Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка P2P – системи для децентралізованого обміну файлами з використанням технології .Net

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Олійник Ю.Р. (прізвище та ініціали)

Керівник (підпис) Петрик М.Р. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) Лечаченко Т.А. (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Олійнику Юрію Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка P2P-системи для децентралізованого обміну файлами з використанням технології .Net

Керівник роботи Петрик Михайло Романович, доктор фізико-математичних наук, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Методичні вказівки, рекомендації кафедри, наукові публікації, офіційна технічна документація з розробки програмного забезпечення

4. Зміст роботи (перелік питань, які потрібно розробити)
Огляд предметної області децентралізованого обміну файлами, формування вимог до системи, проєктування архітектури програмного забезпечення, конструювання P2P-системи для обміну файлами, розробка графічного інтерфейсу користувача, тестування програмного забезпечення та верифікація вимог, загальні вимоги безпеки до обладнання та технологічних процесів, долікарська допомога при ранах

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
UML-діаграма ієрархії класів, UML-діаграма варіантів використання, UML-діаграма підсистем, UML-діаграми послідовностей, рисунки лістингів програмного коду, рисунки графічного Інтерфейсу користувача, слайди презентації

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Лазарюк В.В., к.т.н., доц., доц. каф. МТ		
Нормоконтроль	Стоянов Ю.М., к.т.н., доц. каф. ПІ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та затвердження теми кваліфікаційної роботи		
2.	Дослідження предметної області		
3.	Аналіз вимог та вибір стеку технологій		
4.	Підготовка середовища розробки та початкове налаштування проєкту		
5.	Налаштування системи контролю версій і організація Scrum-процесу засобами GitHub		
6.	Проєктування архітектури програмного забезпечення		
7.	Конструювання програмного рішення		
8.	Тестування системи та верифікація вимог		
9.	Безпека життєдіяльності, основи охорони праці		
10.	Нормоконтроль		
11,	Перевірка на плагіат		
12.	Підготовка презентації та доповіді		
13.	Попередній захист кваліфікаційної роботи		
14.	Захист кваліфікаційної роботи		

Студент _____ Олійник Ю.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025, Сторінок 71, таблиць 1, рисунків 27, додатків 2, презентація.

Тема: Розробка P2P-системи для децентралізованого обміну файлами з використанням технології .Net.

У даній кваліфікаційній роботі бакалавра спроєктовано та розроблено спрощену децентралізовану систему для обміну файлами в межах локальної мережі. Ця система надає функціональні можливості для зручного обміну файлами в приватній мережі між пристроями, що у ній взаємодіють. Програмний продукт реалізовано як настільний застосунок, що забезпечує тісну інтеграцію із системою користувача. В цій роботі було проведено аналіз предметної області та порівняння аналогічних рішень, що існують на ринку програмного забезпечення. Здійснено аналіз вимог, спроєктовано архітектуру системи. Розроблено програмне рішення, проведено тестування та верифікацію вимог до системи.

Розроблено програмне рішення на платформі .Net з використанням фреймворку для графічного інтерфейсу користувача WPF, що підтримує виключно платформу Windows. Система застосовує шифрування та використовує симетричні ключі під час передачі файлів та авторизації користувача. Підтримує багатопотокове надсилання та отримання файлів від різних користувачів. Модульна архітектура даного рішення забезпечує легку підтримку та можливість розширення можливостей системи у майбутньому.

Ключові слова: децентралізація, обмін файлами, P2P, desktop-застосунок, проєктування, тестування, C#, .Net.

ABSTRACT

Bachelor's qualification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 “Software Engineering”. TNTU, 2025, Pages 71, tables 1, figures 27, appendices 2, presentation.

Theme: Development of a P2P system for decentralized file sharing using .Net technology.

This bachelor's thesis designs and develops a simplified decentralized system for file sharing within a local network. This system provides the ability to easily exchange files in a private network between devices that interact within it. The software product is implemented as a desktop application that provides tight integration with the user's system. This thesis analyzes the subject area and compares similar solutions available on the software market. Requirements were analyzed and the system architecture was designed. A software solution was developed, tested, and verified against requirements.

A software solution has been developed on the .Net platform using the WPF graphical user interface framework, which supports only the Windows platform. The system uses encryption and symmetric keys during file transfer and user authorization. It supports multithreaded sending and receiving of files from different users. The modular architecture of this solution ensures easy maintenance and the possibility of expanding the system's capabilities in the future.

Keywords: decentralization, file sharing, P2P, desktop application, system design, testing, C#, .Net.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕМІНІВ

P2P (Peer-to-peer) – архітектурна модель мережі, в якій усі вузли взаємодіють без посередника (сервера).

Вузол – окремий учасник Peer-to-peer мережі.

Децентралізація – принцип роботи P2P-fмережі, основою якого є відсутність централізованого керування та рівноправність всіх її учасників.

Трекер – сервер для координації вузлів у P2P-мережі.

IP (Internet Protocol address) – унікальний ідентифікатор пристрою в мережі.

TCP (Transmission Control Protocol) – протокол передачі даних в комп’ютерних мережах із встановленням з’єднання.

UDP (User Datagram Protocol) – протокол передачі даних в комп’ютерних мережах без встановлення з’єднання.

Multicast – спосіб одночасної передачі даних від одного джерела до групи отримувачів.

C# – об'єктно-орієнтована мова програмування для платформи .Net.

.Net – платформа для створення застосунків під різні операційні системи.

WPF (Windows Presentation Foundation) – .NET платформа для створення Windows-застосунків з графічним інтерфейсом користувача.

xUnit – фреймворк для написання тестів на платформі .NET.

Moq – бібліотека для імітацій об’єктів під час розробки програмних систем.

Serilog – бібліотека логування для платформи .NET.

RSA (Rivest, Shamir, Adleman) – алгоритм асиметричного шифрування на основі пари ключів.

AES-256 (Advanced Encryption Standard) – симетричний алгоритм шифрування.

SHA-256 / SHA-512 (Secure Hash Algorithm) – алгоритми хешування.

UML (Unified Modeling Language) – мова моделювання програмних систем.

GUI (Graphical User Interface) – графічний інтерфейс користувача.

ЗМІСТ

ВСТУП

Двадцять перше століття можна без вагань назвати століттям інформаційних технологій. Інформація оточує нас всюди, де б ми не були. Основним способом розповсюдження інформації в сучасному світі став інтернет, як невід'ємна частина нашого життя. Інтернет-мережі повсюди: вдома, на роботі, в кафе, на вулиці.. Більша частина населення планети має доступ до цього чудового інструменту. Завдяки інтернет-технології поширення новин чи обмін електронними листами більше не становить труднощів. На сьогоднішній день ці завдання є одними з найбільш типових, що вирішує глобальна мережа.

Більш специфічним завданням є обмін безпосередньо файлами між користувачами мережі, особливо коли ці користувачі знаходяться в одній локальній мережі. Для такого типу завдань зазвичай використовуються поштові клієнти або месенджери. Проте, такий процес передбачає посередника у вигляді сервера, що зменшує ефективність передачі та ставить під сумнів питання збереження конфіденційності переданої інформації. Для таких випадків чудовою альтернативою є пряма P2P (peer-to-peer) передача даних між користувачами.

