НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
інформаційних систем і технологій

_____ Швиденко М.З., к.е.н., доц.
(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Проектування та розробка платформи соціальної взаємодії»

Спеціальність 122 «Комп'ютерні науки»

Гарант освітньої програми

к.т.н. доцент _____ Руденський Р.А.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

д-р філос., доц. _____ Корольчук В.І.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Кикоть Ю.О.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
інформаційних систем і технологій

_____/ Швиденко М.З., к.е.н., доц. /

підпис

“_ ”_____2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студент Кикоть Юрій Олексійович

Спеціальність 122 «Комп’ютерні науки»

Тема бакалаврської кваліфікаційної роботи: Проектування та розробка платформи соціальної взаємодії

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2246 “С”

Термін подання завершеної роботи на кафедру _____

(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань , які потрібно розробити:

Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання “16” 12 2024р.

Керівник бакалаврської кваліфікаційної роботи

____д-р філос., доц._____ _____ ___Корольчук В.І._

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Кикоть Ю.О.

(підпис) (ПІБ студента

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 5
ВСТУП 6
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Опис предметної області 9
1.2 Моделювання предметної області 11
1.3 Огляд інформаційних джерел та існуючих рішень 16
1.4 Аналіз вимог до програмної системи 21
1.5 Постановка завдання 24
2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 27
2.1 Логічна модель даних у вигляді ER-діаграми 27
2.2 Діаграма класів та кооперації 29
2.3 Діаграма пакетів системи 32
2.4 Діаграма компонентів 35
3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 39
3.1 Система управління базою даних 39
3.2 Розробка інформаційної бази 41
3.3 Архітектура програмного забезпечення 43
3.4 Вибір інструментарію для створення прикладного програмного забезпечення 46
3.5 Алгоритми обробки даних та побудови родинних структур 48
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ І ЕКСПЛУАТАЦІЇ ПРОГРАМНОЇ СИСТЕМИ 51
4.1 Тестування системи 51
4.2 Вимоги до апаратного та програмного забезпечення 54
4.3 Склад інсталяційного пакету 56

ВИСНОВКИ 58
СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТОК А 63
ДОДАТОК Б 66

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

– GUI — Graphical User Interface, графічний інтерфейс користувача.

– JSON — JavaScript Object Notation, текстовий формат представлення структурованих даних.

– MVC — Model–View–Controller, архітектурний шаблон проєктування програмного забезпечення.

– UML — Unified Modeling Language, уніфікована мова моделювання.

– СУБД — система управління базами даних.

– SQLite — вбудована реляційна СУБД, що не потребує серверного середовища.

– DOT — формат графового представлення структури (використовується у Graphviz).

– HTTPS — Hypertext Transfer Protocol Secure, захищений мережевий протокол передачі даних.

– RAM — Random Access Memory, оперативна пам’ять комп’ютера.

– ОС — операційна система.

– ПК — персональний комп’ютер.

– UX — User Experience, досвід користувача при взаємодії з інтерфейсом.

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та глобальної цифровізації соціальні мережі стали невід'ємною частиною повсякденного життя мільйонів користувачів у всьому світі. Вони не лише виконують функцію комунікаційного середовища, а й трансформуються у багатофункціональні платформи для обміну інформацією, ведення бізнесу, організації спільнот та збереження особистих даних. Зростання популярності таких сервісів обумовлює актуальність створення нових рішень, орієнтованих на задоволення специфічних потреб користувачів, зокрема, персоналізованих функціоналів для збереження родинних зв'язків та побудови сімейних історій [5], [6].

Актуальність теми зумовлена відсутністю у більшості сучасних соціальних платформ інструментів, що дозволяють інтегровано та зручно формувати сімейні дерева в рамках єдиного цифрового простору. Популярні сервіси або взагалі не підтримують подібний функціонал, або реалізують його у вигляді окремих спеціалізованих програм, що не мають соціальної складової у традиційному розумінні [7]. Таким чином, розробка соціальної мережі з вбудованим модулем побудови сімейного дерева є важливим напрямом для задоволення запитів користувачів, зацікавлених у збереженні родоводів, організації сімейних спільнот та підтримці комунікації між поколіннями.

Підставами для розробки даної тематики є поєднання тенденцій персоналізації цифрового контенту, розвитку засобів візуалізації складних структур (зокрема графів і дерев) та потреби суспільства у збереженні культурної й родинної спадщини [1], [3]. Вихідними даними для реалізації проєкту стали методології об'єктно-орієнтованого аналізу та проєктування, зокрема використання UML-діаграм для моделювання архітектури системи [1, а також інструменти графічної побудови деревоподібних структур .

Завданнями кваліфікаційної роботи є:

1. Виконати системний аналіз предметної області та визначити ключові характеристики програмної системи;
2. Проаналізувати вимоги до програмної системи та сформувати їх специфікацію.
3. Провести моделювання предметної області із використанням UML та ER-діаграм.
4. Здійснити огляд існуючих інформаційних джерел та програмних рішень у сфері соціальних мереж і генеалогічних систем;
5. Розробити логічну модель даних та побудувати основні діаграми (класи, пакети, компоненти) для проєктування системи;
6. Вибрати інструментарій та реалізувати інформаційне і програмне забезпечення, включаючи базу даних та прикладні модулі;
7. Надати рекомендації щодо впровадження, тестування та подальшої експлуатації розробленої програмної системи.

Метою дослідження є розробка концепції та реалізація програмного забезпечення соціальної мережі із вбудованим функціоналом побудови сімейного дерева, що забезпечує інтерактивну взаємодію користувачів, збереження родинних зв’язків та зручну візуалізацію генеалогічної інформації.

Об’єктом дослідження є процеси проєктування та розробки веборієнтованих соціальних платформ із розширеним функціоналом управління персональними даними.

Предметом дослідження виступають методи та засоби побудови архітектури соціальної мережі з інтегрованим модулем формування сімейного дерева, включаючи моделювання взаємозв’язків, візуалізацію структур та забезпечення зручності користувацької взаємодії.

Наукова новизна роботи полягає у розробці унікальної соціальної мережі, із можливістю масштабування, редагування та взаємодії між користувачами в реальному часі. Запропоновано архітектурні рішення (модульну архітектуру), які поєднують класичні принципи побудови соціальних платформ із сучасними засобами візуалізації деревоподібних структур [6], [11], [18].

Апробація результатів дослідження здійснювалась у рамках тестової реалізації прототипу системи, що включає базовий функціонал реєстрації користувачів, створення профілів, встановлення родинних зв’язків та автоматичної генерації сімейного дерева за заданими параметрами. Візуалізація генеалогічних даних реалізована із використанням інструментів PlantUML та Graphviz [9], [10], що дозволило досягти гнучкості у відображенні складних структур.

Структура роботи відображає логіку проведеного дослідження і складається з трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі наведено теоретичні основи створення соціальних мереж і методи візуалізації родинних структур.

Другий розділ присвячений аналізу інструментальних засобів та побудові UML-моделей системи.

Третій розділ містить опис процесу розробки програмного забезпечення, тестування функціоналу та аналіз отриманих результатів.

Четвертий розділ містить опис тестування системи, розроблену діаграму розгортання

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Дедалі більшої популярності набувають програмні рішення, орієнтовані на персоналізоване збереження та управління інформацією користувачів. Соціальні мережі залишаються ключовими платформами для комунікації, однак більшість із них функціонують у форматі веборієнтованих сервісів, що передбачають централізоване зберігання даних і залежність від інтернет-з'єднання [6], [7]. Водночас зростає потреба у локальних програмних засобах, які забезпечують автономність, підвищену конфіденційність і можливість індивідуального налаштування функціоналу.

Предметна область даного дослідження стосується розробки десктопної програми на Python, яка поєднуватиме елементи соціальної мережі з функціоналом побудови та управління сімейним деревом. На відміну від класичних соціальних платформ, запропоноване рішення працюватиме локально, що забезпечить повний контроль користувача над персональними даними без потреби постійного доступу до мережі Інтернет.

Майбутній програмний продукт являтиме собою кросплатформену програму, розроблену із використанням мови Python та відповідних бібліотек для реалізації графічного інтерфейсу користувача (GUI), управління базами даних та візуалізації графів. Основні компоненти системи включатимуть:

1. Локальну інформаційну базу, побудовану на основі вбудованої СУБД, для збереження профілів користувачів, їх соціальних зв'язків та генеалогічної інформації [21].

2. Модуль управління соціальними зв'язками, що дозволяє створювати локальну соціальну мережу: додавати профілі знайомих, друзів, родичів, організовувати спільноти та вести переписку у межах програми.

3. Генеалогічний модуль, орієнтований на побудову сімейного дерева із динамічною візуалізацією родинних зв’язків. Для цього планується використання бібліотек Graphviz і networkx для створення та рендерингу графічних структур [9], [10].

4. Графічний інтерфейс користувача, реалізований за допомогою PyQt, що забезпечить інтуїтивно зрозуміле керування функціоналом програми та доступ до основних сервісів [24].

Особливістю предметної області є необхідність організації ефективного зберігання та обробки структурованих даних про взаємозв’язки між користувачами, а також забезпечення гнучкої системи візуалізації складних деревоподібних структур [11]. При цьому важливо врахувати принципи об’єктно-орієнтованого проєктування для побудови модульної архітектури програмного забезпечення, що сприятиме подальшій підтримці та розширенню функціоналу [16].

Таким чином, розробка десктопної програми дозволить реалізувати автономний інструмент для:

- ведення локальної соціальної взаємодії без використання зовнішніх серверів;
- створення та редагування персонального сімейного дерева;
- забезпечення конфіденційності та безпеки даних користувача;
- інтеграції інструментів аналітики для вивчення структури соціальних і родинних зв’язків.

У межах цієї предметної області програмний продукт буде орієнтований як на приватних користувачів, що бажають зберігати інформацію про власну родину та соціальне оточення, так і на невеликі організації або спільноти, яким необхідний локальний засіб для управління взаємозв’язками без залучення сторонніх онлайн-сервісів.

1.2 Моделювання предметної області

У межах даного проєкту було виконано комплексне моделювання предметної області майбутньої десктопної програми на Python, основною особливістю якої є інтеграція функціоналу соціальної мережі з можливістю побудови та управління сімейним деревом. Програма покликана забезпечити користувачам зручний інструмент для комунікації, ведення персональних профілів, обміну інформацією та, водночас, надавати потужний механізм для створення інтерактивного родинного дерева з візуалізацією взаємозв'язків. Враховуючи багатокомпонентну структуру системи, моделювання дозволило сформувати чітке уявлення про її архітектуру, визначити ключові функціональні модулі, їхні взаємозв'язки та сценарії використання.

Для досягнення максимальної формалізації концепції програмного забезпечення було застосовано різні типи UML-діаграм, оскільки кожен вид діаграми слугує для відображення конкретного аспекту роботи системи. Зокрема, діаграми прецедентів використовувалися для опису взаємодії користувачів із програмою, що дозволило визначити ролі та доступний функціонал для кожної категорії користувачів. Діаграми послідовності дали можливість деталізувати процеси обміну повідомленнями між компонентами системи під час виконання основних операцій, таких як додавання родича до сімейного дерева. Водночас, діаграми активності описали логіку виконання бізнес-процесів, враховуючи як стандартні сценарії, так і можливі відхилення у випадку помилок або некоректних дій користувача.

На першому етапі було розроблено діаграму прецедентів (use case diagram), яка ілюструє взаємодію основних користувачів із системою та доступні для них функції. На рис. 1.1 наведено основні сценарії використання програми для трьох типів акторів: Користувач, Гість та Адміністратор.

Рис. 1.1. Діаграма прецедентів взаємодії користувачів із системою

Як видно з діаграми (рис. 1.1), Користувач має доступ до базового функціоналу соціальної мережі: створення публікацій, коментування, додавання друзів, обміну повідомленнями та ключової функції – побудови сімейного дерева. Крім того, передбачені операції авторизації. Гість може переглядати публічні профілі та реєструватися в системі, а Адміністратор виконує модерацію контенту, управління обліковими записами та перегляд звітності. Такий розподіл ролей забезпечує чітку ієрархію доступу та функціональних можливостей.

Другим етапом стало створення діаграми послідовності (sequence diagram), яка описує процес додавання нового родича до сімейного дерева. На рис. 1.2 зображено детальний сценарій взаємодії користувача з інтерфейсом програми та внутрішніми модулями.

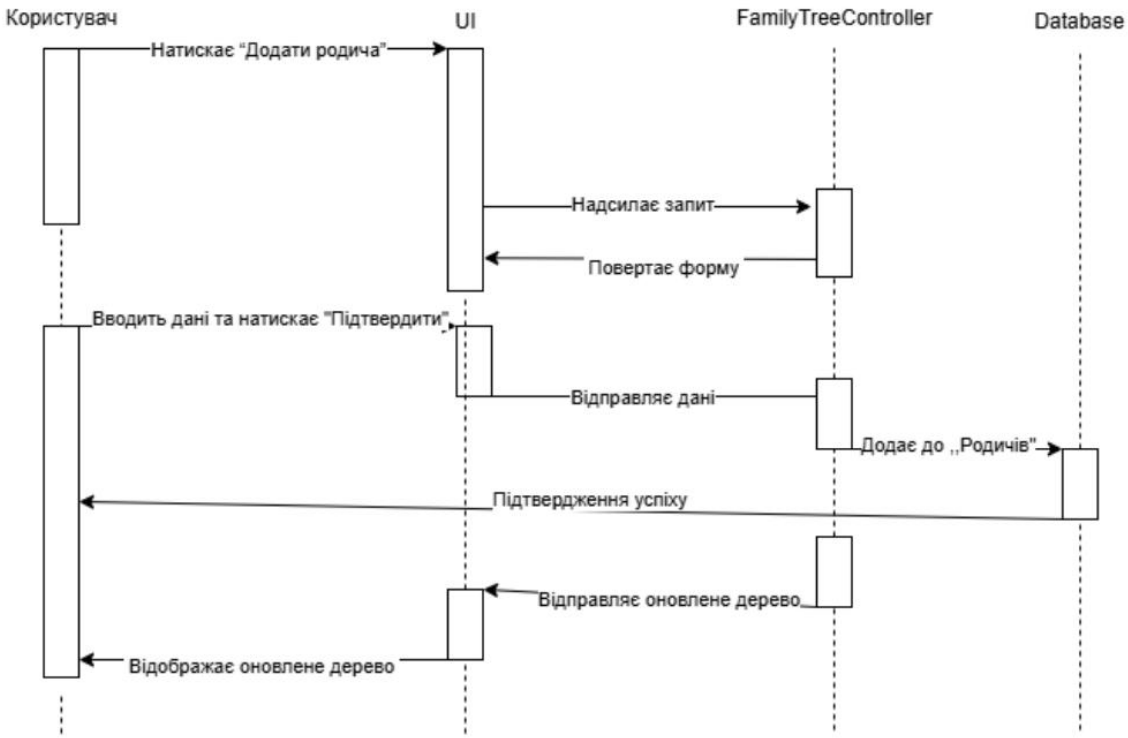

Рис. 1.2. Діаграма послідовності процесу додавання родича до сімейного дерева

Дана діаграма (рис. 1.2) демонструє типовий алгоритм взаємодії користувача із системою під час виконання операції додавання нового родича до сімейного дерева. Процес розпочинається з ініціації дії користувачем через графічний інтерфейс (UI), коли він натискає кнопку "Додати родича". У відповідь система відправляє запит до модуля логіки — FamilyTreeController, який відповідає за обробку бізнес-логіки, зокрема, за керування процесами, пов'язаними з модифікацією структури сімейного дерева.

Контролер формує та повертає користувачу динамічну форму для введення необхідних даних про нового родича (ПІБ, тип родинного зв’язку, дата народження тощо). Після заповнення форми та підтвердження введених даних, UI надсилає отриману інформацію назад до FamilyTreeController. На цьому етапі контролер здійснює перевірку коректності та повноти даних відповідно до встановлених правил валідації.

У разі успішної перевірки, контролер ініціює транзакцію до бази даних, де новий запис додається до таблиці «Родичі» або до відповідної структури у графовій моделі даних. Після завершення операції оновлення, контролер формує повідомлення про успішне виконання дії та надсилає його до UI. Одночасно система генерує оновлене представлення сімейного дерева, яке також передається на рівень інтерфейсу для візуалізації користувачу в інтерактивному вигляді.

