary>

/// <param name="member">Known participant.</param>

/// <returns>The other participant.</returns>

internal Member GetOther(Member member) =>

member.Id == Member1.Id ? Member2 : Member1;

/// <summary>

/// Checks whether a member participates in this relation.

/// </summary>

/// <param name="member">Member to test.</param>

/// <returns><c>true</c> if the member participates; otherwise, <c>false</c>.</returns>

internal bool Involves(Member member) =>

member.Id == Member1.Id || member.Id == Member2.Id;

}

}

//FriendRequest.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Represents a friend request between two members.

/// </summary>

public class FriendRequest

{

private static int \_nextId = 1;

/// <summary>

/// Initializes a new instance of the <see cref="FriendRequest"/> class.

/// </summary>

/// <param name="sender">Sending member.</param>

/// <param name="receiver">Receiving member.</param>

public FriendRequest(Member sender, Member receiver)

{

Id = \_nextId++;

Sender = sender ?? throw new ArgumentNullException(nameof(sender));

Receiver = receiver ?? throw new ArgumentNullException(nameof(receiver));

Status = RequestStatus.Pending;

}

/// <summary>Gets the unique request identifier.</summary>

public int Id { get; }

/// <summary>Gets the request sender.</summary>

public Member Sender { get; }

/// <summary>Gets the request receiver.</summary>

public Member Receiver { get; }

/// <summary>Gets or sets the current request status.</summary>

public RequestStatus Status { get; set; }

}

}

//Group.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Represents a community inside the social network.

/// </summary>

public class Group

{

/// <summary>

/// Initializes a new instance of the <see cref="Group"/> class.

/// </summary>

/// <param name="id">Unique group identifier.</param>

/// <param name="name">Group name.</param>

/// <param name="owner">Member who owns the group.</param>

public Group(int id, string name, Member owner)

{

Id = id;

Name = name ?? throw new ArgumentNullException(nameof(name));

Owner = owner ?? throw new ArgumentNullException(nameof(owner));

\_members = new List<Member> { owner };

owner.Groups.Add(this);

}

/// <summary>Gets the group's unique identifier.</summary>

public int Id { get; }

/// <summary>Gets the group's name.</summary>

public string Name { get; }

/// <summary>Gets the member who owns the group.</summary>

public Member Owner { get; }

private readonly List<Member> \_members;

/// <summary>Gets the current group members.</summary>

public IReadOnlyCollection<Member> Members => \_members.AsReadOnly();

/// <summary>

/// Adds a member to the group.

/// </summary>

/// <param name="member">Member to add.</param>

/// <exception cref="ArgumentNullException">If <paramref name="member"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If the member is already in the group.</exception>

public void AddMember(Member member)

{

\_ = member ?? throw new ArgumentNullException(nameof(member));

if (\_members.Any(m => m.Id == member.Id))

throw new InvalidOperationException("Member already in the group.");

\_members.Add(member);

member.Groups.Add(this);

}

/// <summary>

/// Removes a member from the group.

/// </summary>

/// <param name="member">Member to remove.</param>

/// <exception cref="ArgumentNullException">If <paramref name="member"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If the member is not found in the group.</exception>

public void RemoveMember(Member member)

{

\_ = member ?? throw new ArgumentNullException(nameof(member));

if (!\_members.Remove(member))

throw new InvalidOperationException("Member not found in the group.");

member.Groups.Remove(this);

}

}

}

//IFriendManager.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Provides basic operations for managing a friend list.

/// </summary>

public interface IFriendManager

{

/// <summary>Adds a member to the friend list.</summary>

/// <param name="friend">Member to add.</param>

void AddFriend(Member friend);

/// <summary>Removes a member from the friend list.</summary>

/// <param name="friend">Member to remove.</param>

void RemoveFriend(Member friend);

/// <summary>Gets an up-to-date list of friends.</summary>

/// <returns>Collection of friends.</returns>

List<Member> GetFriends();

}

}

//Member.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Represents a registered user of the social network.

/// </summary>

public class Member : Person, IFriendManager

{

/// <summary>

/// Initializes a new instance of the <see cref="Member"/> class.