Ідеєю цієї роботи є створення надійного та просто у використанні застосунку для P2P передачі файлів у домашній або корпоративній мережі. Цей застосунок є чудовою альтернативою різноманітних torrent-клієнтів та сервісів, що передбачають використання глобальної мережі інтернет. Кожна операційна система для настільних ПК має вбудований функціонал для такої взаємодії, проте, потребує глибших знань та вмінь користувача.

Метою застосунку, що розробляється у цій роботі є спрощення процесу передачі файлів для посереднього користувача ПК, зберігаючи надійність та захищеність. Основну увагу приділено саме безпеці, що є однією з найважливіших вимог для корпоративних рішень. Система застосовує шифрування файлів за допомогою асиметричної пари ключів, які ж використовуються для авторизації у застосунок. Доступ до системи захищено PIN-

кодом, який і відкриває доступ до пари ключів. Застосунок містить мінімальну кількість налаштувань, а вся робота з безпековими ключами відбувається «під капотом». Взаємодія із програмою обмежена роботою з доданими у систему спільними файлами, збереженими вузлами та отриманими файлами від інших вузлів.

Для розробки програмного рішення було використано сучасну об’єктно-орієнтовану мову програмування C# 12 на платформі .Net 8 з використанням фреймворку WPF, що забезпечує роботу застосунку на найбільш поширеній операційній системі — Windows. Дане рішення використовує мережеві протоколи UDP та TCP для визначення вузлів у мережі та їх прямої взаємодії. Для шифрування використано стандарт шифрування AES-256 та криптосистему RSA. Для перевірки цілісності переданих файлів використано алгоритми хешування SHA-256 та SHA-512, які застосовуються залежно від умов системи. Тестування проводиться з підтримкою фреймворку xUnit та бібліотеки Moq. Система логування реалізована на базі фреймворку Serilog.

Дана кваліфікаційна робота має на меті продемонструвати знання та навички набуті на шляху до здобуття освітнього ступеня бакалавра. У роботі продемонстровано ключові етапи життєвого циклу розробки програмного забезпечення. Від аналізу предметної області до тестування розробленої системи. Значну частину уваги приділено проєктуванню архітектури рішення. Застосовано ряд архітектурних патернів, що включає фасад, спостерігач та інші. Система побудована на основі модульної архітектури, покрита модульними та інтеграційними тест-кейсами. Комплексний підхід до розробки системи забезпечує можливість ефективного розширення можливостей даного рішення у майбутньому.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ СИСТЕМ ДЛЯ ДЕЦЕНТРАЛІЗОВАНОГО ОБМІНУ ФАЙЛАМИ

1.1 Огляд конкурентів

Ринок програмного забезпечення P2P-систем для обміну файлами має багато різноманітних рішень для будь-яких цілей. Хмарні, локальні, гібридні рішення, для корпоративного середовища чи особистого використання. До уваги взято системи, які більшою мірою відповідають тим же критеріям, що й запропоноване рішення. До таких критеріїв відноситься виключно локальна робота, а також призначення застосунку для домашнього або корпоративного використання.

Одним з найвідоміших програмних рішень для обміну файлами сьогодні залишається uTorrent, програмне забезпечення на базі протоколу BitTorrent.

BitTorrent – це протокол для розповсюдження файлів. Він ідентифікує контент за URL-адресою та розроблений для безперешкодної інтеграції з Інтернетом. Його перевага над звичайним HTTP полягає в тому, що коли одночасно відбувається кілька завантажень одного й того ж файлу, завантажувачі завантажують файли один одному, що дозволяє джерелу файлів підтримувати дуже велику кількість завантажувачів лише з незначним збільшенням навантаження [1].

uTorrent — один із найпопулярніших клієнтів BitTorrent у світі. Свою популярність клієнт здобув завдяки високій пропускній здатності, точним контролем над швидкістю передачі та простоті використання. Окрім цього цей клієнт містить широкий спектр функціональних можливостей, серед яких відновлення зупинених завантажень, керування їх послідовністю, планування завантажень, та інші. Серед цікавих можливостей uTorrent є пошук та завантаження медіа-контенту через офіційні пакети BitTorrent [2].

На рисунку 1 зображено логотип uTorrent.

Рисунок 1. Логотип застосунку uTorrent [3]

Іншою сучасною P2P-системою є Resilio Sync. Це програмне забезпечення також використовує технологію BitTorrent. Його завданням є синхронізація файлів між пристроями користувача. Для встановлення зв'язку використовується як локальна мережа, так і допоміжні засоби, такі як використання статичних відомих вузлів, DHT та класичних трекерів BitTorrent. Є два варіанти обміну файлами: напряму між пристроями або із використанням ретрансляційних серверів. Трафік шифрується алгоритмом шифрування AES-256. Після встановлення клієнт дозволяє згенерувати 21-символьний секретний ключ, що надає можливість зв'язати різні пристрої для синхронізації файлів між ними. Таким чином весь процес синхронізації не залежить від посередників [4].

На рисунку 2 зображено логотип BitTorrent Sync.

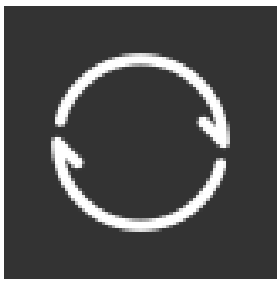

Рисунок 2. Логотип застосунку Resilio Sync (BitTorrent Sync) [5]

Syncthing — відкрита, надійна та децентралізована альтернатива хмарним платформам синхронізації. Syncthing імітує функціонал хмарних сервісів, надаючи можливість резервоного копіювання даних, проте, без центрального сховища. Сервіс використовує технологію P2P, використовуючи шифрування,

таким чином, дані передаються між комп’ютерами, мобільними телефонами та іншими пристроями без посередників [6].

На відміну від попередньо описаних сервісів Syncthing використовує протокол обміну блоками BEP. Цей протокол використовується між двома і більше пристроями утворюючи кластер. Файли передаються блоками даних. Якщо файл було змінено, то передаються лише блоки, що змінилися. Використовується шифрування за допомогою TLS [7].

Пристрої мають унікальні ідентифікатори, на базі хеш алгоритму SHA-256. Для виявлення пристроїв передбачено кілька способів: ручне встановлення IP-адреси та порта, виявлення пристроїв у локальній мережі, та використання служби глобального виявлення [8].

На рисунку 3 зображено логотип Syncthing.

Рисунок 3. Логотип застосунку Syncthing [9]

Отже, цифровий ринок наповнений різноманіттям програмних рішень для вирішення завдання децентралізованого обміну файлами. Кожне рішення є унікальним, хоч використовує схожий принцип роботи. uTorrent спрямований на «масового користувача», тобто на широку аудиторію, оскільки використовує публічні торрент-трекери, його метою є саме поширення файлів. Resilio Sync позиціонується як закрите програмне забезпечення як для комерційних установ, так і для приватного користувача. Надає можливості обміну файлів та їх синхронізації між пристроями. Syncthing у свою чергу є повністю відкритим проєктом, що орієнтований на синхронізацію даних.

1.2 Обґрунтування вибору напрямку дослідження

Рішення реалізувати децентралізований P2P файлообмінник обумовлено цікавістю до P2P технології та низьким рівнем обізнаності у сфері мережевого програмування та безпеки. Ідея цього проєкту виникла внаслідок аналізу власних потреб в корисних програмних рішеннях для повсякденного користування.

Основною технологією, що використовувалась для розробки було обрано рішення від компанії Microsoft — .Net. Це рішення приймалось з огляду на попередній практичний досвід та зацікавленість у поглибленні знань та досвіду з цією та пов’язаними технологіями.