Така модель чітко відображає принцип розподілу відповідальності між компонентами системи:

- UI відповідає за взаємодію з користувачем і відображення інформації;
- FamilyTreeController забезпечує обробку логіки і є посередником між інтерфейсом та базою даних;
- Database виконує функцію довгострокового зберігання структурованих даних.

Послідовність викликів та обміну повідомленнями, представлена на діаграмі, дозволяє мінімізувати ризики виникнення помилок, забезпечити узгодженість даних і підвищити гнучкість системи для подальшої модифікації функціоналу. Такий підхід відповідає принципам MVC-архітектури (Model-View-Controller), яка є рекомендованою для побудови масштабованих програмних рішень [14], [17].

Крім того, ця модель дозволяє легко інтегрувати додаткові функції, наприклад, логування дій користувачів, перевірку прав доступу або відправлення сповіщень після змін у дереві, що підвищує загальну ефективність і безпеку роботи програми.

На третьому етапі було побудовано діаграму активності (activity diagram), яка відображає логіку дій користувача під час роботи з функціоналом побудови сімейного дерева (рис. 1.3).

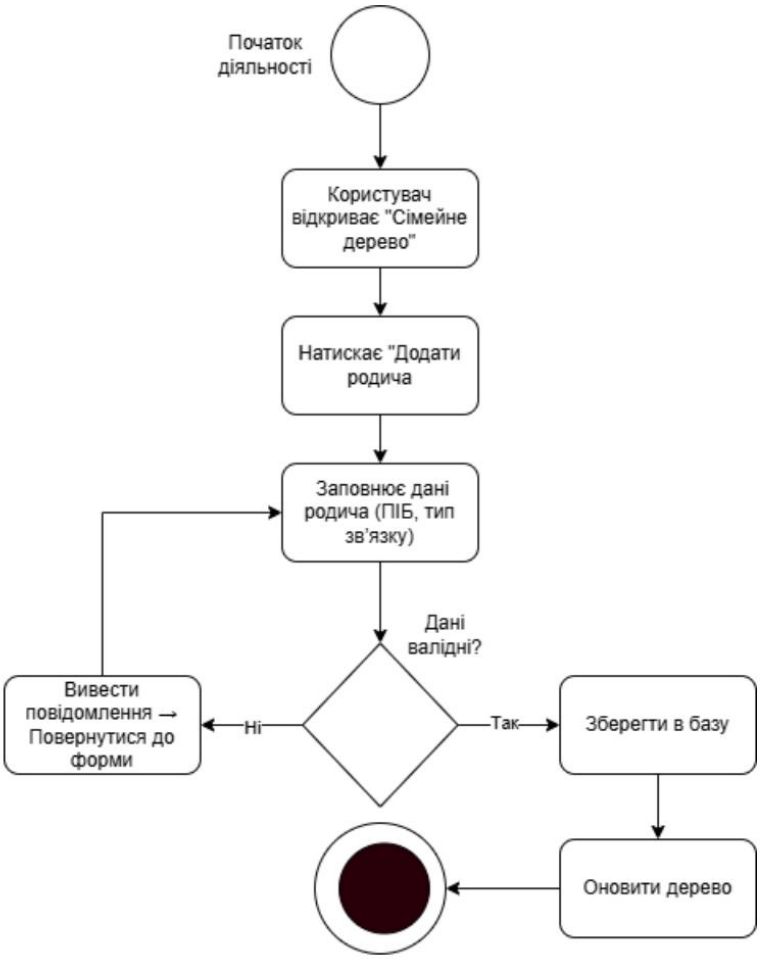

Рис. 1.3 . Діаграма активності процесу додавання родича

Діаграма (рис. 1.3) ілюструє основні кроки, які виконує користувач: відкриття відповідного модуля, ініціалізація процесу додавання родича, введення даних, перевірка їх валідності та подальше збереження у базі даних із оновленням візуального представлення дерева. У разі виявлення помилок у введених даних система повертає користувача до етапу коригування інформації. Такий підхід дозволяє описати не лише прямий сценарій, але й альтернативні гілки розвитку подій.

Комплексне використання різних типів діаграм у моделюванні предметної області дозволило сформувати цілісне уявлення про структуру та поведінку майбутньої системи. Діаграми забезпечують:

– візуалізацію ролей і прав доступу користувачів (рис. 1.1);
– Чіткий опис взаємодії між інтерфейсом і логікою програми (рис. 1.2);
– Формалізацію бізнес-процесів і сценаріїв використання (рис. 1.3).

1.3 Огляд інформаційних джерел та існуючих рішень

Розробка програмного забезпечення завжди базується на детальному аналізі наявних інформаційних джерел та вивченні існуючих аналогів. Це дозволяє не лише уникнути повторення вже реалізованих рішень, але й виявити недоліки та обмеження сучасних систем, що відкриває можливості для впровадження інноваційного функціоналу. У рамках даного дослідження було проаналізовано ряд популярних платформ, які поєднують у собі елементи соціальної взаємодії та генеалогічного аналізу.

Однією з найбільш відомих соціальних мереж є Facebook - глобальна платформа, яка забезпечує багатофункціональне середовище для комунікації, обміну контентом та організації спільнот. На рис. 1.4 представлено інтерфейс Facebook у темній темі, що демонструє класичний набір функцій: стрічку новин, пости користувачів, перегляд подій і можливість інтеграції з іншими сервісами.

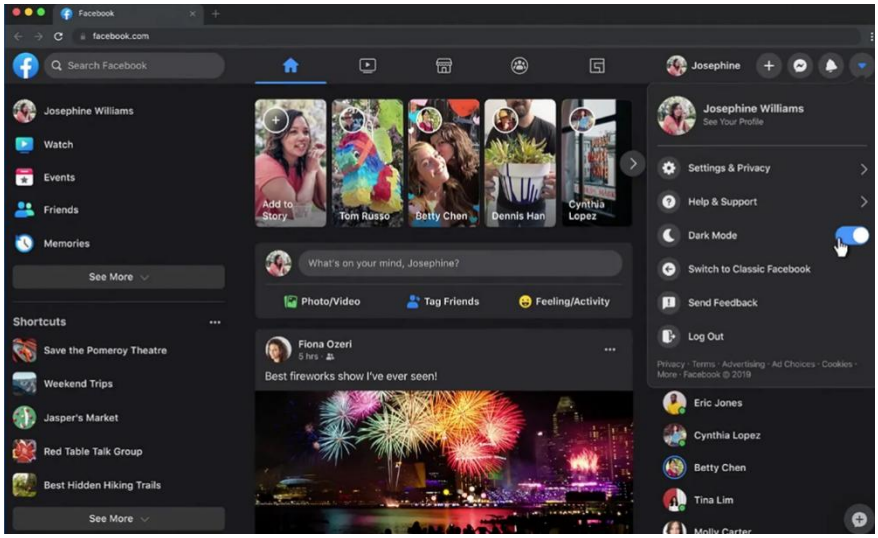

Рис. 1.4. Інтерфейс соціальної мережі Facebook

Важливо зазначити, що хоча Facebook дозволяє встановлювати родинні зв’язки між користувачами у профілі, однак повноцінного функціоналу для побудови сімейного дерева ця платформа не містить. Взаємозв’язки не візуалізуються у вигляді графа, що унеможливлює використання Facebook для систематизації родинної інформації [6].

Іншим прикладом є сервіс MyHeritage, який спеціалізується на генеалогічних дослідженнях. Він надає користувачам можливість будувати родинні дерева, зберігати історичні записи та навіть проводити ДНК-тести для визначення етнічного походження. На рис. 1.5 зображено інтерфейс модуля аналізу ДНК у MyHeritage.

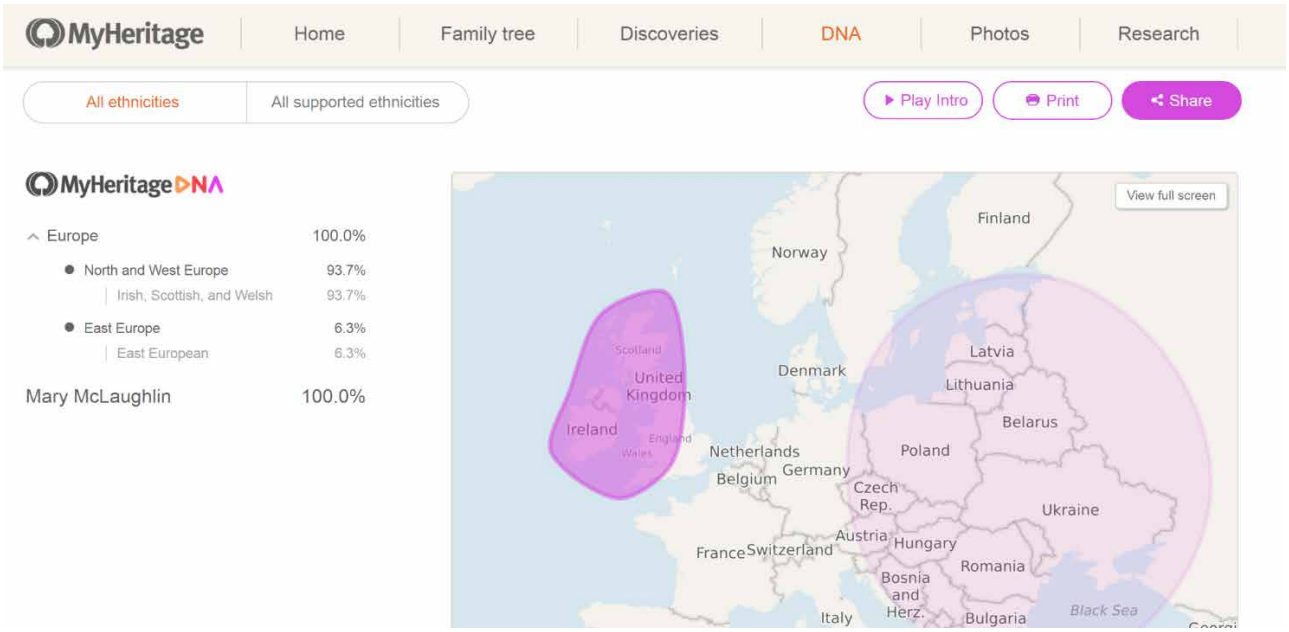

Рис. 1.5. Функціонал генеалогічного аналізу у MyHeritage

Попри широкі можливості у сфері генеалогії, MyHeritage не є соціальною мережею у класичному розумінні. Його комунікаційні функції обмежуються обміном інформацією між членами родинного дерева, без створення соціальних груп чи публічних спільнот [5].

Ще одним прикладом є платформа Geni, яка позиціонує себе як соціальна мережа для побудови колективних сімейних дерев. На рис. 1.6 показано приклад розгалуженої структури дерева у Geni, де користувачі можуть додавати родичів і співпрацювати з іншими учасниками для розширення єдиної глобальної генеалогічної бази.

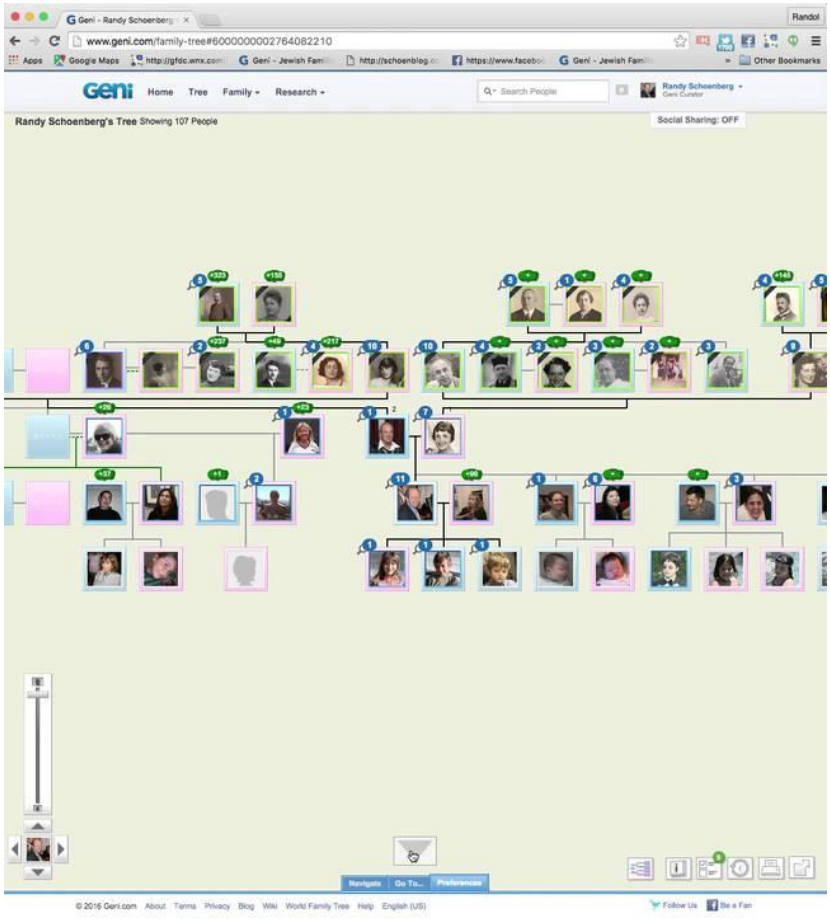

Рис. 1.6 . Побудова сімейного дерева у Geni

Характерною рисою Geni є орієнтація на спільну роботу та об’єднання родоводів, проте платформа залишається суто веборієнтованою і не надає можливостей для автономного використання чи локального збереження даних. Крім того, функціонал соціальної взаємодії є доволі обмеженим порівняно з традиційними соціальними мережами [7].

Ще одним відомим рішенням є сервіс FamilySearch, який, як і Geni, зосереджений на створенні родинних дерев та обміні генеалогічною інформацією. На рис. 1.7 представлено інтерфейс FamilySearch із відображенням родинних зв’язків.

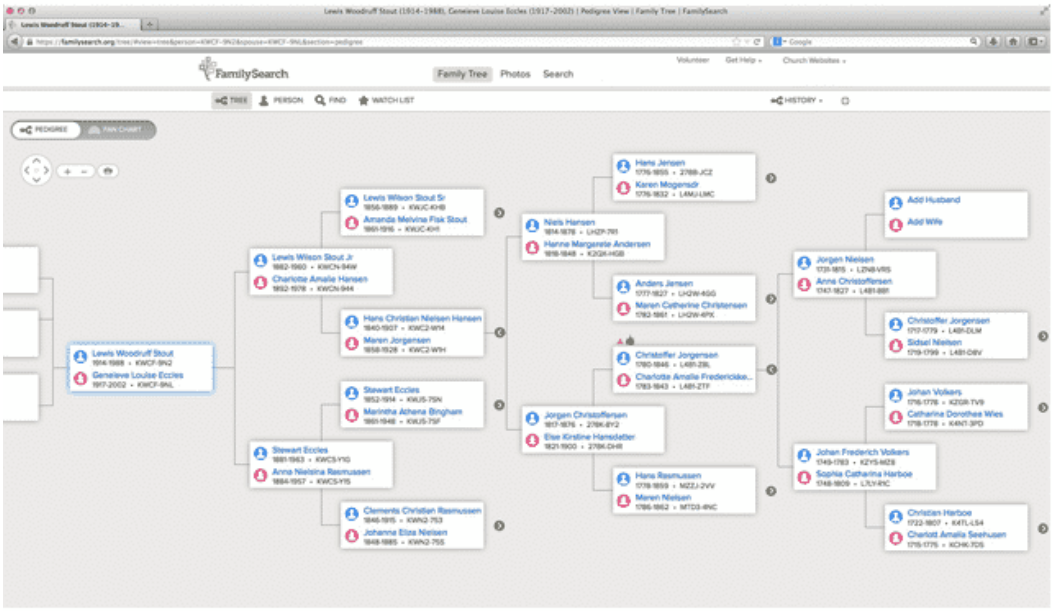

Рис. 1.7 . Відображення родинних зв’язків у FamilySearch

FamilySearch є безкоштовною платформою, яка активно використовується для дослідницької роботи у сфері генеалогії, однак не містить розвинених інструментів соціальної взаємодії. Основна увага зосереджена на роботі з архівами та історичними записами.

Для систематизації результатів аналізу доцільно виконати порівняння основних характеристик розглянутих платформ за ключовими критеріями, що мають значення для проєктування майбутнього програмного забезпечення. До таких критеріїв віднесено: тип платформи, наявність соціального функціоналу, можливість побудови сімейного дерева, режим роботи (онлайн/офлайн), рівень приватності даних та орієнтацію на колективну чи індивідуальну роботу.