/// </summary>

/// <param name="id">Unique identifier.</param>

/// <param name="name">Display name.</param>

/// <param name="email">E-mail address.</param>

public Member(int id, string name, string email)

{

Id = id;

Name = name ?? throw new ArgumentNullException(nameof(name));

Email = email ?? throw new ArgumentNullException(nameof(email));

RegistrationDate = DateTime.UtcNow;

FriendRelationships = new List<FriendRelationship>();

SentRequests = new List<FriendRequest>();

ReceivedRequests = new List<FriendRequest>();

Groups = new List<Group>();

IsActive = true;

}

/// <summary>Gets the UTC date and time when the member registered.</summary>

public DateTime RegistrationDate { get; }

/// <summary>Gets a value indicating whether the member account is active.</summary>

public bool IsActive { get; private set; }

/// <summary>Gets established friendship relations.</summary>

public List<FriendRelationship> FriendRelationships { get; }

/// <summary>Gets outgoing friend requests.</summary>

public List<FriendRequest> SentRequests { get; }

/// <summary>Gets incoming friend requests.</summary>

public List<FriendRequest> ReceivedRequests { get; }

/// <summary>Gets groups in which the member participates.</summary>

public List<Group> Groups { get; }

#region IFriendManager implementation

/// <inheritdoc/>

public void AddFriend(Member friend)

{

\_ = friend ?? throw new ArgumentNullException(nameof(friend));

if (GetFriends().Any(f => f.Id == friend.Id))

throw new InvalidOperationException("A friendship with this user already exists.");

FriendRelationships.Add(new FriendRelationship(this, friend));

friend.FriendRelationships.Add(new FriendRelationship(friend, this));

}

/// <inheritdoc/>

public void RemoveFriend(Member friend)

{

\_ = friend ?? throw new ArgumentNullException(nameof(friend));

var relation = FriendRelationships.FirstOrDefault(r => r.Involves(friend));

if (relation is null)

throw new InvalidOperationException("The friendship does not exist.");

FriendRelationships.Remove(relation);

var reciprocal = friend.FriendRelationships.FirstOrDefault(r => r.Involves(this));

if (reciprocal is not null)

friend.FriendRelationships.Remove(reciprocal);

}

/// <inheritdoc/>

public List<Member> GetFriends() =>

FriendRelationships

.Select(r => r.GetOther(this))

.Distinct()

.ToList();

#endregion

#region Friend-request logic

/// <summary>

/// Sends a friend request to another member.

/// </summary>

/// <param name="receiver">Receiver of the request.</param>

/// <exception cref="ArgumentNullException">If <paramref name="receiver"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If users are already friends or a pending request exists.</exception>

public void SendFriendRequest(Member receiver)

{

\_ = receiver ?? throw new ArgumentNullException(nameof(receiver));

if (GetFriends().Any(f => f.Id == receiver.Id))

throw new InvalidOperationException("A friendship with this user already exists.");

if (SentRequests.Any(r => r.Receiver.Id == receiver.Id && r.Status == RequestStatus.Pending))

throw new InvalidOperationException("A friend request has already been sent.");

var request = new FriendRequest(this, receiver);

SentRequests.Add(request);

receiver.ReceivedRequests.Add(request);

}

/// <summary>

/// Accepts an incoming friend request.

/// </summary>

/// <param name="request">Request to accept.</param>

/// <exception cref="ArgumentNullException">If <paramref name="request"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If the request cannot be processed.</exception>

public void AcceptFriendRequest(FriendRequest request)

{

\_ = request ?? throw new ArgumentNullException(nameof(request));

if (!ReceivedRequests.Contains(request))

throw new InvalidOperationException("The request was not found among incoming requests.");

if (request.Status != RequestStatus.Pending)

throw new InvalidOperationException("The request has already been processed.");

request.Status = RequestStatus.Accepted;

AddFriend(request.Sender);

}

/// <summary>

/// Declines an incoming friend request.

/// </summary>

/// <param name="request">Request to decline.</param>

/// <exception cref="ArgumentNullException">If <paramref name="request"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If the request cannot be processed.</exception>

public void DeclineFriendRequest(FriendRequest request)

{

\_ = request ?? throw new ArgumentNullException(nameof(request));

if (!ReceivedRequests.Contains(request))

throw new InvalidOperationException("The request was not found among incoming requests.");

if (request.Status != RequestStatus.Pending)

throw new InvalidOperationException("The request has already been processed.");

request.Status = RequestStatus.Declined;

}

#endregion

#region Group helpers

/// <summary>Joins the specified group.</summary>

/// <param name="group">Group to join.</param>

public void JoinGroup(Group group) => group?.AddMember(this);

/// <summary>Leaves the specified group.</summary>

/// <param name="group">Group to leave.</param>

public void LeaveGroup(Group group)

{

\_ = group ?? throw new ArgumentNullException(nameof(group));

if (group.Owner.Id == Id)

throw new InvalidOperationException("Owner cannot leave the group.");

group.RemoveMember(this);

}

#endregion

}

}

//Person.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Base class that supplies identity, name and e-mail.