.Net — це безкоштовна платформа з відкритим вихідним кодом для розробки додатків, призначених для різних типів пристроїв та операційних систем. Платформа підтримує запуск коду, написаний багатьма мовами програмування, серед яких найпопулярнішою мовою є C#. Серед особливостей .Net варто виділити: автоматичне керування пам’яттю, строга типізація змінних, можливість паралельного програмування та портативність коду для різних архітектур та операційних систем [10]. Всі ці аспекти суттєво допомагають при розробці технічно складних рішень.

Мову програмування було обрано C#. Ця мова також є безкоштовним кросплатформним рішенням, призначеним для розробки на різні типи пристроїв та різні операційні системи. До того ж, це найпопулярніше мова для платформи .Net. C# є об’єктно орієнтованою мовою зі строгою типізацією та C-подібним синтаксисом. Її основа покладена на ООП принципах, тим не менш, вона також містить функціонал з інших парадигм [11].

Для реалізації P2P мережі спершу було прийнято рішення використати готовий користувацький фреймворк для виявлення пристроїв у мережі. Проте, його реалізація для C# була лише частково реалізована, тому, все ж було використано стандартний протокол UDP. Для безпосередньої взаємодії між пірами було використано протокол TCP.

Важливою частиною P2P-систем файлообміну є безпека. В даній реалізації існує кілька ключових моментів пов’язаних з безпекою. Шифрування трафіку відбувається шляхом використання алгоритму шифрування AES-256.

AES — це всесвітній стандарт та один з найпоширеніших криптографічних алгоритмів, розроблений у 1997 році. Залишається одним з найбезпечніших стандартів блокового шифрування у моделі з одним невідомим ключем [12].

Шифрування та дешифрування файлів під час передачі відбувається за допомогою AES-ключа. Цей ключ шифрується та дешифрується парою ключів RSA. Публічний ключ розповсюджується в мережі за допомогою раніше згаданого протоколу UDP. Приватний ключ шифрується окремо за допомогою іншого ключа, який в свою чергу генерується з використанням PIN-коду користувача та зберігається локально у файлі з розширенням .key.

RSA — це метод шифрування, що реалізує криптосистему з відкритим ключем. Безпека методу частково ґрунтується на складності факторизації великих чисел. Суть методу полягає у тому, що публічне розкриття ключа шифрування не надає можливості розшифрувати дані. Таким чином, лише отримувач може розшифрувати повідомлення [13].

Шифрування приватного ключа RSA відбувається шляхом використання функції PBKDF2, що призначена для генерації ключа шифрування з парольної фрази та рядка солі [14].

Перевірка цілісності переданих файлів здійснюється за допомогою SHA-256 або SHA-512 — хеш-функції, що належать до сімейства хеш-алгоритмів SHA. Суть даних алгоритмів полягає у створенні рядка фіксованої довжини з довільної кількості вхідних даних. Відмінність цих функцій полягає у складності обчислення та, відповідно, у розмірі вихідних значень [15].

Для взаємодії користувача з програмою було прийнято рішення використати сучасний фреймворк, що підтримує кросплатформність. Проте, через обмежені часові ресурси та проблеми з тестами на інших платформах було прийнято рішення зосередитись на сумісності з операційною системою Windows. Серед переліку різноманітних фреймворків для створення користувацького інтерфейсу,

що сумісні з .Net, було обрано WPF. Одним з факторів вибору цього рішення був наявний практичний досвід розробки з використанням цієї технології.

WPF — це платформа інтерфейсу користувача для сучасних систем, що надає функціональні можливості для розробки застосунків. Використовує векторний механізм візуалізації, підтримує розширювану мову розмітки додатків, дво- та тривимірну графіку, анімацію, стилі та інше. Існує дві версії WPF, версія .Net підтримує лише Windows [16].

Також було використано ряд фреймворків для зручного тестування системи та логування дій користувача. Для інтеграційного та Unit-тестування було обрано фреймворк xUnit та бібліотеку Moq. Логування відбувалось з використанням бібліотеки Serilog.

За даними офіційного сайту [17], xUnit.net — це безкоштовний інструмент з відкритим вихідним кодом, орієнтований на спільноту, для модульного тестування .Net Framework.

За даними офіційної документації [18], Moq — це бібліотека для створення макетів у .Net. Підтримує імітацію класів та інтерфейсів, надає можливості використання дерева виразів .Net Linq та лямбда-виразів. Надає розробнику простий механізм налаштування залежностей для модульного тестування.

Serilog — це проста бібліотека для ведення діагностичного журналу для різних типів ресурсів у .Net застосунках [19].

Окрім технологій, що використовуються під час розробки важливим кроком є вибір інструментів розробки, що забезпечать зручність та продуктивність усього процесу реалізації програмного рішення. Середовищем розробки було обрано рішення від компанії JetBrains — JetBrains Rider. Для розробки використовувалось безкоштовна розширена версія застосунку для студентів.

За даними офіційного сайту [20], JetBrains Rider — це універсальне, кросплатформне середовище розробки, створене для розробників, які працюють з повним технологічним стеком платформи .Net.

Для забезпечення надійності та зручності розробки хорошою практикою є використання системи контролю версіями. Найпопулярнішою системою в цій

царині є Git. Для керування версіями проєкту було задіяно саме цю систему в поєднанні з сервісом GitHub.

Згідно з офіційним визначенням [21]: Git — це безкоштовна розподілена система контролю версій з відкритим кодом, розроблена для швидкої та ефективної обробки будь-яких проектів, від малих до дуже великих.

GitHub — хмарна платформа для зберігання, поширення та спільної роботи над програмним кодом. Платформа тісно інтегрована з системою контролю версій Git. Ця зв’язка дозволяє працювати з файлами локально та синхронізувати зміни з «віддаленим» репозиторієм на GitHub [22].

Для спрощення процесу розробки проєкту та підвищення ефективності роботи було використано сучасні інструменти зі штучним інтелектом, а саме ChatGPT, Claude та GitHub Copilot. Ці рішення слугували допоміжними засобами протягом всього життєвого циклу розробки програмного забезпечення. Від генерації ідей, допомоги з проєктуванням архітектури, написання та вдосконалення коду, до формування текстової частини пояснювальної записки. Завдяки цим інструментам вдалось підтримувати хороший рівень проєкту з використанням сучасних рішень та підходів до розробки програмного забезпечення.

У процесі підготовки англомовних елементів використовувались допоміжні інструменти перекладу, зокрема DeepL.

1.3 Методологія розробки

Успішна розробка проєкту вимагає добре структурованого плану, правил та принципів у сфері менеджменту, дотримання яких дозволить не лише ефективно працювати над розробкою проєкту, а й успішно досягнути поставлених цілей у зазначені терміни. Існує велика кількість методологій та підходів до розробки

програмного забезпечення. Для невеликих проєктів Scrum є найбільш оптимальним варіантом.

Scrum — підхід до управління проєктами, основою якого є розбиття роботи на ітерації, так звані спринти. Тривалість спринтів сягає від одного до чотирьох тижнів, результатом кожного спринта є готовий робочий продукт, який можна протестувати та використовувати у подальшій розробці. Загалом, Scrum є підходом в методології Agile, яка власне фокусує увагу на зміни та взаємодію між людьми в команді. Завдяки використанню коротких циклів під час роботи над проєктом, Scrum є дуже гнучкою методологією, що забезпечує швидку адаптацію команди до змін [23].