Таблиця 1.1

Порівняння функціональних можливостей існуючих рішень

№	Система	Тип платформи	Соц. функції	Сімейне дерево	Режим роботи	Приватність даних	Спільна робота
1	Facebook	Веб-сервіс	✓	✗	Онлайн	Низька	✓
2	MyHeritage	Веб + десктоп	Обмежено	✓	Онлайн	Середня	✓
3	Geni	Веб-сервіс	Обмежено	✓	Онлайн	Низька	✓

Продовження таблиці 1.1

4	FamilySearch	Веб-сервіс	✗	✓	Онлайн	Висока	✓
5	Розроблювана система	Десктоп (локальна)	✓	✓	Офлайн	Висока	Індивідуально

Як видно з таблиці 1.1, усі існуючі рішення функціонують переважно у вебсередовищі та вимагають постійного підключення до Інтернету. Лише MyHeritage частково підтримує десктопну версію, але її функціонал обмежений і орієнтований на синхронізацію з хмарними сервісами. Водночас лише Facebook пропонує повноцінні соціальні можливості, проте зовсім не має інструментів для побудови родинних дерев.

Аналізуючи переваги кожної з платформ, доцільно виділити ті елементи, які можуть бути адаптовані у процесі розробки власної десктопної програми:

- від Facebook доцільно перейняти концепцію організації соціальної взаємодії між користувачами, зокрема механізми створення публікацій, коментування, додавання друзів і обміну повідомленнями [6].
- Від MyHeritage можна використати підхід до структурованого зберігання генеалогічних даних та реалізацію інструментів візуалізації сімейного дерева із можливістю розширення його функціоналу [5].
- Від Geni варто врахувати зручний інтерфейс побудови родинних зв’язків та механізми навігації по великих деревоподібних структурах [7].
- Від FamilySearch доцільно запозичити принципи забезпечення конфіденційності даних користувачів і можливість роботи з локальними копіями генеалогічної інформації.

Таким чином, розроблюване програмне забезпечення поєднає найкращі рішення, враховуючи при цьому потребу в автономності, високій приватності та кросплатформенності. Запропонована система буде позбавлена недоліків існуючих сервісів, зокрема залежності від мережі та обмеженого контролю користувачів над власними даними.

1.4 Аналіз вимог до програмної системи

Аналіз вимог є фундаментальним етапом розробки програмного забезпечення, оскільки саме на цьому етапі формується уявлення про функціональні можливості системи, обмеження, технічні параметри та очікування кінцевих користувачів. Коректно визначені вимоги забезпечують узгодженість між замовником і розробником, слугують основою для архітектурного проєктування, а також є критерієм для подальшого тестування та оцінки якості програмного продукту [16], [18].

Вимоги до програмної системи традиційно класифікуються на функціональні, нефункціональні, вимоги до безпеки та технічні вимоги. У межах цього пункту буде детально розглянуто кожну з категорій.

Функціональні вимоги визначають основні дії, які система повинна виконувати для задоволення потреб користувача. Вони описують зовнішню поведінку програми, її сервіси та реакції на ті чи інші події [17].

Таблиця 1.2

Функціональні вимоги до програмної системи

№	Вимога	Опис функціональності
1	Реєстрація та авторизація	Забезпечення створення облікового запису та входу в систему
2	Створення та редагування профілю	Можливість змінювати персональні дані користувача
3	Соціальна взаємодія	Додавання друзів, обмін повідомленнями, створення публікацій
4	Побудова сімейного дерева	Створення, редагування та візуалізація родинних зв’язків
5	Перегляд історії взаємодій	Відображення списку подій, повідомлень, змін у дереві
6	Адміністрування	Модерація контенту, управління користувачами

Запропонований функціонал охоплює як базові можливості соціальної мережі, так і спеціалізовані інструменти для роботи з генеалогічною інформацією, що дозволяє задовольнити широкий спектр потреб кінцевого користувача [6], [7].

Нефункціональні вимоги визначають якісні характеристики системи, такі як продуктивність, зручність використання, масштабованість та сумісність [16].

Таблиця 1.3

Нефункціональні вимоги до програмної системи

№	Категорія	Опис вимог
1	Продуктивність	Час відгуку інтерфейсу не більше 1 секунди
2	Зручність використання	Інтуїтивно зрозумілий GUI, реалізований на базі PyQt
3	Портативність	Підтримка Windows, Linux, macOS
4	Масштабованість	Можливість розширення функціоналу без зміни архітектури
5	Надійність	Стійкість до збоїв при некоректних діях користувача

Дотримання цих вимог гарантує стабільну та комфортну роботу програми у різних середовищах, а також забезпечує можливість подальшого розвитку продукту [18].

З огляду на те, що система працює з персональними та генеалогічними даними, питання безпеки є пріоритетним. Навіть у локальному середовищі необхідно забезпечити захист даних від несанкціонованого доступу [3].

Таблиця 1.4

Вимоги до безпеки програмної системи

№	Вимога	Опис
1	Аутентифікація	Захист доступу до облікового запису паролем
2	Шифрування	Збереження конфіденційної інформації у зашифрованому вигляді
3	Резервне копіювання	Автоматичне створення локальних резервних копій
4	Розмежування прав доступу	Відмінність між правами користувача та адміністратора
5	Захист від несанкціонованого редагування	Блокування критичних операцій без підтвердження

Застосування сучасних методів захисту дозволить мінімізувати ризики втрати або компрометації даних, навіть у разі фізичного доступу до пристрою сторонніми особами [9].

Технічні вимоги визначають апаратне та програмне забезпечення, необхідне для коректної роботи системи.

Таблиця 1.5

Технічні вимоги до програмної системи

№	Параметр	Значення
1	Операційна система	Windows 10+, Linux, macOS 10.13+
2	Процесор	Мінімум 2 ядра, 2.0 GHz
3	Оперативна пам’ять	Від 4 ГБ
4	Вільне місце на диску	Від 500 МБ
5	Програмне середовище	Python 3.10+, бібліотеки PyQt5, SQLite3, Graphviz
6	Додаткове ПЗ	Встановлений Graphviz для візуалізації дерев

Визначені технічні вимоги гарантують працездатність системи на більшості сучасних ПК без потреби у високопродуктивному обладнанні, що відповідає концепції доступного та універсального програмного забезпечення [10], [27].

Проведений аналіз вимог дозволив сформувати повний та структурований перелік характеристик, яким повинна відповідати розроблювана програмна система, що поєднує функціонал соціальної мережі та інструменти побудови сімейного дерева. Чітке розмежування функціональних і нефункціональних вимог забезпечує не лише розуміння основних сервісів та можливостей системи, але й визначає якісні показники, що впливають на зручність, продуктивність і стабільність її роботи в різних умовах експлуатації. Особливу увагу було приділено вимогам до безпеки, оскільки система працюватиме з конфіденційними персональними та генеалогічними даними, що потребує впровадження сучасних методів захисту інформації, таких як аутентифікація, шифрування та контроль доступу.

Окрім цього, врахування технічних вимог дозволяє забезпечити оптимальну працездатність програмного продукту на більшості сучасних операційних систем без потреби у ресурсомісткому апаратному забезпеченні, що підвищує доступність розроблюваної системи для широкого кола користувачів.

1.5 Постановка завдання

Постановка завдання є завершальним етапом системного аналізу, що визначає чіткі цілі, засоби реалізації та очікувані результати розробки програмного забезпечення. На основі проведеного аналізу предметної області, огляду існуючих рішень та сформованих вимог, визначено необхідність створення автономної десктопної програми, яка поєднуватиме функціонал соціальної мережі з інструментами побудови та управління сімейним деревом.

На відміну від існуючих веборієнтованих сервісів, розроблюване програмне забезпечення орієнтоване на локальне зберігання даних, що забезпечує високий рівень конфіденційності та незалежність від інтернет-з'єднання. Враховуючи зазначені особливості та сучасні тенденції у сфері програмної інженерії, для реалізації проєкту обрано мову програмування Python як одну з найпотужніших і гнучких платформ для розробки кросплатформених застосунків [16].

Python забезпечує низку переваг для створення таких систем, зокрема:

- високий рівень читабельності коду та швидкість розробки;
- Великий вибір бібліотек для роботи з графічними інтерфейсами, базами даних та візуалізацією графів;
- Кросплатформеність, що дозволяє запускати програму на різних операційних системах без суттєвих змін у коді;
- Активна спільнота підтримки та доступність відкритого програмного забезпечення [18].

Для створення графічного інтерфейсу користувача (GUI) у межах даного проєкту буде використано бібліотеку PyQt6, яка є сучасною обгорткою над фреймворком Qt. Вибір PyQt6 обумовлений такими факторами:

- підтримка побудови інтуїтивно зрозумілих та адаптивних інтерфейсів із використанням сучасних компонентів;
- Можливість реалізації багатовіконних застосунків із чіткою структурою взаємодії;

– Висока продуктивність та підтримка кастомізації елементів управління;

– Вбудована підтримка роботи з сигналами і слотами для організації подієво-орієнтованої архітектури [14].

Крім того, для візуалізації сімейного дерева планується інтеграція бібліотек Graphviz і networkx, які дозволяють генерувати та відображати графові структури, що є ключовим елементом у реалізації генеалогічного функціоналу [10].

Відповідно до сформульованих вимог, основне завдання дипломної роботи полягає у розробці десктопної програми на мові Python із графічним інтерфейсом, реалізованим засобами PyQt6, яка забезпечуватиме:

1. Реєстрацію, авторизацію та управління обліковими записами користувачів;
2. Можливість створення соціальних взаємодій: додавання друзів, обміну повідомленнями, публікацій та коментарів;
3. Формування, редагування та візуалізацію сімейного дерева із динамічним оновленням структури;
4. Зберігання всіх даних у локальній базі даних (SQLite), що гарантує автономність роботи програми;
5. Реалізацію адміністративного функціоналу для модерації контенту та управління користувачами;
6. Забезпечення високого рівня безпеки шляхом використання механізмів аутентифікації, шифрування даних та розмежування прав доступу.
7. Додатково передбачено реалізацію:
8. Механізму експорту сімейного дерева у графічний формат (PNG, PDF);
9. Логування дій користувача для підвищення прозорості роботи системи;
10. Адаптації інтерфейсу під різні роздільні здатності екранів та тем оформлення (світла/темна тема).

Розробка цієї системи охоплюватиме всі етапи життєвого циклу програмного забезпечення — від проєктування до реалізації та тестування, з акцентом на використання інструментальних засобів Python та PyQt6 як основних платформних рішень. Реалізація поставленого завдання дозволить створити конкурентоспроможний програмний продукт, що поєднує сучасні підходи до соціальної взаємодії та управління генеалогічною інформацією в локальному середовищі [5], [6], [16].

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

логічна модель даних є основою для побудови реляційної структури зберігання інформації в інформаційній системі, що поєднує функціонал соціальної мережі та інструменти побудови родинного дерева. Для її формалізації було створено ER-діаграму, яка ілюструє основні сутності, їх атрибути та міжтабличні зв’язки, з урахуванням принципів нормалізації, цілісності та масштабованості представлена на рисунку 2.1.

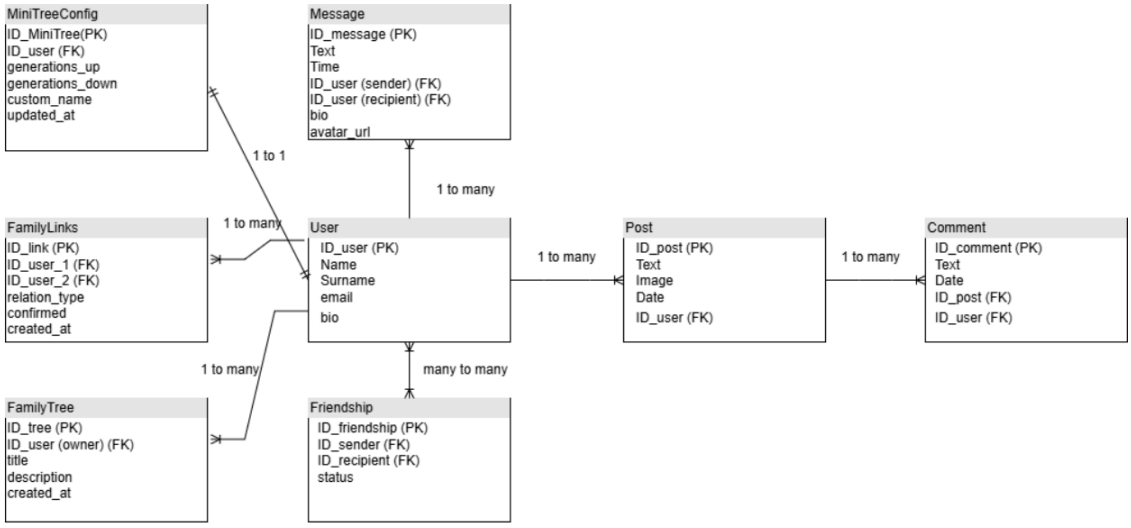

Рис. 2.1 . ER-діаграма проектованої системи

Візуалізація реалізована з дотриманням рекомендацій щодо моделювання логічного рівня згідно з UML-практиками [16; 18]. У системі використано сутності, що забезпечують реалізацію ключового функціоналу: збереження користувачів, взаємодію між ними, побудову генеалогічної структури, створення контенту та керування конфігураціями дерева.

Всі сутності пов’язані між собою зв’язками типу один до одного, один до багатьох або багато до багатьох, що забезпечує гнучкість доступу до інформації і підтримку багатовимірних взаємозалежностей між даними. Так, наприклад, між

користувачами реалізовано одночасно два типи зв’язків — соціальний (Friendship) та родинний (FamilyLinks), кожен з яких має окремі ознаки, що визначають тип зв’язку та його стан.

Операції обміну повідомленнями, публікацій і коментування реалізовані як типові односторонні відношення, що мають зовнішні ключі до основної сутності користувача. Родинне дерево задається як окрема логічна одиниця, що може бути створена будь-яким користувачем, а набір зв’язків реалізується через окрему таблицю зв’язків, у якій зберігається тип спорідненості та факт підтвердження. Модель також містить допоміжну таблицю з конфігурацією вигляду дерева, що дозволяє кожному користувачу налаштувати кількість відображуваних поколінь та інші параметри візуалізації. Усі ці структурні елементи представлені на рис. 2.1.

Для більш формального представлення сутностей та їх функціонального призначення доцільно звернутися до табл. 2.1, у якій узагальнено призначення кожної таблиці, її ключові поля та тип взаємозв’язків у системі.

Таблиця 2.1

Опис сутностей логічної моделі даних

№	Назва таблиці	Ключові поля	Функціональне призначення
1	User	ID_user (PK), name, surname, email	Зберігання облікових записів користувачів і базових персональних даних
2	Post	ID_post (PK), ID_user (FK)	Збереження публікацій користувачів, текстового та графічного контенту
3	Comment	ID_comment (PK), ID_post (FK), ID_user	Коментування публікацій, зв’язок з постом і автором коментаря
4	Message	ID_message (PK), sender, recipient	Обмін приватними повідомленнями між користувачами
5	Friendship	ID_friendship (PK), ID_sender, recipient	Модель багатосторонніх соціальних зв’язків (дружба) між користувачами
6	FamilyTree	ID_tree (PK), ID_user (owner)	Логічне представлення родинного дерева користувача

Продовження таблиці 2.1

7	FamilyLinks	ID_link (PK), ID_user_1, ID_user_2	Встановлення родинних зв’язків між користувачами (тип зв’язку, підтвердження)
8	MiniTreeConfig	ID_MiniTree (PK), ID_user (FK)	Індивідуальні налаштування вигляду родинного дерева (покоління, назва, дата оновлення)

Наведена модель повністю задовольняє вимоги до функціональності системи, забезпечуючи ізоляцію даних, підтримку множинних типів зв’язків та можливість масштабування. Вона формує логічний рівень, що безпосередньо лягає в основу реляційної реалізації та сумісна з об’єктно-орієнтованим підходом до реалізації інтерфейсу системи, який відповідає архітектурі MVC [1; 17; 20].