/// </summary>

public abstract class Person

{

/// <summary>Gets or sets the unique identifier.</summary>

public int Id { get; set; }

/// <summary>Gets or sets the person’s display name.</summary>

public string Name { get; set; } = string.Empty;

/// <summary>Gets or sets the person’s e-mail address.</summary>

public string Email { get; set; } = string.Empty;

}

}

//RequestStatus.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Defines possible states of a friend request.

/// </summary>

public enum RequestStatus

{

/// <summary>The request is waiting for the receiver’s action.</summary>

Pending,

/// <summary>The receiver accepted the request.</summary>

Accepted,

/// <summary>The receiver declined the request.</summary>

Declined

}

}

//SocialNetwork.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SocialNetworkLibrary

{

/// <summary>

/// Root entity that aggregates members and groups.

/// </summary>

public class SocialNetwork

{

private int \_nextGroupId = 1;

/// <summary>

/// Initializes a new instance of the <see cref="SocialNetwork"/> class.

/// </summary>

public SocialNetwork()

{

Members = new List<Member>();

Groups = new List<Group>();

}

/// <summary>Gets registered members.</summary>

public List<Member> Members { get; }

/// <summary>Gets existing groups.</summary>

public List<Group> Groups { get; }

/// <summary>

/// Registers a new member in the network.

/// </summary>

/// <param name="member">Member to add.</param>

/// <exception cref="ArgumentNullException">If <paramref name="member"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If the identifier is already used.</exception>

public void AddMember(Member member)

{

\_ = member ?? throw new ArgumentNullException(nameof(member));

if (Members.Any(m => m.Id == member.Id))

throw new InvalidOperationException("A member with this identifier already exists.");

Members.Add(member);

}

/// <summary>

/// Creates a new group in the network.

/// </summary>

/// <param name="name">Group name.</param>

/// <param name="owner">Group owner.</param>

/// <returns>Created group instance.</returns>

/// <exception cref="ArgumentNullException">If <paramref name="name"/> or <paramref name="owner"/> is <c>null</c>.</exception>

/// <exception cref="InvalidOperationException">If the name is already taken.</exception>

public Group CreateGroup(string name, Member owner)

{

\_ = name ?? throw new ArgumentNullException(nameof(name));

\_ = owner ?? throw new ArgumentNullException(nameof(owner));

if (Groups.Any(g => g.Name.Equals(name, StringComparison.OrdinalIgnoreCase)))

throw new InvalidOperationException("Group name already taken.");

var group = new Group(\_nextGroupId++, name, owner);

Groups.Add(group);

return group;

}

/// <summary>

/// Removes a member (and cleans up all related data).

/// </summary>

/// <param name="memberId">Identifier of the member to remove.</param>

/// <exception cref="InvalidOperationException">If the member is not found.</exception>

public void RemoveMember(int memberId)

{

var member = Members.FirstOrDefault(m => m.Id == memberId);

if (member is null)

throw new InvalidOperationException("Member not found.");

foreach (var g in member.Groups.ToList())

g.RemoveMember(member);

foreach (var f in member.GetFriends().ToList())

member.RemoveFriend(f);

Members.Remove(member);

}

}

}

//MainTests.cs

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Linq;

namespace SocialNetworkLibrary.Tests

{

#region Member tests

[TestClass]

public class MemberTests

{

/// <summary>Checks that two different members can be added successfully.</summary>

[TestMethod]

public void AddMember\_AllowsUniqueId()

{

var net = new SocialNetwork();

net.AddMember(new Member(1, "Alice", "alice@mail.com"));

net.AddMember(new Member(2, "Bob", "bob@mail.com"));

Assert.AreEqual(2, net.Members.Count);

}

/// <summary>Adding a member with a duplicate Id must throw an exception.</summary>

[TestMethod]

public void AddMember\_DuplicateId\_Throws()

{

var net = new SocialNetwork();

net.AddMember(new Member(1, "Alice", "alice@mail.com"));