За матеріалами статті [23]: існує кілька основних ролей в Scrum-підході. Product Owner, тобто власник продукту, формулює бачення продукту, тобто розуміє всю його цілісність, контролює процес розробки, концентруючись на головних цілях проєкту. Scrum Master, це та особа, що займається плануванням списку завдань, допомагає команді ефективно виконувати свої задачі. Останньою ланкою є авжеж команда розробників, які займаються виконанням поставлених задач.

Згідно з наведеними матеріалами, дана методологія також передбачає ряд артефактів, які активно використовуються протягом спринтів, а саме Product Backlog (список усіх завдань проєкту), Sprint Backlog (список завдань на окремий спринт) та Increment (кінцевий продукт спринта).

До особливостей даної методології належать певні події, що відбуваються протягом усього етапу роботи команди над проєктом. На початку кожного спринта відбувається подія планування спринта, визначаються цілі команди, вибираються завдання, планується робота. Після цього деякий час триває спринт (як зазначалось раніше, тривалість від 1 до 4 тижнів). Крім цього, цей процес супроводжують щоденні зустрічі команди, на яких команда ділиться інформацією про стан роботи, ця подія називається щоденний Scrum. Після завершення ітерації, відбувається огляд спринту, підсумування результатів та отримання зворотного зв’язку від власника продукту. Ще однією подією є ретроспектива

спринту. Під час цієї події команда аналізує процес роботи та намагається знайти шляхи покращення його ефективності.

В рамках розробки поточного проєкту активно використовувався адаптований підхід на основі методології Scrum. Оскільки робота над проєктом здійснювалась однією особою, робота із методологією дещо відрізнялась від стандартної практики. Розробник одночасно також виконував обов’язки власника продукту і Scrum Master-а. Події обмежились плануванням, безпосередньо спринтом та оглядом результатів. Надалі у тексті термін "Scrum" вживається з урахуванням цієї адаптації

Робота з методологією Scrum відбувалась у GitHub-репозиторії. Організація відбувалась у GitHub Projects, в GitHub-репозиторії проєкту, що розробляється в рамках цієї кваліфікаційної роботи. Задачі формувались та підтримувались у GitHub Issues, які інтегруються з GitHub Projects. Це забезпечило гнучкий контроль над процесом розробки. Налаштування Scrum-проєкту було здійснено за матеріалами, поданими у відеоролику [24].

Таким чином було сформовано ряд беклогів. Таблиця зі всіма історіями, дошки завдань, що відображають статус історій наступного та поточного спринта, а також дорожні карти для епіків, спринтів та історій, що допомагають відстежувати та планувати ці всі події за часовою шкалою. Варто звернути увагу, що також було використано поняття Epic. Згідно з визначенням, поданим у статті [25], Epic — це велика частина, що є надто великою для виконання у одній ітерації, тобто спринті. Під час організації Scrum створювались картки у вигляді GitHub Issues з описом історій, спринтів та епіків, а також Sprint Review. Для формування карток використовувались шаблони, згенеровані Claude AI, а також повністю готові рішення, які генерувались за запитом.

Отже, під час розробки цього проєкту було прийнято рішення сформувати три епіки та 7 історій, проте, ще на початковому етапі список було скорочено до шести. В цьому випадку спринти умовно позначали певну функціональну частину продукту.

Список спринтів, запланованих у цьому проєкті:

- Реалізація базової P2P-мережі
- Впровадження функціоналу передачі файлів
- Розробка механізму контролю доступу користувачів
- Реалізація авторизації та безпечної передачі
- Розробка графічного інтерфейсу користувача
- Оптимізація, виправлення та тестування

Кожен спринт повинен мати однакову визначену тривалість [26]. Однак, через брак досвіду та теоретичної основи цей принцип не був дотриманий. Загальна тривалість розробки складала три місяці, а кожен спринт тривав від кількох днів до кількох тижнів, та навіть більше місяця. На рисунку 4 зображено «backlog» усіх історій.

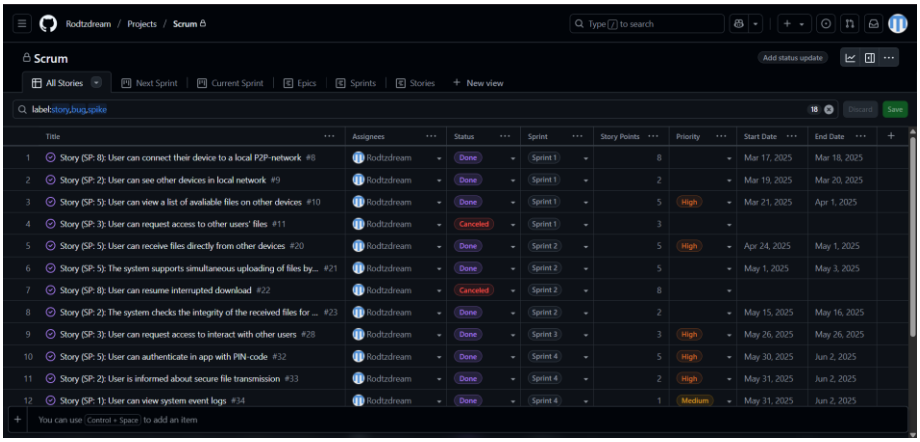

Рисунок 4. Backlog усіх історій проєкту в GitHub Projects

До кожної історії позначено дату початку та завершення виконання. Проте, ці дати позначають саме запланований термін виконання завдань, але не завжди відповідають їх фактичній тривалості. Значення Story Points обиралось з вибірки чисел Фібоначі. ЇЇ особливість полягає у тому, що кожне наступне число є сумою двох попередній. Використання саме цієї послідовності чисел часто застосовують для оцінки «розміру» завдань та «історій» для майбутніх спринтів [27].

Варто також відзначити, що оскільки планування та розробка здійснювались однією особою, детальне впровадження методології спричиняло деякі загальні затримки. На рисунку 5 зображено приклад однієї з історій.

Рисунок 5. Картка Sprint Story в GitHub Issues

Загалом, незважаючи на всі неточності та проблеми, які виникали під час застосування Scrum-підходу, дана методологія допомогла організувати та структурувати весь процес розробки проєкту.

1.4 Формування вимог до системи

Аналіз та формування вимог вважається одним з основних етапів циклу розробки програмного забезпечення. На цьому етапі важливо чітко визначити основні функціональні та нефункціональні вимоги до системи, що розроблятиметься.

Відповідно до лекційних матеріалів курсу SE322 [28], «функціональні вимоги регламентують функціонування або поведінку системи». Також у матеріалах зазначено, що «ці вимоги відповідають на питання "що повинна робити система" в тих або інших ситуаціях». Нефункціональні вимоги, у свою чергу: «регламентують внутрішні і зовнішні умови або атрибути функціонування системи».

На етапі планування проєкту з підтримкою штучного інтелекту (ChatGPT) було сформульовано та додатково опрацьовано ряд вимог.