2.2 Діаграма класів та кооперації

Для відображення структури програмної системи на рівні об'єктно-орієнтованого проектування використано діаграму класів, яка формалізує основні класи, їх атрибути, методи, зв’язки наслідування, асоціації, а також залежності між модулями. Побудована модель орієнтована на реалізацію архітектури типу MVC, у якій інтерфейс, логіка й джерела даних функціонують із чітким розподілом відповідальності [18]. На основі вихідного коду системи було визначено ключові модулі: FramelessLogin, MainWindow, AuthManager, ChatManager, TreeManager, а також відповідні джерела даних у форматі .json і текстових логів. Структура взаємозв’язків між компонентами відображена на рис. 2.2.

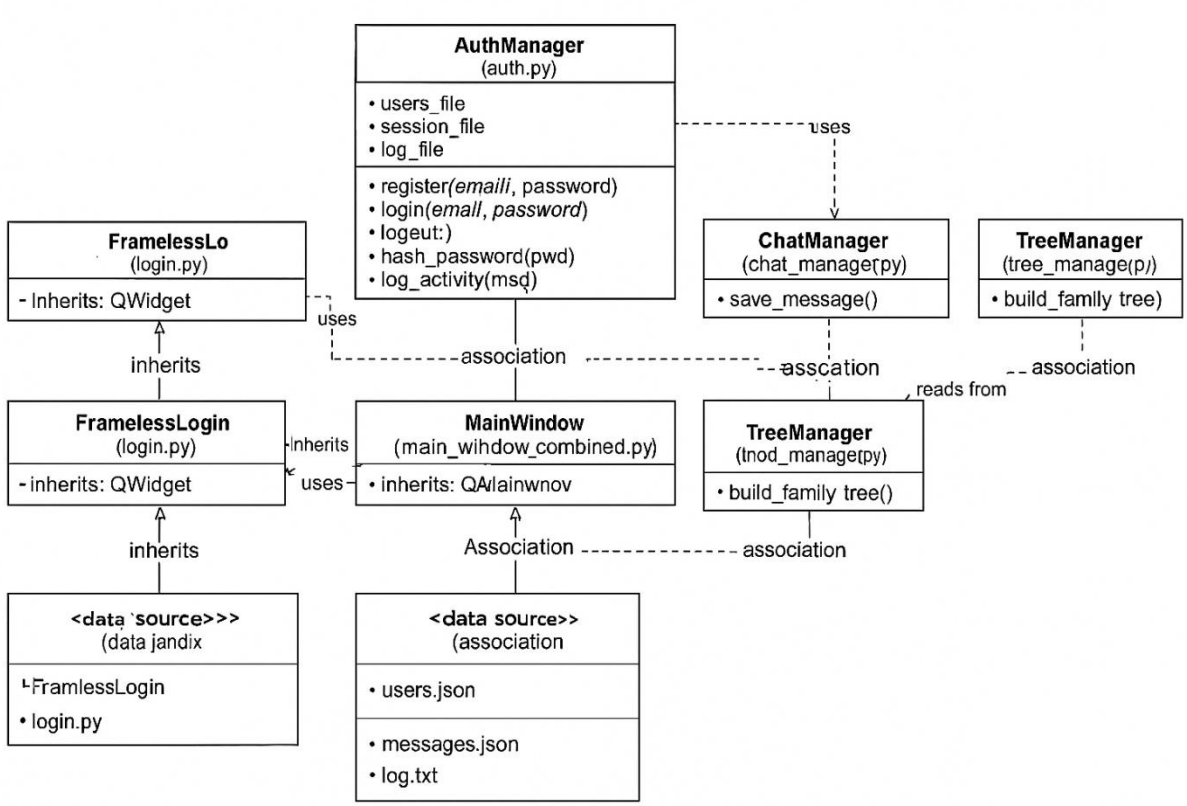

Рис. 2.2 . Діаграма класів системи з асоціаціями

На діаграмі класів (рис. 2.2) видно, що класи FramelessLogin та MainWindow наслідують функціонал від базового класу QWidget, реалізованого у рамках фреймворку PyQt6. Це забезпечує графічну відображуваність компонентів інтерфейсу та дозволяє реалізувати подієво-орієнтовану архітектуру. Обидва ці класи утворюють асоціації з сервісними класами логіки — AuthManager, ChatManager та TreeManager, через які реалізується функціональність реєстрації, повідомлень та побудови дерева відповідно. Для зберігання даних використовуються зовнішні джерела типу users.json, messages.json, session.json, що зчитуються або оновлюються відповідними сервісами. Така модель забезпечує низький рівень зв’язності між модулями та сприяє розширюваності проєкту [3; 16].

Для конкретизації логіки виконання взаємодій між об'єктами системи використано діаграми кооперації (sequence diagrams), що демонструють порядок виклику методів під час реалізації ключових сценаріїв. На рис. 2.3 представлено механізм надсилання повідомлення користувачем через головне вікно інтерфейсу.

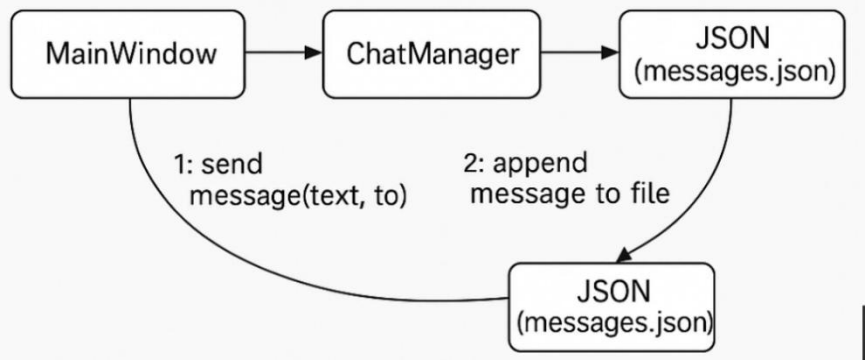

Рис. 2.3. Послідовність взаємодії між інтерфейсом і модулем збереження повідомлень

У відповідності до рис. 2.3, клас MainWindow ініціює передачу повідомлення, викликаючи метод send_message() класу ChatManager, який, у свою чергу, зберігає повідомлення в локальному JSON-файлі (messages.json) через додавання нового запису до масиву. Така логіка відображає базовий механізм асинхронної взаємодії між модулями та реалізує шаблон «контролер-обробник-джерело даних». Важливо, що файл не перезаписується повністю, а оновлюється з мінімальним втручанням, що підвищує продуктивність роботи із збереженням повідомлень [7].

Другий ключовий сценарій пов’язаний із реєстрацією нового користувача в системі. На рис. 2.4 представлено послідовність операцій при створенні нового облікового запису через клас FramelessLogin.

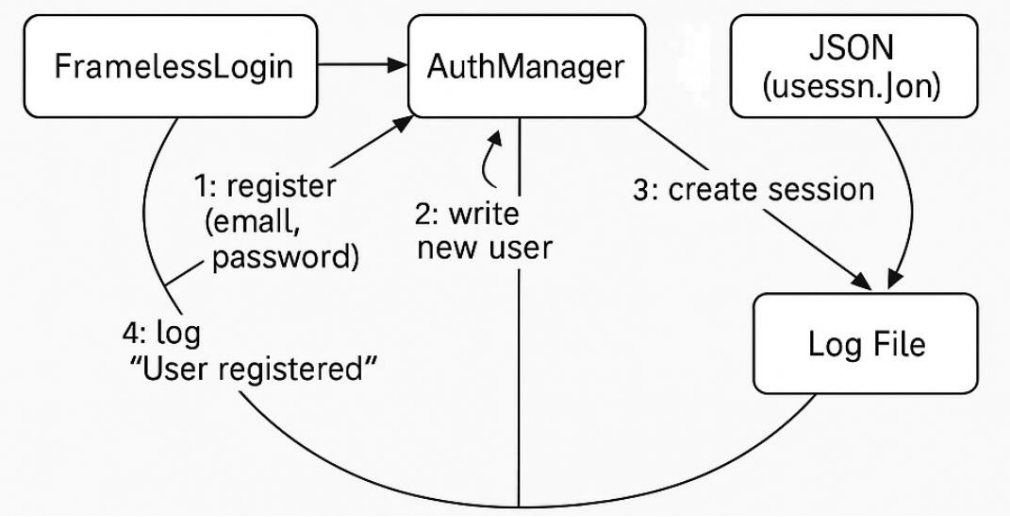

Рис. 2.4 . Сценарій реєстрації нового користувача та збереження сесії

Відповідно до діаграми, після ініціації процесу користувачем, модуль AuthManager створює новий запис у файлі users.json, генерує відповідну сесію у session.json та ініціює логування операції в log.txt. Це забезпечує реалізацію повного життєвого циклу події «реєстрація користувача» з фіксацією її на трьох логічних рівнях: дані, сесія, журнал подій. Такий підхід дозволяє забезпечити як функціональність, так і простежуваність операцій з боку адміністратора системи [6; 20].

Сумарна архітектурна модель, представлені класи та їх взаємодії повністю відповідають вимогам системи щодо розмежування обов’язків, підтримки низького рівня залежностей, забезпечення повторного використання коду та ефективної обробки даних. Така структура дозволяє масштабування як у бік розширення функціоналу (наприклад, додавання модулів сповіщень або архівування), так і в бік інтеграції нових джерел зберігання. Використання діаграм класів та кооперації забезпечує формалізоване представлення архітектурної моделі, що є критично важливим при реалізації об'єктно-орієнтованих систем [17; 19; 26].

2.3 Діаграма пакетів системи

У процесі проєктування архітектури програмного забезпечення важливим етапом є формалізація модульної структури системи за допомогою діаграми пакетів. Вона дозволяє відобразити логічну організацію вихідного коду, розмежування відповідальності між пакетами (директоріями, модулями) та залежності між ними, що є критичним при масштабуванні та підтримці програмного продукту [16; 17]. У даному випадку було побудовано UML-діаграму пакетів, яка відображає загальну структуру системи, зокрема поділ на підсистеми ui, logic, data, utils та external. Ієрархічна декомпозиція компонентів системи подана на рис. 2.5.

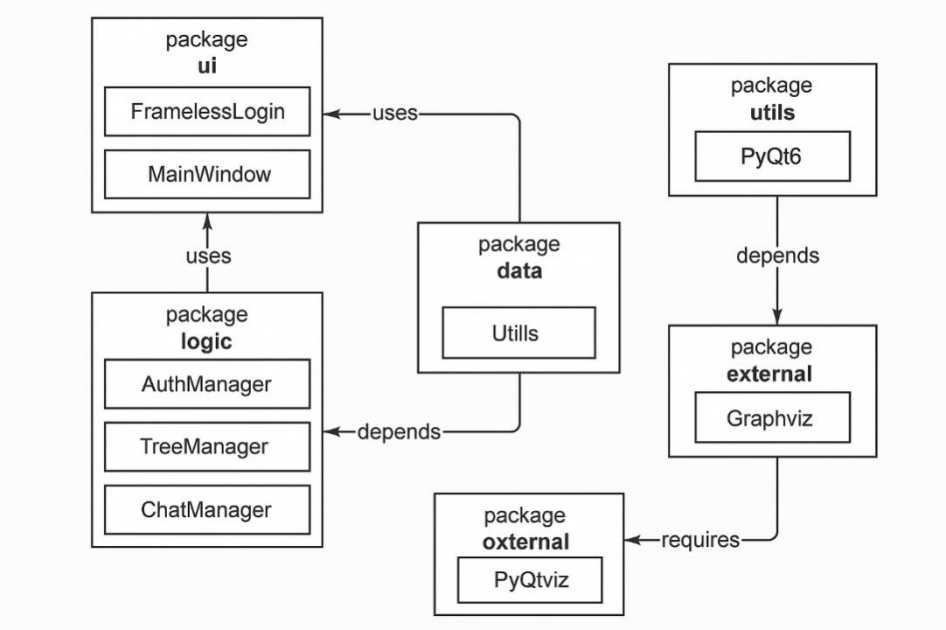

Рис. 2.5. Діаграма пакетів архітектури програмної системи

Пакет ui (user interface) об’єднує компоненти FramelessLogin та MainWindow, які відповідають за побудову графічного інтерфейсу користувача та взаємодію з основними сервісами системи. Цей пакет використовує (uses) функціонал бізнес-логіки, інкапсульований у пакеті logic, зокрема методи керування автентифікацією, побудовою дерева та обробкою повідомлень. Пакет logic містить три ключові модулі — AuthManager, TreeManager та ChatManager, які реалізують основну функціональність програмної системи та використовують службові методи з пакета data.

Пакет data забезпечує доступ до файлової системи, зокрема зчитування й збереження даних у форматі JSON (users.json, session.json, messages.json), а також реалізує допоміжні сервіси, такі як логування чи резервне копіювання. logic залежить від data у межах операцій читання/запису сесій, повідомлень, користувачів тощо. Пакет data також доступний для ui, але лише опосередковано, через виклики до логіки, що дозволяє реалізувати принцип інверсії залежностей.

Пакет utils містить універсальні інструменти, що не належать до конкретної бізнес-логіки, але є необхідними для підтримки функціонування системи, зокрема роботу з PyQt6, валідацією форм, або генерацією інтерфейсних подій. Залежність utils від external указує на те, що ці утиліти побудовані на

основі сторонніх бібліотек. external у свою чергу є формальним пакетом-обгорткою, що містить зовнішні залежності — такі як Graphviz (для візуалізації родинного дерева) та PyQtviz (графічна інкапсуляція бібліотек в інтерфейсі). Компонент TreeManager з пакета logic вимагає (requires) присутність Graphviz, що відображено у вигляді прямої залежності з підписом requires.

У системі реалізовано мінімальний рівень зв’язності між пакетами, що відповідає принципам модульності, повторного використання коду та інкапсуляції [1; 18]. Така структура забезпечує:

- розділення відповідальностей (separation of concerns);
- чітку ієрархію залежностей (UI → Logic → Data);
- спрощення процесу тестування кожного модуля окремо;
- зручність для масштабування системи (додавання нових пакетів із розширеним функціоналом).

Загальний перелік пакетів та їх функціональне призначення представлено у табл. 2.2.

Таблиця 2.2

Пакети системи та їх призначення

№	Назва пакета	Вміст	Функціональне призначення
1	ui	FramelessLogin, MainWindow	Побудова графічного інтерфейсу користувача
2	logic	AuthManager, TreeManager, ChatManager	Реалізація бізнес-логіки: автентифікація, обробка дерев, повідомлень
3	data	Utills, JSON-файли	Робота з файлами, сесіями, повідомленнями, логами
4	utils	PyQt6	Допоміжні утиліти та компоненти GUI
5	external	Graphviz, PyQtviz	Зовнішні залежності, необхідні для візуалізації та інтерфейсних фреймворків

Діаграма пакетів дозволяє ефективно структурувати програмний продукт, відобразити ключові залежності між модулями, зменшити зв’язаність та підвищити масштабованість системи. Вона є основою для побудови

компонентної, сервісної або мікросервісної архітектури у майбутніх версіях розробки [3; 14; 27].

2.4 Діаграма компонентів

Для формалізації архітектурної структури програмної системи було побудовано діаграму компонентів, яка відображає логічний поділ на основні функціональні блоки, їх залежності, зовнішні інтерфейси та джерела даних. Дана діаграма дозволяє наочно уявити організацію модулів системи, характер взаємодії між ними та напрям обміну інформацією, що є критично важливим при проєктуванні масштабованих десктопних застосунків з модульною структурою [16, с. 141; 17].

Як показано на рис. 2.6, система складається з трьох основних логічних областей: інтерфейс користувача (UI), компоненти бізнес-логіки та зовнішні джерела даних. Окремо виділено компонент візуалізації Graphviz, що виконує функцію генерації графів для візуального представлення структури сімейного дерева. Усі компоненти системи позначено з використанням UML-стереотипу <<component>>, а файли з даними — як <<data source>>.