Assert.ThrowsException<InvalidOperationException>(

() => net.AddMember(new Member(1, "Another", "x@y.z")));

}

/// <summary>Sends a friend request and leaves it in the Pending state.</summary>

[TestMethod]

public void SendFriendRequest\_CreatesPendingRequest()

{

var alice = new Member(1, "Alice", "a@a.com");

var bob = new Member(2, "Bob", "b@b.com");

alice.SendFriendRequest(bob);

Assert.AreEqual(1, alice.SentRequests.Count);

Assert.AreEqual(RequestStatus.Pending, alice.SentRequests.First().Status);

Assert.AreEqual(1, bob.ReceivedRequests.Count);

}

/// <summary>Sends the same request twice and expects an exception.</summary>

[TestMethod]

public void SendFriendRequest\_DuplicatePending\_Throws()

{

var alice = new Member(1, "Alice", "a@a.com");

var bob = new Member(2, "Bob", "b@b.com");

alice.SendFriendRequest(bob);

Assert.ThrowsException<InvalidOperationException>(() =>

alice.SendFriendRequest(bob));

}

/// <summary>Accepting a request creates a friendship on both sides.</summary>

[TestMethod]

public void AcceptFriendRequest\_AddsFriendship()

{

var alice = new Member(1, "Alice", "a@a.com");

var bob = new Member(2, "Bob", "b@b.com");

alice.SendFriendRequest(bob);

var request = bob.ReceivedRequests.Single();

bob.AcceptFriendRequest(request);

Assert.AreEqual(RequestStatus.Accepted, request.Status);

Assert.AreEqual(1, alice.GetFriends().Count);

Assert.AreEqual("Bob", alice.GetFriends().Single().Name);

Assert.AreEqual(1, bob.GetFriends().Count);

}

/// <summary>Declining a request leaves both users without friendship.</summary>

[TestMethod]

public void DeclineFriendRequest\_LeavesNoFriendship()

{

var alice = new Member(1, "Alice", "a@a.com");

var bob = new Member(2, "Bob", "b@b.com");

alice.SendFriendRequest(bob);

var request = bob.ReceivedRequests.Single();

bob.DeclineFriendRequest(request);

Assert.AreEqual(RequestStatus.Declined, request.Status);

Assert.AreEqual(0, alice.GetFriends().Count);

Assert.AreEqual(0, bob.GetFriends().Count);

}

/// <summary>Removing a friend wipes the relation for both users.</summary>

[TestMethod]

public void RemoveFriend\_RemovesFromBothSides()

{

var alice = new Member(1, "Alice", "a@a.com");

var bob = new Member(2, "Bob", "b@b.com");

alice.AddFriend(bob); // one call is enough

Assert.AreEqual(1, alice.GetFriends().Count);

alice.RemoveFriend(bob);

Assert.AreEqual(0, alice.GetFriends().Count);

Assert.AreEqual(0, bob.GetFriends().Count);

}

}

#endregion

#region Group & network tests

[TestClass]

public class GroupAndNetworkTests

{

/// <summary>Creating a group adds it to the network and joins the owner.</summary>

[TestMethod]

public void CreateGroup\_AddsGroupAndOwner()

{

var net = new SocialNetwork();

var alice = new Member(1, "Alice", "alice@mail.com");

net.AddMember(alice);

var group = net.CreateGroup("Chess Club", alice);

Assert.AreEqual(1, net.Groups.Count);

Assert.AreSame(alice, group.Owner);

Assert.IsTrue(group.Members.Contains(alice));

Assert.IsTrue(alice.Groups.Contains(group));

}

/// <summary>Attempting to create a group with a taken name must fail.</summary>

[TestMethod]

public void CreateGroup\_DuplicateName\_Throws()

{

var net = new SocialNetwork();

var alice = new Member(1, "Alice", "a@a.com");

net.AddMember(alice);

net.CreateGroup("Chess", alice);

Assert.ThrowsException<InvalidOperationException>(() =>

net.CreateGroup("Chess", alice));

}

/// <summary>Adding a member via Group.AddMember updates both collections.</summary>

[TestMethod]

public void Group\_AddMember\_AddsBothSides()

{

var net = new SocialNetwork();

var owner = new Member(1, "Owner", "o@o.com");

var bob = new Member(2, "Bob", "b@b.com");

net.AddMember(owner);

net.AddMember(bob);

var club = net.CreateGroup("Club", owner);

club.AddMember(bob);