Функціональні вимоги:

- Підключення пристроїв: користувачі можуть підключатися до мережі через локальну або глобальну (у майбутньому) мережу.
- Додавання спільних файлів та папок: користувач може відзначати файли та папки, які будуть доступні іншим учасникам.
- Перегляд метаданих доступних файлів: інші учасники мережі можуть бачити список спільних файлів (назва, розмір, дата зміни).
- Передача файлів між пристроями: можливість завантажувати файли безпосередньо від інших користувачів у мережі.
- Контроль доступу: користувач може вручну надавати чи відкликати дозвіл на доступ до своїх файлів. Доступ можливий за одним із рівнів: всім у мережі, лише обраним пристроям, нікому (лише за підтвердженням запиту).
- Логування дій: система фіксує основні події, попередження й помилки

Нефункціональні вимоги:

- Продуктивність: система повинна ефективно працювати у локальній мережі з мінімальними затримками при пошуку та передачі файлів.
- Надійність: після передачі файлів система має перевіряти їхню коректність, щоб уникнути помилок і пошкоджених даних. Якщо передача файлу була перервана, її слід відновлювати з місця зупинки (опціонально).
- Безпека: користувач самостійно визначає, хто може отримувати доступ до його файлів. Передача файлів відбувається зашифрованими каналами для захисту від перехоплення. Автентифікація відбувається за допомогою паролів, приватних ключів для захисту доступу до програми.
- Зручність використання: система має бути простою у використанні, надаючи зручні механізми для управління файлами та доступом до них. Забезпечувати легке налаштування мережі та підключення пристроїв.

Таким чином, було сформовано вимоги до системи. У процесі створення проєкту вимоги дещо змінювались, проте основні залишились незмінними. Після завершення формування вимог можна переходити до проєктування системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ P2P-СИСТЕМИ ДЛЯ ОБМІНУ ФАЙЛАМИ

Проєктування програмного забезпечення є ключовим етапом життєвого циклу розробки ПЗ. Суть цього етапу полягає у моделюванні системи на простому абстрактному рівні, з плавним переходом на програмний, технічний рівень.

За теоретичними матеріалами курсу «Моделювання та аналіз програмного забезпечення» [29], моделювання дозволяє розв’язати чотири задачі:

- візуалізувати систему в її поточному або бажаному для замовника стані;
- описати структуру або поведінку системи;
- одержати шаблон, що дозволяє сконструювати систему;
- документувати прийняті рішення, використовуючи отримані моделі.

Таким чином, моделювання складних систем дозволяє спростити їх сприйняття як для замовника, так і для розробника. Попереднє моделювання елементів системи значно прискорює процес розробки та дозволяє заздалегідь уникнути критичних помилок, що можуть несподівано виникнути під час її розробки. Для розробки подібних шаблонів використовується уніфікована мова моделювання (UML).

За матеріалами цього ж курсу, UML — це стандартний мова для моделювання та проєктування артефактів програмних систем. Використовується для візуалізації, специфікації, конструювання і документування артефактів програмного забезпечення. Приклади використання цієї мови будуть подані згодом в цьому розділі.

Для проєктування системи було використано інструмент під назвою IBM Rational Software Architect — середовище розробки та моделювання ПЗ, яке використовує UML для проєктування архітектур застосунків на різних об’єктно-орієнтованих мовах програмування [29].

2.1 Розробка моделі предметної області

Проєктування програмного комплексу складається з кількох ключових етапів. Спершу слід приділити увагу абстрактному рівню. На цьому етапі визначаються сутності системи, моделюється предметна область. Таким чином, до основних сутностей системи належать Peer та File.

Peer: вузол у мережі, який бере участь в обміні файлами та їх метаданими. Кожен “Peer” відповідає одному користувачу системи. Це основна сутність системи, що містить такі атрибути: ім’я вузла, унікальний криптографічний публічний ключ, рівень довіреності до вузла (довірений, не довірений) та час останньої поміченої активності в мережі.

File: дані про файл, яким обмінюються вузли. Одна сутність відповідає одному файлу на пристрої. Файли поділяються на ті, що поширюються від імені власника в мережі (мої файли) та отримані файли від інших вузлів. Визначено такі атрибути: унікальний ідентифікатор, назва файлу, розмір, дата останнього редагування, локальний шлях до файлу на пристрої користувача. У випадку, якщо файл отримано від іншого вузла, також зберігається криптографічний публічний ключ відправника.

Окрім зазначених сутностей також визначено також деякі допоміжні структури, що не є ключовими сутностями предметної області, але будуть надалі використовуватись при проєктуванні системи. Основною допоміжною структурою є PeerInfo, як технічна інформація про вузол у мережі. Вона зберігає ім’я, IP-адресу та порт, за яким “Peer” очікує на вхідні з’єднання від інших вузлів, час останньої активності вузла та криптографічний публічний ключ для ідентифікації. Також визначено інші структури, що використовуються для інформування користувача про перебіг подій у системі, та для опрацювання запитів від інших користувачів.

2.2 Розробка бізнес моделі

Наступним кроком є розробка бізнес моделі, яка детально описує функціональні можливості системи. Таким чином, для побудови цієї моделі слід визначити акторів системи та їх можливості. Для цього завдання було використано UML-діаграму варіантів використання.

Діаграма варіантів використання (ВВ) — UML-діаграма, що демонструє набір варіантів використання системи і діючих акторів, та зв’язки між ними [29].

Таким чином, було визначено трьох діючих акторів:

- Sender: відправник, основною дією є відправка файлів та обробка пов’язаних запитів.
- Receiver: отримувач, володіє можливістю перегляду доступних вузлів, надсилання запиту на доступ до метаданих файлів та обробляти запити на отримання файлів від інших вузів.
- User: користувач, якому належать спільні дії, доступні як для відправника, так і для отримувача.

Виходячи з того, що застосунок використовує однорангову архітектуру, тобто, кожен клієнт одночасно є й сервером, було прийнято рішення відокремити варіанти використання системи, що не залежать від сценарію її роботи (відправлення та отримання файлів) та пов’язати їх з незалежним актором User.

Оскільки за підсумками проєктування, діаграма ВВ вийшла досить об’ємна, було прийнято рішення подати її у вигляді трьох рисунків, що відображають кожного окремого актора, дотичні до нього варіанти використання та зв’язки. Отже, на рисунку 6 зображено фрагмент діаграми ВВ для актора Sender.

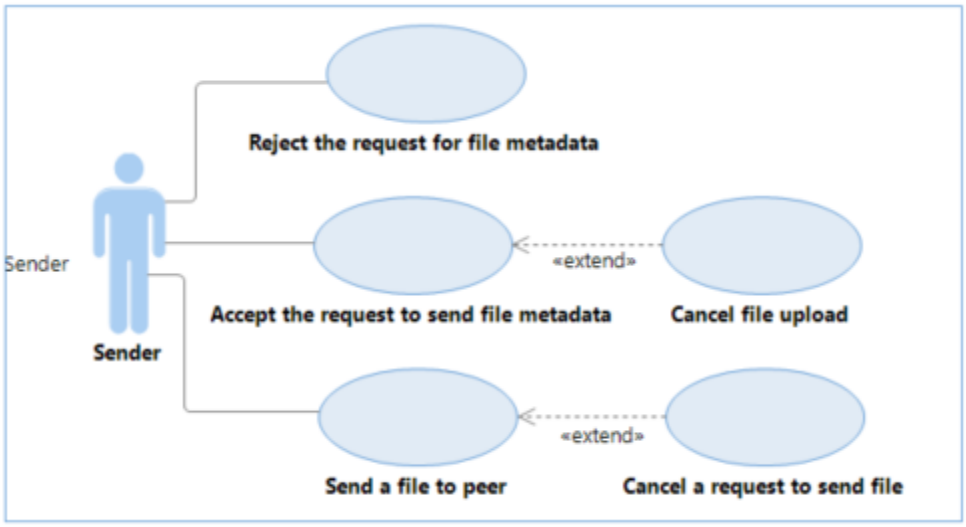

Рисунок 6. Діаграма варіантів використання для актора Sender

Даний актор є найбільш обмежений у діях, відносно інших. До його можливостей входить ініціація надсилання будь-якого локального файлу іншому вузлу та обробка запиту на поширення метаданих власних файлів доданих у систему. На рисунку 7 зображено фрагмент діаграми ВВ для актора Receiver.