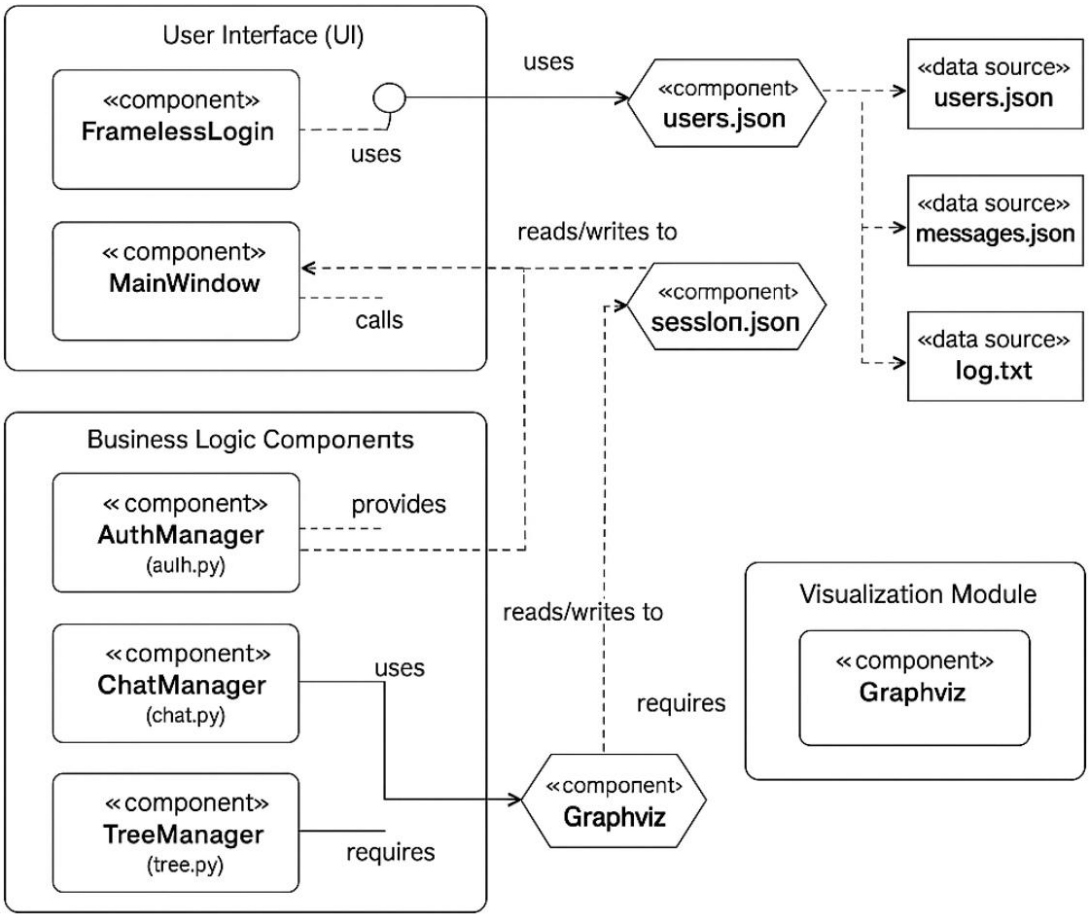

Рис. 2.6. Діаграма компонентів програмної системи

Компоненти інтерфейсу користувача FramelessLogin та MainWindow, реалізовані на основі PyQt6, відповідають за взаємодію з користувачем і викликають відповідні методи бізнес-логіки через асоціацію типу calls або uses. Зокрема, FramelessLogin використовує компонент AuthManager для автентифікації користувачів, а також безпосередньо звертається до джерела даних users.json для зчитування інформації. Водночас MainWindow оперує компонентами ChatManager та TreeManager, що відповідають за обробку повідомлень і побудову генеалогічного дерева відповідно.

У межах компонента бізнес-логіки AuthManager реалізовано функції реєстрації, логіну, створення сесій і логування активностей. Цей компонент забезпечує зв’язок із файлами session.json (створення/зчитування сесій) та log.txt (журналювання дій), які, у свою чергу, доступні й для інших сервісів. ChatManager виконує операції збереження повідомлень у messages.json і не

взаємодіє безпосередньо з інтерфейсом візуалізації. Компонент TreeManager забезпечує формування структури сімейного дерева та безпосередньо використовує модуль Graphviz для генерації його візуального представлення. Як видно з діаграми, для компонента Graphviz встановлено відношення requires, оскільки для функціонування TreeManager його наявність є критичною.

Особливістю реалізованої компонентної структури є відокремлення зовнішніх залежностей (наприклад, бібліотека Graphviz) від внутрішніх модулів логіки, що дозволяє легко змінювати або масштабувати функціонал без втручання у ядро системи. Таке розмежування також сприяє покращенню супроводжуваності та тестованості коду відповідно до принципів інкапсуляції та модульності [1; 19, с. 77].

Систематизований опис функцій кожного компонента наведено у табл. 2.3, яка демонструє функціональні ролі елементів діаграми та характер взаємодії між ними.

Таблиця 2.3

Компоненти системи та їх функціональні призначення

№	Назва компонента	Призначення та взаємодія
1	<<component>> FramelessLogin	Вікно входу/реєстрації, використовує AuthManager, працює з users.json
2	<<component>> MainWindow	Головне вікно системи, викликає ChatManager, TreeManager, працює з Graphviz
3	<<component>> AuthManager	Логіка авторизації, взаємодіє з users.json, session.json, log.txt
4	<<component>> ChatManager	Обробка повідомлень, запис у messages.json
5	<<component>> TreeManager	Побудова дерева, взаємодія з Graphviz
6	<<component>> Graphviz	Зовнішній компонент візуалізації, потрібен для рендерингу дерева
7	<<data source>> users.json	Зберігання облікових записів користувачів

Продовження таблиці 2.3

8	<<data source>> messages.json	Архів повідомлень

9	<<data source>> session.json	Дані про поточну сесію користувача
10	<<data source>> log.txt	Журнал дій користувача

Запропонована компонентна модель повністю відповідає концепції слабкозв’язаної архітектури та дозволяє реалізувати систему з високим ступенем автономності, розширюваності та ізоляції обов’язків між модулями. Вона також забезпечує чітке розмежування між UI-компонентами, логікою та збереженням даних, що спрощує тестування і підтримку системи на всіх етапах життєвого циклу [14; 18; 26].

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

У контексті розробки десктопного програмного забезпечення для побудови родинних дерев із функціоналом соціальної взаємодії обґрунтованим є вибір локальної системи управління базами даних, що забезпечує автономність, конфіденційність даних і високу швидкодію. У якості такої СУБД було обрано SQLite, що є вбудованою реляційною СУБД з відкритим кодом і не потребує розгортання окремого серверного середовища/

SQLite є придатною для застосування в персональних десктопних програмах завдяки таким характеристикам:

– зберігання всіх таблиць і індексів у єдиному локальному файлі;
– підтримка SQL-92 з розширеннями;
– наявність транзакційності та забезпечення цілісності даних;
– сумісність із Python через стандартну бібліотеку sqlite3.

Завдяки цим властивостям SQLite дає змогу уникнути складнощів, пов’язаних з розгортанням і підтримкою повноцінної серверної СУБД, водночас зберігаючи структурованість даних і забезпечуючи високий рівень контролю над доступом.

У таблиці 3.1 представлено основні характеристики використаної СУБД.

Таблиця 3.1

Характеристики обраної СУБД SQLite

№	Параметр	Значення
1	Тип	Вбудована реляційна СУБД
2	Формат зберігання	Один файл бази даних
3	Механізм доступу	Через бібліотеку sqlite3 (Python)
4	Підтримка транзакцій	Так (ACID-сумісність)
5	Сумісність із ОС	Windows, Linux, macOS

Продовження таблиці 3.1

6	Забезпечення безпеки	Через локальне зберігання, без мережевого доступу
7	Можливість масштабування	Локальна – обмежена; оптимальна для невеликих систем
8	Призначення	Персональні застосунки, мобільні системи

Оскільки система також підтримує простіший обмін даними, локальне зберігання повідомлень, сесій та профілів користувачів у форматі JSON, в окремих випадках використовується структура ключ-значення, що зберігається у вигляді масивів об'єктів. Такий підхід доцільний для зберігання неструктурованих або допоміжних даних, які не потребують складної логіки запитів. Нижче наведено приклад вмісту файлу users.json, який слугує джерелом даних для модуля автентифікації, як показано на рис.3.1.

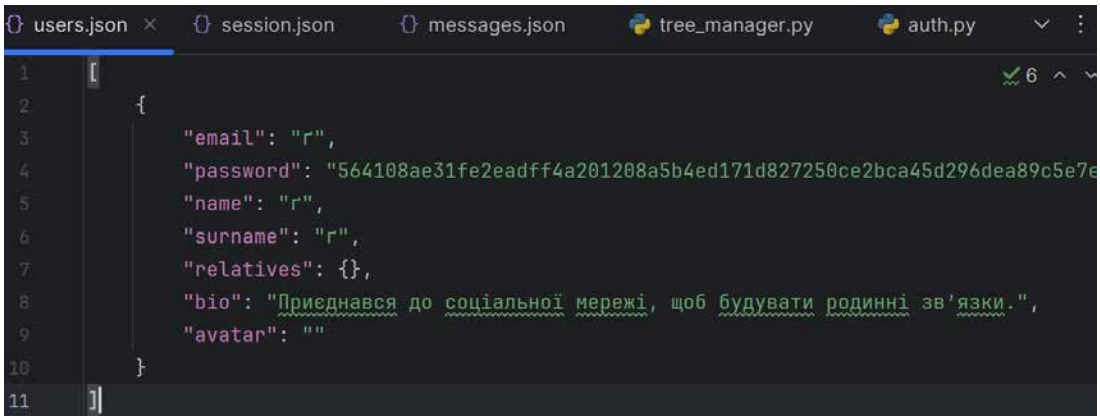

Рис.3.1. Запис інформації у файл users.json

Поєднання SQLite для структурованих таблиць (користувачі, повідомлення, родинні зв’язки) та JSON для тимчасових або допоміжних даних (сесії, конфігурації дерева) дає змогу реалізувати гібридну систему зберігання, яка забезпечує баланс між продуктивністю та гнучкістю.

Застосування вбудованої СУБД забезпечує відповідність вимогам до офлайн-роботи, приватності користувача та сумісності з архітектурою PyQt6-додатка, а використання JSON-файлів доповнює функціонал зберігання в частині неструктурованих даних та логів подій.

3.2 Розробка інформаційної бази

Процес проєктування інформаційної бази є ключовим етапом розробки програмної системи, оскільки забезпечує структуроване зберігання, цілісність та логічну доступність усіх даних користувача. На підставі логічної моделі, сформованої у попередньому підрозділі, було реалізовано фізичну модель бази даних, яка адаптована до використання локальної СУБД SQLite із додатковими джерелами даних у форматі JSON для тимчасових або сесійних записів.

Розроблена фізична модель реалізує класичну реляційну структуру з чітко визначеними первинними (PK) та зовнішніми (FK) ключами, що забезпечує підтримку основних типів зв'язків між сутностями: один до багатьох, багато до багатьох та один до одного. Крім того, передбачено структури для соціальної взаємодії, збереження повідомлень, профілів користувачів, родинних зв'язків та конфігурацій дерева.

На рисунку 3.2 наведено фізичну модель даних у вигляді діаграми у нотації Crow's Foot, яка відображає основні таблиці, типи полів, ключі та зв'язки між сутностями.

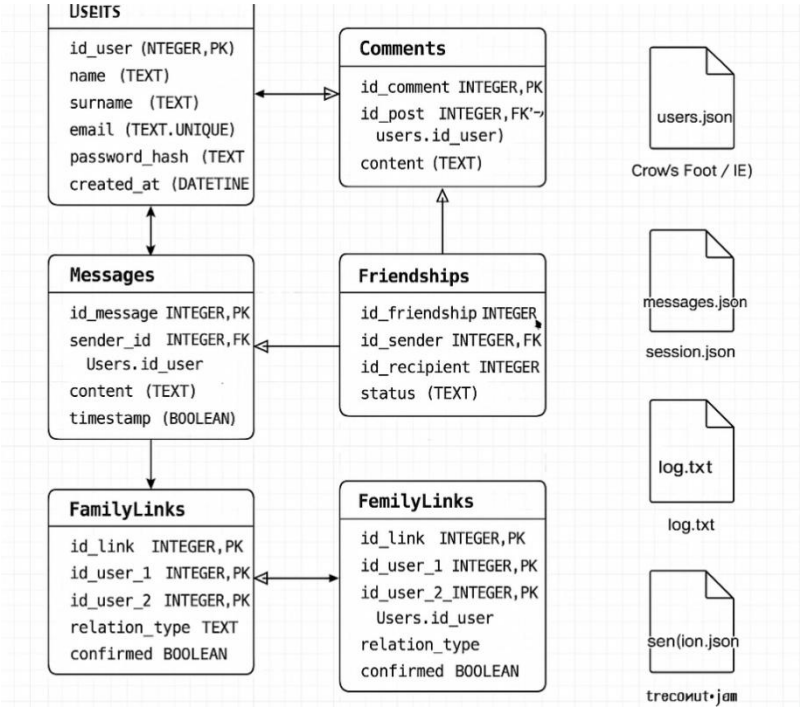

Рис. 3.2. Фізична модель даних інформаційної бази

У структурі бази даних реалізовано такі основні таблиці:

– Users – містить дані облікових записів: ідентифікатор користувача, ім’я, електронну пошту, геш пароля та дату створення. У полі email встановлено обмеження унікальності.

– Posts і Comments – реалізують механізми створення публікацій і коментування, зі збереженням зв’язків із користувачем-автором.

– Messages – таблиця для обміну приватними повідомленнями з вказанням відправника, одержувача та часу надсилання.

– Friendships – таблиця для реалізації двосторонніх соціальних зв’язків з можливістю вказання статусу (запрошення, підтвердження тощо).

– FamilyLinks – зберігає родинні зв’язки між двома користувачами із зазначенням типу спорідненості та підтвердження.

– MiniTreeConfig – таблиця з індивідуальними налаштуваннями відображення генеалогічного дерева: назва, кількість поколінь, дата оновлення.

Окремим елементом інформаційної бази виступає підсистема локального зберігання JSON-файлів, які використовуються для зберігання даних сесій, повідомлень, профілів користувачів та логів. Такий підхід забезпечує швидкий доступ до допоміжної інформації без завантаження основної бази, а також дозволяє створити резервні копії.

На рисунку 3.3 наведено приклад файлової структури проєкту, де розміщено основні джерела даних у форматі .json.

Рис. 3.3. Локальне розміщення JSON-файлів у структурі проєкту

Поєднання реляційного підходу (SQLite) з гнучкістю зберігання у JSON дозволяє досягти ефективної моделі, яка задовольняє вимоги автономної десктопної системи. Така гібридна інформаційна база поєднує переваги формалізованої структури SQL із простотою зберігання та обробки допоміжних даних у JSON-форматі, що повністю відповідає вимогам до локальних програмних систем

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення розробленої системи реалізована відповідно до принципів багаторівневої моделі та об’єктно-орієнтованого підходу. Основою побудови структури є патерн MVC (Model–View–Controller), який забезпечує чітке розмежування функцій між компонентами, підвищує масштабованість системи та спрощує супровід у процесі її експлуатації. Структура передбачає поділ на три логічні рівні: рівень представлення (Presentation Layer), рівень логіки (Business Logic Layer) та рівень доступу до даних (Data Access Layer), що відповідає загальноприйнятим практикам інженерії програмного забезпечення для десктопних застосунків [2, с. 86].

Графічне подання архітектурної моделі наведено на рисунку 3.4. У цій схемі зображено основні компоненти системи, їх взаємозв’язки, залежності та напрями інформаційного обміну. Компоненти згруповані відповідно до функціональних рівнів, що забезпечує інкапсуляцію відповідальностей та логічну структуризацію програмного коду.

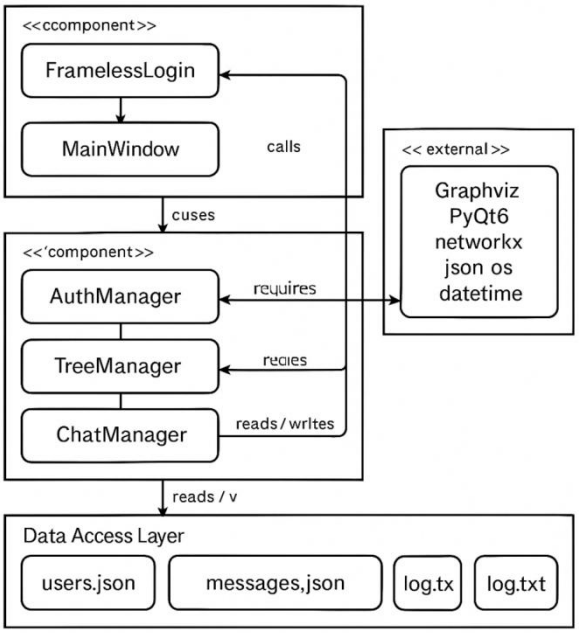

Рис. 3.5. Архітектура програмного забезпечення системи

Рівень представлення включає два основні інтерфейсні модулі – FramelessLogin та MainWindow, реалізовані засобами бібліотеки PyQt6. Перший забезпечує авторизацію та реєстрацію користувачів, другий – надає доступ до основного функціоналу: взаємодії з повідомленнями, побудови сімейного дерева, перегляду профілів. Цей рівень оперує лише інтерфейсними подіями, делегуючи обробку логіці.