Assert.IsTrue(club.Members.Contains(bob));

Assert.IsTrue(bob.Groups.Contains(club));

}

/// <summary>Adding the same member twice throws an exception.</summary>

[TestMethod]

public void Group\_AddMember\_Duplicate\_Throws()

{

var owner = new Member(1, "Owner", "o@o.com");

var bob = new Member(2, "Bob", "b@b.com");

var club = new Group(1, "Club", owner);

club.AddMember(bob);

Assert.ThrowsException<InvalidOperationException>(() =>

club.AddMember(bob));

}

/// <summary>Removing a member clears references on both sides.</summary>

[TestMethod]

public void Group\_RemoveMember\_RemovesBothSides()

{

var owner = new Member(1, "Owner", "o@o.com");

var bob = new Member(2, "Bob", "b@b.com");

var club = new Group(1, "Club", owner);

club.AddMember(bob);

club.RemoveMember(bob);

Assert.IsFalse(club.Members.Contains(bob));

Assert.IsFalse(bob.Groups.Contains(club));

}

/// <summary>Removing a non-member throws an exception.</summary>

[TestMethod]

public void Group\_RemoveMember\_NotFound\_Throws()

{

var owner = new Member(1, "Owner", "o@o.com");

var club = new Group(1, "Club", owner);

var stranger = new Member(2, "Stranger", "s@s.com");

Assert.ThrowsException<InvalidOperationException>(() =>

club.RemoveMember(stranger));

}

/// <summary>Removing a user from the network cleans friendships and groups.</summary>

[TestMethod]

public void RemoveMember\_CleansFriendshipsAndGroups()

{

var net = new SocialNetwork();

var alice = new Member(1, "Alice", "a@a.com");

var bob = new Member(2, "Bob", "b@b.com");

net.AddMember(alice);

net.AddMember(bob);

alice.AddFriend(bob);

var club = net.CreateGroup("Club", alice);

club.AddMember(bob);

net.RemoveMember(bob.Id);

Assert.IsFalse(net.Members.Contains(bob));

Assert.AreEqual(0, alice.GetFriends().Count);

Assert.IsFalse(club.Members.Contains(bob));

}

/// <summary>An unknown Id passed to RemoveMember must raise an exception.</summary>

[TestMethod]

public void RemoveMember\_NotFound\_Throws()

{

var net = new SocialNetwork();

Assert.ThrowsException<InvalidOperationException>(() =>

net.RemoveMember(42));

}

}

#endregion

}

//MSTestSettings.cs

[assembly: Parallelize(Scope = ExecutionScope.MethodLevel)]

// Проекти

//SocialNetworkLibrary.Tests.csproj

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>net8.0</TargetFramework>

<LangVersion>latest</LangVersion>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

<EnableMSTestRunner>true</EnableMSTestRunner>

<OutputType>Exe</OutputType>

<TestingPlatformDotnetTestSupport>true</TestingPlatformDotnetTestSupport>

<!--

Displays error on console in addition to the log file. Note that this feature comes with a performance impact.

For more information, visit https://learn.microsoft.com/dotnet/core/testing/unit-testing-platform-integration-dotnet-test#show-failure-per-test

-->

<TestingPlatformShowTestsFailure>true</TestingPlatformShowTestsFailure>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.14.1" />

<PackageReference Include="Microsoft.Testing.Extensions.CodeCoverage" Version="17.14.2" />

<PackageReference Include="Microsoft.Testing.Extensions.TrxReport" Version="1.7.2" />

<PackageReference Include="MSTest" Version="3.9.2" />

</ItemGroup>

<ItemGroup>

<ProjectReference Include="..\SocialNetworkLibrary\SocialNetworkLibrary.csproj" />

</ItemGroup>

<ItemGroup>

<Using Include="Microsoft.VisualStudio.TestTools.UnitTesting" />

</ItemGroup>

</Project>

//SoicalNetworkLibrary.csproj

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

//SocialNetworkApp.csproj

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>WinExe</OutputType>

<TargetFramework>net8.0-windows</TargetFramework>

<Nullable>enable</Nullable>

<ImplicitUsings>enable</ImplicitUsings>

<UseWPF>true</UseWPF>

</PropertyGroup>

<ItemGroup>

<ProjectReference Include="..\SocialNetworkLibrary\SocialNetworkLibrary.csproj" />

</ItemGroup>

</Project>