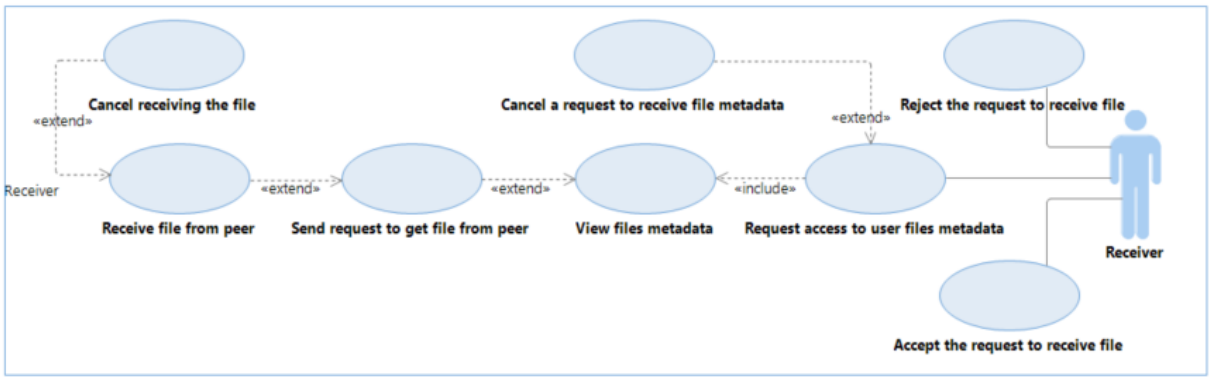

Рисунок 7. Діаграма варіантів використання для актора Receiver

Цей актор володіє можливостями, пов’язаними з отримуванням файлів та метаданих файлів інших вузлів. Послідовність наступна: надсилання запиту на отримання метаданих файлів вузла, після підтвердження запиту перегляд метаданих та за бажанням надсилання запиту на отримання певного файлу. Кожен запит супроводжується можливістю його скасування. Також Receiver здатен

обробити вхідний запит на отримання файлу від іншого вузла. На рисунку 8 зображено фрагмент діаграми ВВ для актора User.

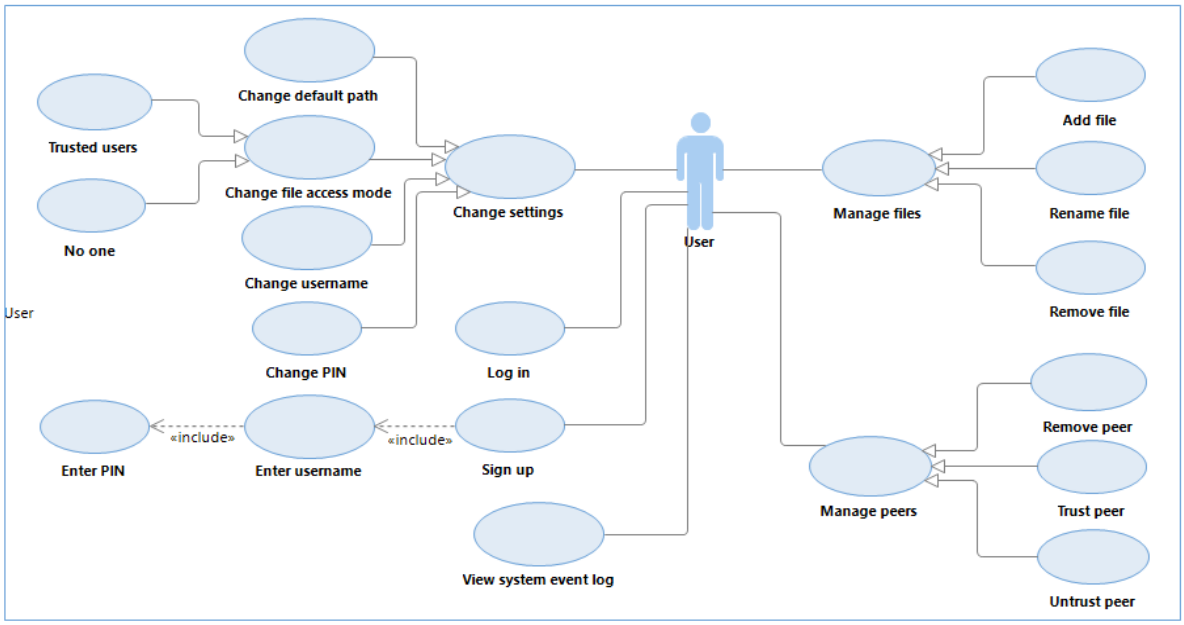

Рисунок 8. Діаграма варіантів використання для актора User

Як згадувалось раніше, User є актором, якому належать спільні варіанти використання для Sender та Receiver. Серед основних його можливостей, реєстрація та авторизація у систему, зміна налаштувань застосунку, управління вузлами та доданими у систему файлами, перегляд журналу системних подій.

Таким чином було завершено процес створення бізнес моделі P2P-системи для обміну файлами в локальній мережі.

2.3 Проєктування архітектури

Останнім і найважливішим етапом перед розробкою є безпосереднє проєктування архітектури, тобто остаточний перехід від абстрактної моделі до програмної. Оскільки система, що розробляється є досить складною та комплексною, було прийнято рішення розділити її на підсистеми. Це забезпечить

структурованість та гнучкість архітектури програмного рішення. В таблиці 1 наведено перелік визначених підсистем та їх опис.

Таблиця 1 – Підсистеми застосунку

№	Підсистема	Опис
1	Core	Центральний елемент управління всією системою. Координує роботу інших підсистем, надаючи зручний інтерфейс взаємодії для користувацького інтерфейсу та інтерфейсу командного рядка.
2	Network	Відповідає за мережеву взаємодію між вузлами. Основними завданнями є встановлення та керування P2P з’єднаннями, обмін повідомленнями та файлами.
3	FileStorage	Забезпечує контроль над локальною файловою системою застосунку, що складається із спільних та отриманих від інших вузлів файлів.
4	Database	Відповідає за збереження та управління даними про учасників мережі та локальні файли системи у базі даних.
5	Security	Забезпечує управління ключами, шифрування та хешування для безпечного обміну файлами.
6	Infrastructure	Надає доступ до інфраструктури застосунку, зокрема до системи налаштувань, логування та обробки подій.
7	Tests	Забезпечує автоматизоване тестування компонентів системи. Включає модульні та інтеграційні тести.
8	CLI	Точка входу в програму у вигляді інтерфейсу командного рядка. Забезпечує зручний спосіб взаємодії із системою для технічних користувачів.
9	UI	Графічний інтерфейс користувача, основна точка входу для взаємодії з системою. Забезпечує інтуїтивний доступ до функцій системи.