На рівні бізнес-логіки реалізовано три модулі: AuthManager, TreeManager та ChatManager. Компонент AuthManager відповідає за валідацію користувацьких даних, створення сесій, зберігання паролів у хешованому вигляді. TreeManager обробляє структури родинного дерева, здійснює додавання, редагування, збереження та візуалізацію родинних зв’язків. Для рендерингу графів застосовуються зовнішні бібліотеки Graphviz і networkx, що безпосередньо підключені через залежності requires. Компонент ChatManager реалізує функціонал надсилання повідомлень, формування діалогів та запис історії спілкування у локальний файл.

Рівень доступу до даних побудований на основі локальної файлової системи із застосуванням форматів JSON і TXT. Файли users.json, messages.json, session.json, log.txt містять відповідно інформацію про користувачів,

повідомлення, активні сесії та дії в системі. Такий підхід відповідає вимогам автономності, дозволяє здійснювати читання та запис без залучення зовнішніх баз даних, забезпечуючи високу швидкодію та простоту резервування.

Усі компоненти системи реалізовані як слабо зв’язані модулі з чітко визначеними інтерфейсами взаємодії. Взаємозв’язки між модулями позначено стрілками з відповідними підписами: calls, uses, requires, reads/writes, що вказує на напрям і характер залежності. Відокремлення зовнішніх залежностей (Graphviz, PyQt6, datetime, json) в окремий блок <<external>> дозволяє централізовано контролювати всі сторонні ресурси та спростити їх оновлення або заміну.

Загальна структура системи відповідає принципам слабкої зв’язаності, повторного використання коду, модульності та тестованості, що забезпечує гнучкість у разі розширення функціоналу. Деталізований опис компонентів, їх призначення та належність до відповідних рівнів наведено у таблиці 3.2.

Таблиця 3.2

Компоненти архітектури програмного забезпечення

№	Компонент	Рівень	Призначення
1	FramelessLogin	Presentation Layer	Вікно реєстрації та авторизації
2	MainWindow	Presentation Layer	Головний інтерфейс для доступу до функцій системи
3	AuthManager	Business Logic Layer	Автентифікація, управління сесіями, створення облікових записів
4	TreeManager	Business Logic Layer	Побудова, редагування, візуалізація родинного дерева
5	ChatManager	Business Logic Layer	Обробка повідомлень між користувачами
6	users.json	Data Access Layer	Зберігання профілів користувачів
7	messages.json	Data Access Layer	Збереження історії спілкування
8	session.json	Data Access Layer	Фіксація поточної активної сесії
9	log.txt	Data Access Layer	Логування дій у системі
10	Graphviz, PyQt6	External Dependencies	Зовнішні бібліотеки для графів, інтерфейсу та обробки системних подій

Запропонована архітектура є формальною, структурованою та орієнтованою на розширення. Вона дозволяє ефективно поєднати візуальні, логічні та інформаційні компоненти, забезпечуючи надійне функціонування програмної системи в умовах локального середовища без зовнішніх серверів, що цілком відповідає поставленим технічним вимогам.

3.4 Вибір інструментарію для створення прикладного програмного забезпечення

Для реалізації прикладного програмного забезпечення, яке поєднує функціонал соціальної взаємодії та побудови родинного дерева, було обрано інструментарій, орієнтований на ефективну роботу в десктопному середовищі, підтримку візуалізації складних структур і забезпечення високої продуктивності при локальній обробці даних. Основними критеріями відбору інструментальних засобів стали: кросплатформеність, інтегрованість із мовою Python, підтримка графічних інтерфейсів, робота з графами та гнучкість у зберіганні даних.

Базовою мовою програмування було обрано Python, що є сучасним інструментом для швидкої розробки, має розвинену екосистему бібліотек і добре пристосований до реалізації об’єктно-орієнтованої архітектури. Використання Python дає змогу інтегрувати функціональні компоненти без надмірного ускладнення проєкту, а також забезпечити швидке тестування, налагодження й масштабування системи.

Для реалізації графічного інтерфейсу користувача застосовано PyQt6 — обгортку над фреймворком Qt, що підтримує розробку адаптивних GUI-компонентів, роботу з сигналами/слотами, створення вікон і форм із гнучким налаштуванням. Це забезпечує відповідність системи сучасним вимогам до зручності взаємодії з користувачем.

Побудову родинного дерева реалізовано за допомогою бібліотек Graphviz та networkx. Graphviz відповідає за генерацію графів та їх графічне

представлення (DOT-файли, візуалізація), тоді як networkx дає змогу формувати структуру дерева як графову модель з можливістю програмного аналізу та трансформацій. Разом ці засоби створюють основу для візуалізації динамічних, багаторівневих родинних зв’язків.

У якості бази даних використано формат JSON для локального зберігання профілів, повідомлень і конфігурацій. Такий підхід дозволяє уникнути надлишкової складності СУБД, зберігаючи при цьому цілісність і доступність даних для обробки через стандартну бібліотеку json. У перспективі також можлива інтеграція з SQLite, що забезпечує реляційне зберігання з підтримкою SQL-запитів.

Підсумковий склад обраного інструментарію наведено в таблиці 3.3.

Таблиця 3.3

Інструменти, використані для розробки програмного забезпечення

№	Засіб / бібліотека	Призначення
1	Python 3.10	Основна мова розробки
2	PyQt6	Створення графічного інтерфейсу користувача
3	Graphviz	Генерація та візуалізація дерева у вигляді графу
4	networkx	Формування і логічна побудова графової структури родинних зв’язків
5	json (стандартна)	Зберігання даних у локальних файлах
6	os, datetime	Обробка системних подій, роботи з файлами, часовими мітками
7	SQLite (опційно)	Реляційне зберігання інформації у разі масштабування

Застосування вказаних засобів дозволило реалізувати архітектурно незалежне рішення з високим ступенем автономності, зручним інтерфейсом, гнучкою логікою побудови родинних структур і можливістю подальшого масштабування без фундаментальної зміни архітектури. Вибір кожного елементу інструментарію був зумовлений відповідністю функціональним та нефункціональним вимогам системи, що цілком узгоджується з сучасними підходами до проєктування прикладного ПЗ для персонального використання.

3.5 Алгоритми обробки даних та побудови родинних структур

Алгоритмічна складова розробленої програмної системи відіграє ключову роль у забезпеченні її функціональної цілісності та коректної обробки генеалогічної інформації. Побудова родинних структур, збереження зв’язків між учасниками системи, візуалізація деревоподібних графів, а також забезпечення безпеки даних потребують чіткого визначення алгоритмічних процедур. Система оперує складними множинами персональних, соціальних та родинних даних, що вимагає побудови внутрішньої графової моделі із підтримкою ациклічності, рівнів вкладеності та інтеграцією декількох типів споріднених зв’язків.

Загальна логіка обробки родинних зв’язків базується на комбінованому застосуванні класичних методів обходу графів при формуванні дерева, а також алгоритмів перевірки консистентності зв’язків з метою забезпечення відсутності логічних суперечностей при модифікації бази родинних зв’язків. Особлива увага приділяється запобіганню появі циклічних залежностей, які можуть призводити до логічних помилок при візуалізації генеалогічної структури.

Вхідні дані для побудови дерева оброблюються у форматі реляційних таблиць та перетворюються у внутрішню графову модель з наступним формуванням орієнтованого ациклічного графа, що забезпечує коректне визначення батьківсько-нащадкових відносин. Основні алгоритмічні етапи перетворення даних подано у таблиці 3.4.

Таблиця 3.4

Етапи формування родинної структури

№	Етап обробки	Опис операцій
1	Завантаження вихідних даних	Зчитування таблиць користувачів та зв’язків з бази даних
2	Побудова графової моделі	Формування списку суміжності на основі зв’язків FamilyLinks
3	Виявлення кореневих вузлів	Ідентифікація осіб без батьків у графі

4	Обхід графа	Визначення рівнів вузлів методом ширинного обходу

Продовження таблиці 3.4

5	Формування графа для візуалізації	Перетворення у формат DOT для побудови дерева
6	Генерація графічної структури	Створення візуального дерева за допомогою Graphviz

Алгоритмічна складова системи також охоплює процедури безпечного збереження та модифікації інформації при кожній взаємодії користувача із програмною системою. При додаванні нових родинних зв’язків, оновленні профілів або редагуванні соціальних взаємодій реалізовано механізми валідації даних, перевірки цілісності та транзакційного запису змін. Критичні операції додавання спорідненості супроводжуються перевіркою відсутності непрямих циклів у родинній структурі з метою забезпечення її логічної послідовності.

Особливої уваги потребують алгоритми управління безпекою та доступом до даних, оскільки система оперує чутливою персональною інформацією. Розмежування прав доступу реалізується через аутентифікацію користувачів, їхню ідентифікацію при виконанні операцій та захист критичних функцій адміністративного рівня. Основні алгоритмічні процедури захисту представлено в таблиці 3.5.

Таблиця 3.5

Алгоритмічні процедури безпеки даних

№	Категорія контролю	Алгоритмічні механізми реалізації
1	Аутентифікація	Перевірка облікових записів із гешуванням паролів
2	Розмежування прав доступу	Перевірка ролей користувачів перед виконанням операцій
3	Контроль цілісності операцій	Валідація введених даних при збереженні змін
4	Логування дій	Журналювання усіх змін з фіксацією користувача та часу
5	Захист від несанкціонованого редагування	Підтвердження адміністративних операцій через GUI

Додатковим важливим компонентом алгоритмічного забезпечення є модуль створення резервних копій, що дозволяє зберігати актуальні версії даних перед виконанням критичних модифікацій у системі. Алгоритм резервування базується на створенні копій основних файлів бази даних та інформаційних файлів системи із часовими мітками збереження. Завдяки цьому користувач отримує можливість відновлення попереднього стану системи у разі виникнення помилок або аварійних ситуацій.

Журналювання усіх операцій у системі дозволяє створювати повну історію змін, яка є інструментом контролю діяльності системи з боку адміністратора. Формат журналу включає детальну інформацію про тип виконаної операції, ідентифікатор користувача, дату та час виконання дії, а також статус операції.

Комплекс алгоритмів обробки даних, візуалізації деревоподібних структур, управління доступом та контролю стабільності функціонування формує єдину функціонально-збалансовану архітектуру програмної системи. Узагальнені функціональні алгоритмічні блоки системи наведено у таблиці 3.6.

Таблиця 3.6

Функціональні алгоритмічні блоки системи

№	Блок системи	Алгоритмічні функції
1	Обробка родинних зв’язків	Перевірка цілісності зв’язків, запобігання циклам
2	Побудова дерева	Генерація графа спорідненості, візуалізація Graphviz
3	Соціальна взаємодія	Збереження повідомлень, обробка публікацій
4	Управління профілем	Створення, редагування облікових записів
5	Резервне копіювання	Архівування даних перед змінами
6	Аутентифікація та безпека	Авторизація, логування, захист доступу
7	Контроль стабільності	Обробка помилок, забезпечення цілісності операцій

Розроблені алгоритмічні моделі забезпечують ефективне функціонування системи у різних експлуатаційних режимах, дозволяють підтримувати високу стабільність обробки даних, забезпечують адаптивність програми до розширення функціоналу та зберігають логічну цілісність родинних структур за умов зростання складності дерева.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ І ЕКСПЛУАТАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

4.1 Тестування системи

Процес тестування розробленого прикладного програмного забезпечення проводився з метою верифікації відповідності функціоналу сформульованим вимогам, перевірки коректності відображення інтерфейсу та забезпечення надійності взаємодії між компонентами. Особлива увага приділялася перевірці відображення вмісту форм, стабільності запуску, правильності стилів та відповідності елементів користувацького інтерфейсу заявленим у специфікації.

У рамках візуального та інтеграційного тестування було сформовано перелік основних сценаріїв взаємодії з системою, які охоплюють ключові інтерфейсні вікна. Результати валідації представлено у таблиці 4.1.

Таблиця 4.1

Заплановані сценарії тестування інтерфейсних компонентів

№	Назва компонента	Опис тесту	Очікуваний результат	Статус
1	MainWindow	Перевірка вкладок навігації, розмітки головного вікна	Вкладки відображаються, елементи розміщено коректно	✔ Пройдено
2	ChatWindow	Відображення списку діалогів, листування, поле введення	Список користувачів, повідомлення, форма введення працюють	✔ Пройдено
3	TreeVisualization	Перевірка коректності структури сімейного дерева	Зв’язки між елементами побудовано правильно	✔ Пройдено
4	ProfileView	Відображення персональних даних та біографії	Дані вирівняні, текст читабельний, кнопка «Редагувати» працює	✔ Пройдено
5	FramelessLogin	Тест запуску та розташування полів авторизації	Поля вирівняні, кнопки активні	✔ Пройдено

Продовження таблиці 4.1

6	AddRelativeDialog	Модальне вікно, валідація полів та стилізація	Діалог з’являється, елементи активні	✔ Пройдено

Форма MainWindow протестована на предмет відповідності дизайну структурам, зображеним у прототипі. На рисунку 4.1 наведено зовнішній вигляд головного вікна програми, що включає навігаційні вкладки, ліву панель зі списком родичів та праву панель з відображенням дерева.

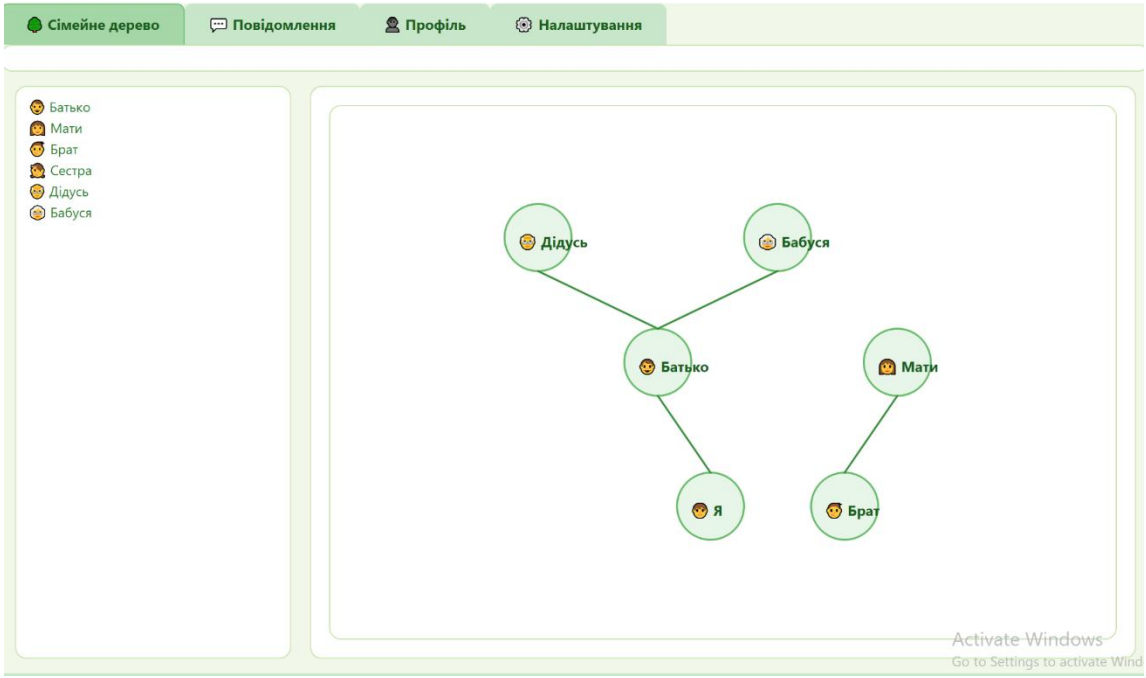

Рис. 4.1. Головне вікно користувача

Окремо було протестовано інтерфейс повідомлень. На рисунку 4.2 представлено макет форми ChatWindow, яка включає список користувачів, зону активного чату та поле введення повідомлення.