Для кращого сприйняття структури системи було створено UML-діаграму підсистем. Однак, незважаючи на те, що до переліку доступних UML-елементів входить «subsystem», технічно, такого поняття як діаграма підсистем не існує. Тому, наступна діаграма була реалізована на основі UML-діаграми компонентів.

За визначенням, отриманим з офіційної документації IBM, діаграма компонентів призначена для відображення архітектурної структури програмної системи на високому рівні, через представлення її компонентів, їхніх інтерфейсів та зв’язків між ними [30].

Таким чином, на рисунку 9 зображено діаграму підсистем, побудовану на основі діаграми компонентів.

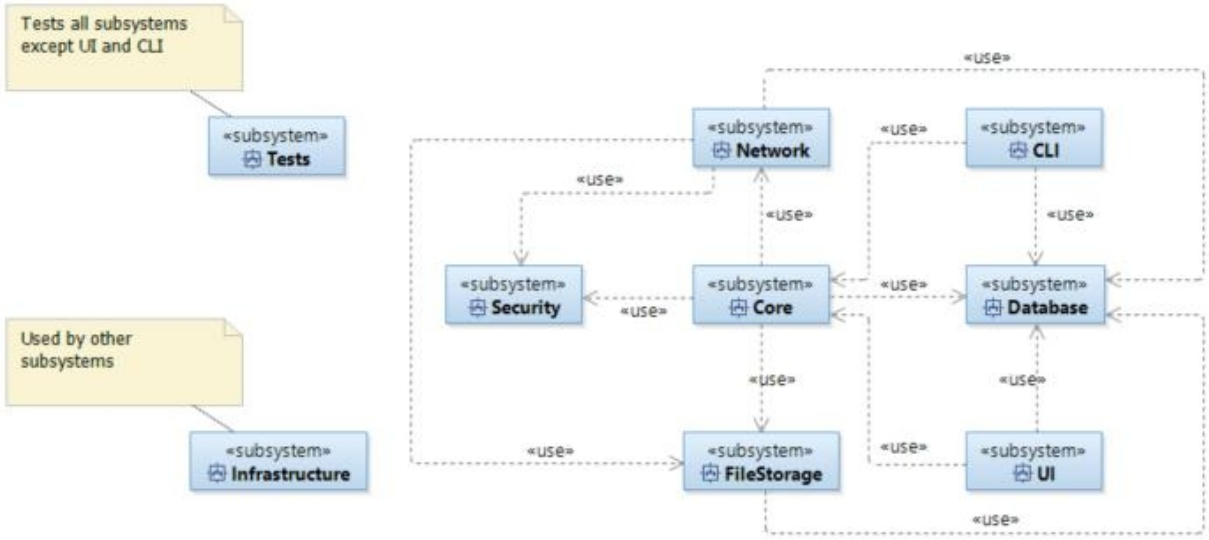

Рисунок 9. Діаграма підсистем P2P-системи для обміну файлами

Відповідно до наведеної діаграми, основна підсистема Core використовує функціонал більшості інших частин системи. База даних використовується мережевою та файловою підсистемами відповідно, до того ж, частина, пов’язана з передачею файлів у мережі потребує доступу до файлової системи та модуля безпеки застосунку. Обидва інтерфейси користувача взаємодіють із центральним елементом системи та базою даних.

Окремо на діаграмі виділено модулі інфраструктури та тестування. Інфраструктура використовується всіма іншими модулями системи, зокрема для

логування та обробки системних подій. Доступ до системи налаштувань надається виключно підсистемам Core та Tests. Остання підсистема Tests користується переважною більшістю наявних функціональних можливостей програмного рішення для модульного та інтеграційного тестування.

Наступним етапом у проєктуванні програмного рішення є визначення основних класів системи та визначення їх зв’язків. Для цього створюється відповідна UML-діаграма, а саме діаграма ієрархії класів.

Діаграма ієрархії класів — тип UML-діаграми, що демонструє набір класів, інтерфейсів, кооперацій та зв’язки між ними [29].

Дана архітектурна модель застосунку була апробована на VIII міжнародній студентській науково-технічній конференції (див. додаток А), тому в цій кваліфікаційній роботі вона використовується без змін. На рисунку 10 зображено зовнішній вигляд загальної UML-діаграми класів P2P-системи для децентралізованого обміну файлами.

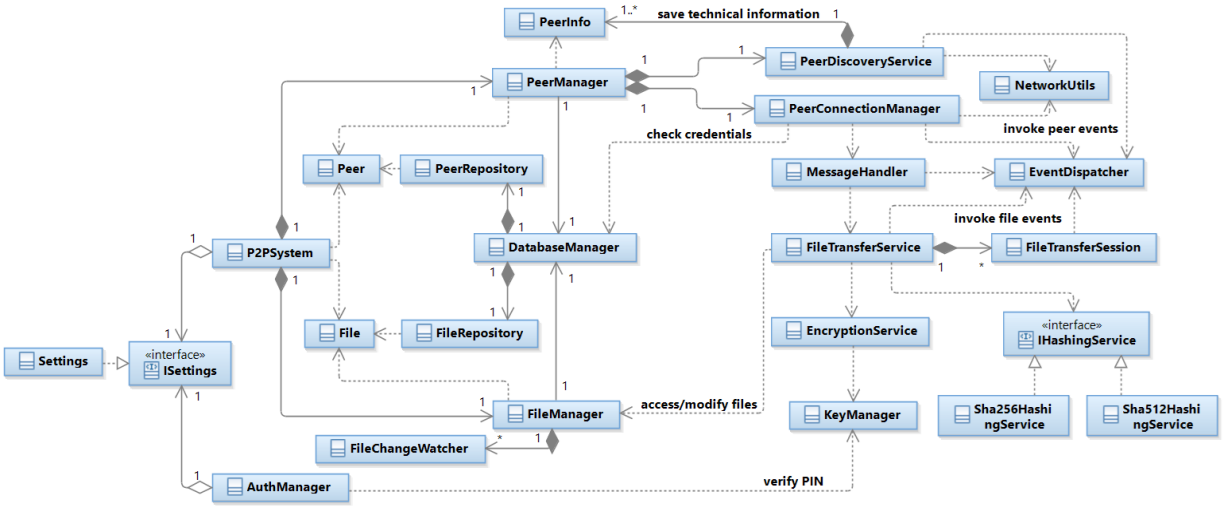

Рисунок 10. Діаграма ієрархії класів P2P-системи для обміну файлами

Діаграма відображає практично всі існуючі класи, інтерфейси та структури системи, їх зв’язки. Однак, вона демонструє виключно бекенд-структуру програмної системи та не включає елементи підсистем CLI та UI.

Як було зазначено раніше, Core є центральним елементом управління системою. Отже, до цієї підсистеми належать класи P2PSystem та AuthManager. Перший з них застосовує патерн проєктування «Фасад», цей патерн буде детально описано згодом. P2PSystem містить основні методи, які забезпечують контроль над усією системою, окрім автентифікації. Обов’язковим аргументом конструктора цього класу є об’єкт інтерфейсу ISettings, що, відповідає за налаштування системи..

AuthManager відповідає за автентифікацію користувача. Для цього використовує клас KeyManager (який буде описано згодом). Конструктор класу AuthManager приймає той же об’єкт інтерфейсу ISettings, що й P2PSystem. Успішна авторизація встановить публічний криптографічний ключ у налаштування. Наявність публічного ключа, збереженого в об’єкті цього класу дозволяє запустити систему (через клас P2PSystem). За його відсутності, буде повернуто помилку, система не розпочне роботу. Таким чином, ці два класи реалізують єдину точку входу в систему.