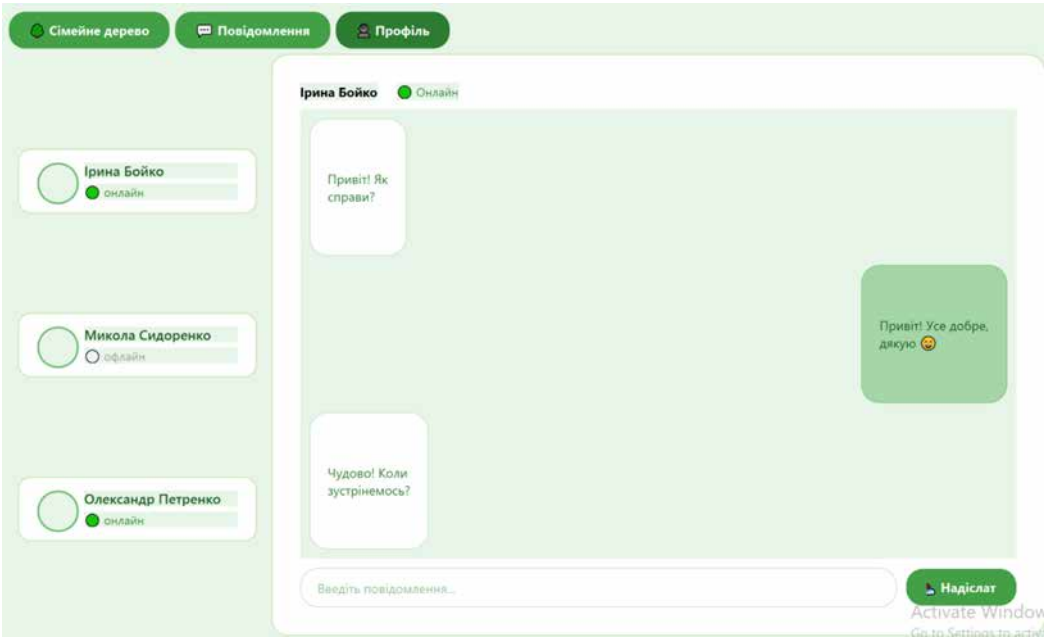

Рис. 4.2. Інтерфейс обміну повідомленнями

Сценарій перегляду персонального профілю перевірено на повноту та правильність відображення всіх структурованих даних. Як видно з рисунка 4.3, усі поля вирівняні, шрифт відповідає вимогам до читабельності, а біографія відображається у форматованому блоці.

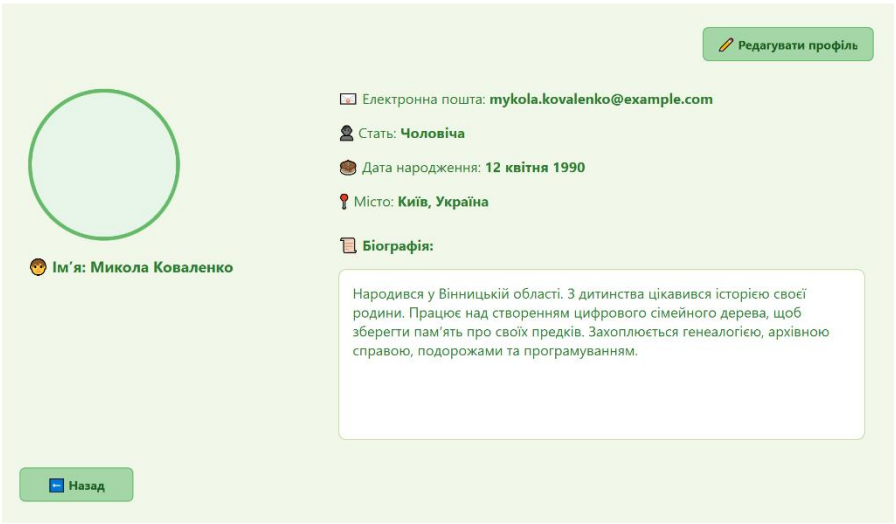

Рис. 4.3. Вікно перегляду профілю

Загальна оцінка результатів тестування показала, що система візуально коректна, усі інтерфейсні елементи відображаються відповідно до технічного завдання, верстка адаптивна, помилок візуалізації або блокувань взаємодії не виявлено. Таким чином, інтерфейсний рівень ПЗ відповідає заявленим функціональним і нефункціональним вимогам

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної та коректної роботи розробленої системи побудови родинного дерева з інтерактивним графічним інтерфейсом та підтримкою локального зберігання даних, визначено мінімальні та рекомендовані вимоги до апаратного та програмного середовища. Основний фокус реалізації системи — десктопне використання у локальному середовищі без залежності від зовнішніх серверів. Відповідно, обрана конфігурація має забезпечувати ефективне функціонування всіх компонентів застосунку на персональному комп’ютері користувача [2, с. 94].

У таблиці 4.2 представлено основні апаратні характеристики, рекомендовані для комфортного використання програмного забезпечення.

Таблиця 4.2

Апаратні вимоги до запуску програмного забезпечення

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2-ядерний, 1.6 ГГц	4-ядерний, 2.5 ГГц або вище
Оперативна пам’ять (RAM)	4 ГБ	8 ГБ або більше
Відеокарта	Інтегрована графіка	Дискретна з підтримкою OpenGL
Вільне місце на диску	200 МБ	500 МБ+ для резервних копій
Дисплей	1366×768	Full HD (1920×1080)
Периферія	Клавіатура, миша	Клавіатура, миша

З погляду програмного забезпечення система не потребує встановлення серверних середовищ, однак передбачає наявність відповідного виконуваного середовища інтерпретатора Python та бібліотек, які застосовуються в системі. Повний перелік програмних компонентів подано у таблиці 4.3.

Таблиця 4.3

Програмне забезпечення, необхідне для роботи системи

Компонент	Призначення	Версія
Python	Основна мова програмування	≥ 3.10
PyQt6	Створення графічного інтерфейсу користувача	Остання стабільна
Graphviz	Візуалізація родинного дерева	≥ 2.44

На рисунку 4.4 представлено діаграму розгортання системи, яка демонструє фізичне розміщення програмних модулів на пристрої користувача, взаємозв’язки з зовнішніми бібліотеками та (опціонально) з хмарним сховищем резервних копій.

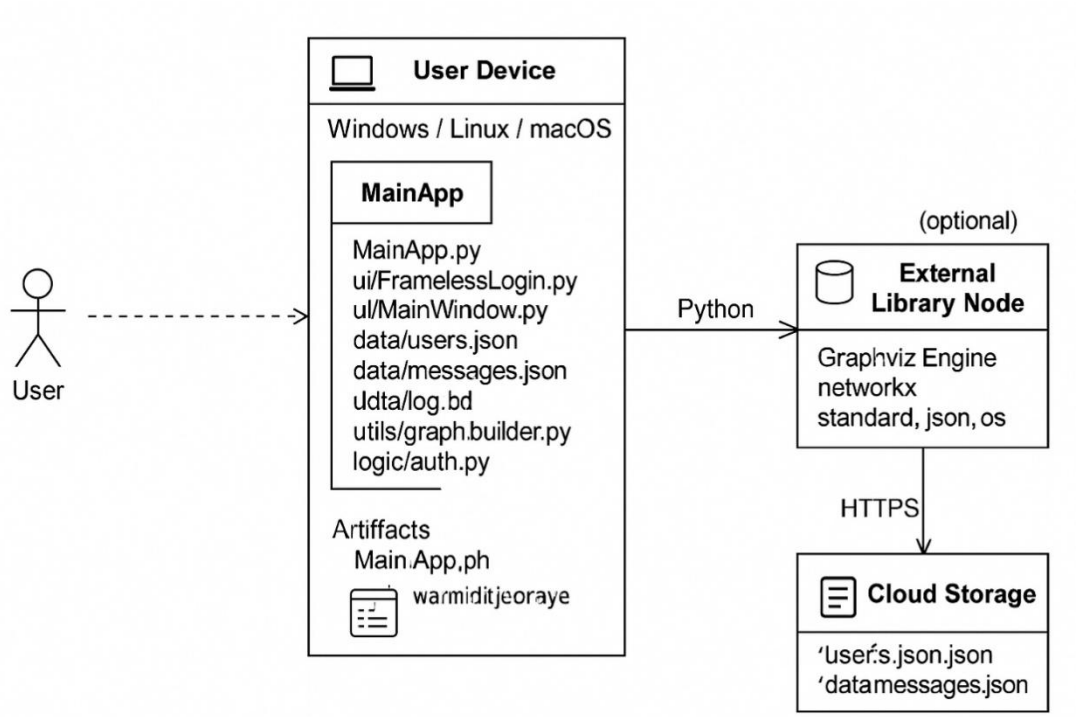

Рис. 4.4. Діаграма розгортання програмної системи

Система повністю підтримує офлайн-режим і не потребує підключення до серверної бази даних. Усі дані користувачів зберігаються у форматі .json у локальній файловій структурі, що гарантує високий рівень конфіденційності. Додатково реалізовано можливість інтеграції з віддаленим сховищем через HTTPS-з’єднання для автоматичного резервного копіювання, яке активується за бажанням користувача.

Таким чином, сформовані апаратні та програмні вимоги демонструють, що система є легко масштабованою, не потребує високопродуктивних обчислювальних ресурсів та може бути ефективно розгорнута на більшості персональних комп’ютерів із типовою конфігурацією.

4.3 Склад інсталяційного пакету

Інсталяційний пакет розробленої програмної системи формується з урахуванням забезпечення її повноцінного функціонування у цільовому середовищі кінцевого користувача, мінімізації зовнішніх залежностей та забезпечення стабільності роботи без додаткового налаштування компонентів. Оскільки програмний продукт реалізований як десктопна автономна система, до складу інсталяційного пакету включено всі необхідні компоненти для її повної працездатності без залучення зовнішніх серверних сервісів чи попередньо встановлених модулів програмної платформи.

Основною задачею формування інсталяційного пакету є забезпечення можливості запуску системи в ізольованому середовищі незалежно від конфігурації комп'ютера користувача. Для досягнення цієї мети було використано механізми створення standalone-дистрибутивів за допомогою спеціалізованих інструментів пакування програм на Python (зокрема PyInstaller), що дозволяє включити у пакет повний стек необхідних залежностей.

Основні елементи, що включено до складу інсталяційного пакету, систематизовано у таблиці 4.4.

Таблиця 4.4

Склад інсталяційного пакету програмної системи

№	Компонент	Зміст пакету	Функціональне призначення
1	Основний виконуваний файл	Зібраний EXE-файл або аналогічний для конкретної ОС	Запуск основної програми, інтегрований код бізнес-логіки, GUI та роботи з БД
2	Каталог бібліотек середовища виконання	PyQt6, sqlite3, networkx, graphviz, json, інші сторонні бібліотеки	Забезпечення коректної роботи всіх функцій незалежно від системних бібліотек
3	Локальні бази даних	users.json, messages.json, session.json	Зберігання профілів користувачів, повідомлень, інформації про сесії
4	SQLite-файл	familytree.db або аналогічний	Збереження структурованих таблиць генеалогічної інформації

Продовження таблиці 4.4.

5	Компоненти Graphviz	Двійкові файли, бібліотеки, виконавчі модулі	Генерація візуалізації сімейного дерева та побудова графів
6	Конфігураційні файли	config.ini, settings.json тощо	Параметри системної ініціалізації, налаштування інтерфейсу
7	Лог-файли	log.txt	Ведення журналу дій користувачів та технічної діагностики
8	Резервні копії	backups/ (каталог резервних даних)	Архівація важливих даних у разі аварійних ситуацій
9	Супровідна документація	README.txt, UserManual.pdf	Інструкції зі встановлення, експлуатації та супроводу
10	Ліцензійна інформація	license.txt	Правила використання програмного забезпечення

Формування інсталяційного пакету із включенням усіх необхідних компонентів дозволяє мінімізувати ризики виникнення помилок при інсталяції, пов'язаних із відсутністю необхідних залежностей, що характерно для типових розгортань програм на мові Python у середовищі кінцевих користувачів.

Окрему увагу приділено включенню у пакет бібліотеки **Graphviz**, яка є критичним компонентом для реалізації функціоналу побудови та візуалізації сімейних дерев. У випадку відсутності Graphviz у системі користувача передбачено автоматичне встановлення відповідних двійкових компонентів разом із інсталяцією основної програми, що забезпечує стабільність виводу графічних структур без необхідності ручного налаштування.

Використання hybrid-схеми збереження даних (поєднання SQLite та JSON-файлів) забезпечує гнучкість архітектури інформаційної бази та адаптивність до різних сценаріїв експлуатації. Конфігураційні файли дозволяють змінювати параметри роботи системи без модифікації основного коду, що істотно спрощує супровід програмного продукту.

Підготовлений інсталяційний пакет охоплює повний набір компонентів, необхідних для коректної роботи системи, забезпечуючи універсальність інсталяції, простоту експлуатації та високу стабільність функціонування незалежно від цільової операційної системи.

ВИСНОВКИ

У межах кваліфікаційної роботи було комплексно вирішено всі поставлені завдання, спрямовані на розробку прикладного програмного забезпечення для побудови інтерактивного родинного дерева з елементами соціальної взаємодії. В результаті проведеного системного аналізу предметної області визначено ключові характеристики майбутньої системи, сформовано вимоги до її функціоналу та архітектури, що стало основою для подальшого проєктування.

На основі зібраних функціональних і нефункціональних вимог сформовано структуровану специфікацію, яка відображає реальні потреби користувачів, що прагнуть зберігати й візуалізувати родинні зв’язки без використання мережевих сервісів. Особливу увагу приділено простоті інтерфейсу, автономності системи та захищеності персональних даних.

Моделювання предметної області здійснено із використанням UML-діаграм, які охоплюють статичні та динамічні аспекти функціонування системи. Побудовано діаграми класів, компонентів, розгортання, кооперації, а також ER-діаграму логічної структури бази даних, що дозволило формалізувати архітектурні рішення та забезпечити цілісність проєктної моделі.

Огляд наявних інформаційних систем у сфері соціальних мереж і цифрової генеалогії дозволив виявити низку суттєвих обмежень таких сервісів, як MyHeritage, Geni, Ancestry. Зокрема, йдеться про відсутність офлайн-доступу, складну структуру інтерфейсу, обмежену персоналізацію. Це стало підставою для створення альтернативного рішення з локальним зберіганням, простим інтерфейсом і можливістю ручного налаштування структури дерева.

Для ефективного впровадження розробленої системи рекомендовано попередньо встановити Python 3.10+, бібліотеки PyQt6 та Graphviz, забезпечити відповідність апаратним вимогам, активувати механізм резервного копіювання, провести тестування в реальних умовах користувача, а також надати короткий посібник для кінцевих користувачів. Завдяки модульній архітектурі система

легко масштабується, що дозволяє у майбутньому додати онлайн-функціонал або інтеграцію з іншими платформами.

Відповідно до технічного завдання реалізовано логічну модель даних та побудовано проєктну документацію у вигляді UML-діаграм, що стали підґрунтям для переходу до реалізації. Система побудована на основі архітектурного шаблону Model–View–Controller (MVC), що забезпечує розділення логіки, інтерфейсу та даних.

Для реалізації застосунку обрано сучасний і ефективний стек технологій: Python як основну мову програмування, PyQt6 для побудови графічного інтерфейсу, бібліотеки Graphviz і networkx для візуалізації родинних дерев, а також формат JSON для зберігання користувацьких даних у локальному середовищі. Реалізовано модулі авторизації, побудови дерева, обміну повідомленнями, а також налаштування профілю, що забезпечує повний життєвий цикл роботи користувача із системою.

У ході тестування перевірено стабільність роботи основних компонентів, відповідність інтерфейсу заявленому дизайну, коректність збереження й обробки інформації. Отримані результати підтвердили надійність архітектурних рішень і функціональну відповідність системи поставленим вимогам.

Таким чином, виконано всі завдання дипломної роботи: від аналізу й моделювання до реалізації, тестування та підготовки рекомендацій з експлуатації. Розроблене програмне забезпечення може бути використане як основа для подальших досліджень у галузі персональних інформаційних систем, цифрової генеалогії, а також як практичний інструмент для користувачів, зацікавлених у побудові родинної історії у зручному й доступному форматі.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Валянський Є. Т. Програмний засіб моделювання... [Електронний ресурс] // DSpace ТНЕУ. – 2021. – Режим доступу: https://dspace.wunu.edu.ua/bitstream/316497/43245/1/Валянський%20ЄТ_БР%202021.pdf.WUNU DSpace

2. Інформаційна технологія надання рекомендацій з прийняття... [Електронний ресурс] // Репозиторій ВНТУ. – 2023. – Режим доступу: https://iq.vntu.edu.ua/repository/getfile.php/7955.pdf.iq.vntu.edu.ua

3. Томашевський О. М. Інформаційні технології та моделювання [Електронний ресурс] / О. М. Томашевський. – К.: НАУ, 2020. – 112 с. – Режим доступу: https://moodle.nati.org.ua/pluginfile.php/14680/mod_resource/content/1/tomashevski_o_m_ta_in_informaciini_tehnologii_ta_modelyuvan.pdf.moodle.nati.org.ua

4. Міністерство освіти і науки України. Донецький національний університет. Тип UML-діаграм як діаграми діяльності... [Електронний ресурс] // Журнал архітектури. – 2021. – Режим доступу: https://jarch.donnu.edu.ua/article/view/10493/10403.jarch.donnu.edu.ua

5. Creately. Family Tree UML [Електронний ресурс]. – Режим доступу: https://creately.com/diagram/example/ilsebqv81/family-tree-uml. Creately+1Creately+1

6. Creately. Social Network Diagram [Електронний ресурс]. – Режим доступу: https://creately.com/diagram/example/jos61nj46/social-network-diagram. Creately

7. Improved Graph Drawings of Family Trees and UML Class Diagrams [Електронний ресурс] // Repositorio Aberto UP. – 2022. – Режим доступу: https://repositorio-aberto.up.pt/bitstream/10216/144498/2/586964.pdf.repositorio-aberto.up.pt

8. Building and Analyzing Node-Link Diagrams to Understand Social Networks [Електронний ресурс] // Kansas State University. – 2012. – Режим доступу: https://idme-test.ome.ksu.edu/BuildingAnalyzingNodeLinkDiagramsSocialNetworks/BuildingAnalyzingNodeLinkDiagramsSocialNetworks_print.html.idme-test.ome.ksu.edu

9. Graphviz [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Graphviz.

10. PlantUML [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/PlantUML.

11. yEd - Graph Editor [Електронний ресурс] // yWorks. – Режим доступу: https://www.yworks.com/products/yed.yWorks, the diagramming experts+1Wikipedia+1

12. YEd [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/YEd.

13. Diagram [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Diagram.

14. 5 Types of Ties and Their Graphs - Social Networks [Електронний ресурс] // Networks Textbook. – Режим доступу: https://olizardo.github.io/networks-textbook/lesson-graphs-ties.html.olizardo.github.io

15. A network diagram ('digraph') representing the social relationships... [Електронний ресурс] // ResearchGate. – Режим доступу: https://www.researchgate.net/figure/A-network-diagram-digraph-representing-the-social-relationships-between-the-10_fig1_31060124.ResearchGate

16. Unified Modeling Language User Guide / Grady Booch, James Rumbaugh, Ivar Jacobson. – 2nd ed. – Addison-Wesley, 2005. – 496 p.

17. UML Distilled: A Brief Guide to the Standard Object Modeling Language / Martin Fowler. – 3rd ed. – Addison-Wesley, 2003. – 208 p.

18. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design / Craig Larman. – 3rd ed. – Prentice Hall, 2004. – 736 p.

19. The Elements of UML 2.0 Style / Scott W. Ambler. – Cambridge University Press, 2005. – 208 p.

20. Object-Oriented Modeling and Design with UML / Michael Blaha, James Rumbaugh. – 2nd ed. – Prentice Hall, 2004. – 496 p.

21. Coursera. Object-Oriented Analysis & Design [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/object-oriented-analysis.

22. edX. Software Construction: Object-Oriented Design [Електронний ресурс]. – Режим доступу: https://www.edx.org/course/software-construction-object-oriented-design.

23. Udemy. UML and Object-Oriented Design Foundations [Електронний ресурс]. – Режим доступу: https://www.udemy.com/course/uml-and-object-oriented-design-foundations/.

24. Pluralsight. UML Fundamentals [Електронний ресурс]. – Режим доступу: https://www.pluralsight.com/courses/uml-fundamentals.

25. LinkedIn Learning. Learning UML [Електронний ресурс]. – Режим доступу: https://www.linkedin.com/learning/learning-uml.

26. Creately. Class Diagram Templates [Електронний ресурс]. – Режим доступу: https://creately.com/diagram/example/ilsebqv81/family-tree-uml. Creately+1Creately+1

27. Graphviz. Drawing graphs with dot [Електронний ресурс]. – AT&T Bell Laboratories, 1991. – Режим доступу: https://graphviz.org.Wikipedia

28. PlantUML. Official website [Електронний ресурс]. – Режим доступу: http://plantuml.com.

29. yEd. Official website [Електронний ресурс]. – Режим доступу: https://www.yworks.com/yed.Wikipedia+1yWorks, the diagramming experts+1

30. Wikipedia. Diagram [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Diagram.

ДОДАТОК А

Фрагмент коду головного вікна графічного інтерфейсу користувача

```
import sysfrom PyQt6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QListWidget, QTabWidget, QStatusBar, QGraphicsScene, QGraphicsView,
    QGraphicsEllipseItem, QGraphicsTextItem, QGraphicsLineItem, QFrame
)from PyQt6.QtCore import Qt, QPointFfrom PyQt6.QtGui import QPen, QBrush,
QColor, QFont, QPainter
class FamilyTreeWidget(QGraphicsView):
    def __init__(self):
        super().__init__()
        self.scene = QGraphicsScene()
        self.setScene(self.scene)
        self.setRenderHint(QPainter.RenderHint.Antialiasing)
        self.draw_tree()

    def draw_person(self, name, x, y):
        radius = 70
        ellipse = QGraphicsEllipseItem(x, y, radius, radius)
        ellipse.setBrush(QBrush(QColor("#e8f5e9")))
        ellipse.setPen(QPen(QColor("#66bb6a"), 2))
        self.scene.addItem(ellipse)

        text = QGraphicsTextItem(name)
        text.setDefaultTextColor(QColor("#1b5e20"))
        text.setFont(QFont("Segoe UI", 11, weight=QFont.Weight.Bold))
        text.setPos(x + 10, y + 25)
        self.scene.addItem(text)

        return QPointF(x + radius / 2, y + radius)

    def draw_line(self, p1: QPointF, p2: QPointF):
        line = QGraphicsLineItem(p1.x(), p1.y(), p2.x(), p2.y())
        line.setPen(QPen(QColor("#388e3c"), 2))
        self.scene.addItem(line)

    def draw_tree(self):
        # Верхній рівень
```

```
        p1 = self.draw_person("👴 Дідусь", 100, 20)
        p2 = self.draw_person("👵 Бабуся", 350, 20)

        # Батько
        p3 = self.draw_person("👨 Батько", 225, 150)
        self.draw_line(p1, QPointF(260, 150))
        self.draw_line(p2, QPointF(260, 150))

        # Мати
        p4 = self.draw_person("👩 Мати", 475, 150)

        # Діти
        p5 = self.draw_person("🧒 Я", 280, 300)
        p6 = self.draw_person("👦 Брат", 420, 300)

        # Зв'язки
        self.draw_line(QPointF(260, 220), QPointF(315, 300))  # Батько → Я
        self.draw_line(QPointF(510, 220), QPointF(455, 300))  # Мати → Брат
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Сімейне дерево – Мій профіль")
        self.setMinimumSize(1200, 720)
        self.setStyleSheet("""            * {                font-family: 'Segoe
UI', sans-serif;            }            QMainWindow {
background-color: #f1f8e9;            }            QTabWidget::pane {
border: none;            }            QTabBar::tab {                background:
#c8e6c9;                padding: 10px 25px;                font-weight: bold;
font-size: 15px;                color: #1b5e20;                border-top-left-
radius: 8px;                border-top-right-radius: 8px;            }
QTabBar::tab:selected {                background: #a5d6a7;
border: 1px solid #66bb6a;                border-bottom: 1px solid #a5d6a7;
}            QListWidget {                background-color: #ffffff;
border: 1px solid #81c784;                padding: 10px;                font-
size: 14px;                color: #2e7d32;                border-radius: 10px;
}            QStatusBar {                background-color: #c8e6c9;
font-size: 13px;                color: #1b5e20;                padding: 5px;
}            QFrame {                background-color: #ffffff;
border: 1px solid #c5e1a5;                border-radius: 12px;
padding: 10px;            }        """)

        # Tabs
        tabs = QTabWidget()
        tabs.addTab(QWidget(), "🌳 Сімейне дерево")
        tabs.addTab(QWidget(), "💬 Повідомлення")
        tabs.addTab(QWidget(), "👤 Профіль")
        tabs.addTab(QWidget(), "⚙️ Налаштування")

        # Layout
        central_widget = QWidget()
        layout = QHBoxLayout()
        layout.setContentsMargins(15, 15, 15, 15)
        layout.setSpacing(20)

        # Left – relatives list
        relatives = QListWidget()
```

```
        relatives.addItems(["👨 Батько", "👩 Мати", "👦 Брат", "👧 Сестра",
"👴 Дідусь", "👵 Бабуся"])
        relatives.setMinimumWidth(220)

        # Right – tree
        tree = FamilyTreeWidget()
        tree_frame = QFrame()
        tree_layout = QVBoxLayout()
        tree_layout.addWidget(tree)
        tree_frame.setLayout(tree_layout)

        layout.addWidget(relatives, 2)
        layout.addWidget(tree_frame, 6)
        central_widget.setLayout(layout)

        self.setMenuWidget(tabs)
        self.setCentralWidget(central_widget)

        status = QStatusBar()
        status.showMessage("🟢 Сесія активна – Користувач: test_user")
        self.setStatusBar(status)
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

ДОДАТОК Б

Реалізація чат-форми

```
import sysfrom PyQt6.QtWidgets import (
    QApplication, QWidget, QLabel, QVBoxLayout, QHBoxLayout,
    QPushButton, QLineEdit, QScrollArea, QFrame
)from PyQt6.QtGui import QFont, QPixmapfrom PyQt6.QtCore import Qt

class MessengerWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("💬 Месенджер")
        self.setMinimumSize(1150, 700)
        self.setStyleSheet("""            QWidget {                background-
color: #e8f5e9;                font-family: 'Segoe UI', sans-serif;
font-size: 14px;            }            QPushButton {
background-color: #43a047;                color: white;                padding:
10px 22px;                border: none;                border-radius: 18px;
font-weight: bold;            }            QPushButton:hover {
background-color: #2e7d32;            }            QLineEdit {
background-color: #ffffff;                border: 1px solid #c8e6c9;
border-radius: 20px;                padding: 10px 16px;                color:
#2e7d32;            }            QScrollArea {                border: none;
background-color: #f9fbe7;            }            QFrame#ChatBox {
background-color: #ffffff;                border-radius: 18px;
padding: 15px;                border: 1px solid #c5e1a5;            }
QLabel#MessageBubbleLeft {                background-color: #ffffff;
color: #2e7d32;                border-radius: 20px;                padding: 12px
16px;                max-width: 300px;                border: 1px solid #c8e6c9;
}            QLabel#MessageBubbleRight {                background-color:
#a5d6a7;                color: #1b5e20;                border-radius: 20px;
padding: 12px 16px;                max-width: 300px;                border: 1px
solid #81c784;            }            QLabel#Username {                font-
weight: 600;                font-size: 15px;                color: #1b5e20;
}            QLabel#StatusOnline {                color: #388e3c;
font-size: 13px;            }            QLabel#StatusOffline {
color: #9e9e9e;                font-size: 13px;            }
QFrame#UserCard {                background-color: #ffffff;
border-radius: 14px;                padding: 6px 12px;                border:
1px solid #c5e1a5;            }        """)

        self.setup_ui()

    def setup_ui(self):
```

```
        main_layout = QVBoxLayout(self)
        menu_layout = QHBoxLayout()
        for name in ["🌳 Сімейне дерево", "💬 Повідомлення", "👤 Профіль"]:
            btn = QPushButton(name)
            btn.setFixedHeight(40)
            btn.setCursor(Qt.CursorShape.PointingHandCursor)
            menu_layout.addWidget(btn)
        menu_layout.addStretch()
        main_layout.addLayout(menu_layout)

        content_layout = QHBoxLayout()
        self.chat_list = QVBoxLayout()
        self.chat_list.setSpacing(15)
        self.chat_list.setContentsMargins(10, 10, 10, 10)

        for name, status, avatar_path in [
            ("Ірина Бойко", "онлайн", "avatar.png"),
            ("Микола Сидоренко", "офлайн", "avatar.png"),
            ("Олександр Петренко", "онлайн", "avatar.png")
        ]:
            profile = QFrame()
            profile.setObjectName("UserCard")
            profile.setFixedHeight(70)
            profile_layout = QHBoxLayout(profile)
            profile_layout.setContentsMargins(8, 8, 8, 8)

            avatar = QLabel()
            avatar.setPixmap(QPixmap(avatar_path).scaled(45, 45,
Qt.AspectRatioMode.KeepAspectRatio, Qt.TransformationMode.SmoothTransformation))
            avatar.setFixedSize(45, 45)
            avatar.setStyleSheet("border-radius: 22px; border: 2px solid
#66bb6a;")

            label_name = QLabel(name)
            label_name.setObjectName("Username")

            label_status = QLabel("🟢 онлайн" if status == "онлайн" else "⚪
офлайн")
            label_status.setObjectName("StatusOnline" if status == "онлайн" else
"StatusOffline")

            text_layout = QVBoxLayout()
            text_layout.addWidget(label_name)
            text_layout.addWidget(label_status)

            profile_layout.addWidget(avatar)
            profile_layout.addLayout(text_layout)
            self.chat_list.addWidget(profile)

        chat_list_container = QFrame()
        chat_list_container.setLayout(self.chat_list)
        chat_list_container.setMaximumWidth(280)

        chat_frame = QFrame()
        chat_frame.setObjectName("ChatBox")
        chat_layout = QVBoxLayout(chat_frame)
        chat_layout.setContentsMargins(15, 15, 15, 15)
        chat_layout.setSpacing(10)
```

```
        header_layout = QHBoxLayout()
        name = QLabel("Ірина Бойко")
        name.setFont(QFont("Segoe UI", 14, weight=QFont.Weight.Bold))
        status = QLabel("🟢 Онлайн")
        status.setObjectName("StatusOnline")
        header_layout.addWidget(name)
        header_layout.addSpacing(10)
        header_layout.addWidget(status)
        header_layout.addStretch()
        chat_layout.addLayout(header_layout)

        scroll = QScrollArea()
        scroll.setWidgetResizable(True)

        messages_container = QWidget()
        self.messages_layout = QVBoxLayout(messages_container)
        self.messages_layout.setSpacing(10)
        self.messages_layout.setContentsMargins(10, 10, 10, 10)

        self.add_message("Привіт! Як справи?", incoming=True)
        self.add_message("Привіт! Усе добре, дякую 😄", incoming=False)
        self.add_message("Чудово! Коли зустрінемось?", incoming=True)

        scroll.setWidget(messages_container)
        chat_layout.addWidget(scroll)

        input_layout = QHBoxLayout()
        self.message_input = QLineEdit()
        self.message_input.setPlaceholderText("Введіть повідомлення...")
        send_btn = QPushButton("📤 Надіслати")
        send_btn.setFixedWidth(120)
        input_layout.addWidget(self.message_input)
        input_layout.addWidget(send_btn)
        chat_layout.addLayout(input_layout)

        content_layout.addWidget(chat_list_container, 1)
        content_layout.addWidget(chat_frame, 3)
        main_layout.addLayout(content_layout)

    def add_message(self, text: str, incoming: bool = True):
        bubble = QLabel(text)
        bubble.setWordWrap(True)
        bubble.setObjectName("MessageBubbleLeft" if incoming else
"MessageBubbleRight")
        layout = QHBoxLayout()
        if incoming:
            layout.addWidget(bubble)
            layout.addStretch()
        else:
            layout.addStretch()
            layout.addWidget(bubble)
        self.messages_layout.addLayout(layout)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MessengerWindow()
    window.show()
```

```
sys.exit(app.exec())
```