Наступними елементами управління системою є три ключові менеджери: PeerManager, FileManager і DatabaseManager. Перші два класи реалізують логіку взаємодії з базою даних, яка, як згадувалось раніше, керує даними про учасників мережі та локальними файлами системи. Окрім цього клас FileChangeWatcher виконує функцію спостерігача за файлами, доданими у систему, реагуючи на їхнє перейменування та видалення з пристрою. PeerManager виступає у ролі додаткового фасаду до всієї мережевої підсистеми, використовує структуру PeerInfo, що зберігає технічну інформацію про активні вузли.

Отже, PeerManager повністю керує життєвим циклом об’єкта сервісу PeerDiscoveryService та об’єкта ще одного менеджера — PeerConnectionManager. Сервіс відповідає за виявлення вузлів у мережі, а також за поширення інформації про локальний вузол серед інших учасників мережі. Використовує протокол UDP. Менеджер відповідає за з’єднання та P2P-взаємодію між вузлами. Використовує протокол TCP та статичний клас MessageHandler, призначений для обробки вхідних та вихідних повідомлень, переданих через TCP. Обидва вищезгадані

класи — PeerDiscoveryService та PeerConnectionManager — користуються методами статичного класу NetworkUtils, що відповідає за визначення IP-адреси, доступних портів та перевірку з’єднання з локальною мережею.

Наступні два класи — FileTransferService та FileTransferSession призначені для безпосередньої передачі файлів. Кожна передача файлу відбувається у окремій сесії. FileTransferService керує життєвим циклом кожної такої сесії. Використовує два інші сервіси, пов’язані з безпекою, що надають функціонал для шифрування та хешування файлів.

До компонентів, що реалізують безпековий функціонал належить клас KeyManager та два сервіси — EncrypionService та IHashingService. Ці класи належать підсистемі Security. Сервіси призначені для шифрування переданих файлів, з метою збереження конфіденційності, та хешування, для перевірки їхньої цілісності. KeyManager відповідає за роботу з криптографічними ключами, використовується для шифрування файлів та автентифікації користувача у системі.

Останнім елементом позначеним на діаграмі класів є EventDispatcher. Цей клас виступає у ролі диспетчера, що координує виклики подій пов’язаних з вузлами та передачею файлів. Працює це наступним чином: певні класи викликають події, пов’язані з поточним виконанням своїх завдань; класи графічного інтерфейсу підписуються на них та відповідним чином обробляють. Завдяки цьому, кінцевий користувач має можливість спостерігати за внутрішніми процесами системи в режимі реального часу та реагувати на них.

Для спрощення побудови архітектури різноманітних програмних систем існує низка шаблонів, що застосовуються при проєктуванні архітектури програмного комплексу. Їх називають шаблонами (патернами) проєктування.

Патерни проєктування описують способи вирішення поширених проблем, що виникають при проєктуванні систем. Їх розділяють на три групи: породжувальні (створення об’єктів), структурні (побудова ієрархій класів) та поведінкові (взаємодія між об’єктами) [31].

При розробці архітектури P2P-системи для децентралізованого обміну файлами було використано низку патернів проєктування. Найбільш вагомі приклади використання супроводжуються UML-діаграмами.

Одним з найважливіших архітектурних шаблонів, що було використано при проєктуванні системи є «Фасад» — патерн проєктування, що надає простий інтерфейс для роботи зі складною внутрішньою системою [31].

Як згадувалось вище, цей шаблон було застосовано до класу P2PSystem. P2PSystem виступає як центральний компонент усієї внутрішньої логіки P2P-системи, окрім системи автентифікації. Він надає доступ до обмеженої кількості методів, що необхідні для взаємодії з ключовими компонентами бекенд-системи. Такий підхід забезпечує зручність взаємодії клієнтів із системою та полегшує її підтримку в майбутньому. Крім цього, варто зазначити, що клас PeerManager є додатковим фасадом до P2PSystem. Цей клас виконує функції центрального пункту управління всіма доступними вузлами, а також взаємодією між ними.

На рисунку 11 зображено діаграму архітектурного патерну «Фасад». На цій діаграмі продемонстровано лише деякі ключові методи пов’язаних з патерном класів.

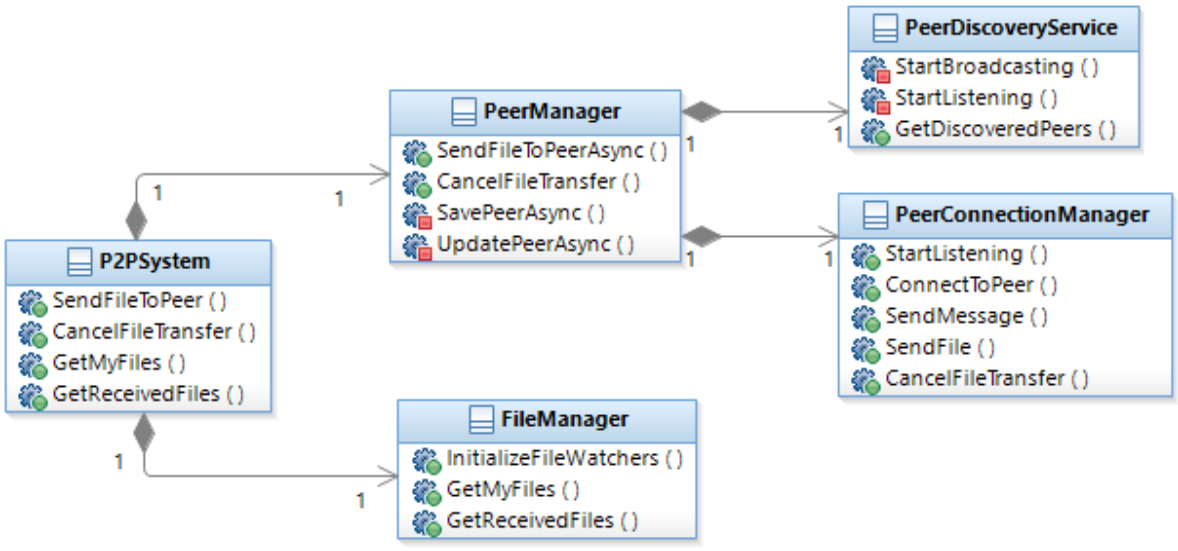

Рисунок 11. UML-діаграма ієрархії класів архітектурної моделі (шаблона «Фасад»)

Наступним ключовим патерном, застосованим у архітектурі цього проєкту, є «Спостерігач». Згідно з [31], «Спостерігач» — поведінковий патерн, що забезпечує механізм створення підписки. Це надає можливість об’єктам відстежувати події, що відбуваються в інших об’єктах.

В поточному рішенні, цей шаблон забезпечує взаємодію користувача та внутрішньої логіки застосунку, використовуючи події. Графічний та командний інтерфейс підписуються на події класу EventDispatcher та відповідним чином реагують на них. Інша класи, серед яких PeerDiscoveryService, MessageHandler, FileTransferSession та інші, «підіймають» події, одночасно передаючи певний набір аргументів. Для таких запитів створено різноманітні структури даних.

Цей патерн реалізовано в застосунку, зокрема, для своєчасного сповіщення про оновлення стану активних вузлів, сповіщення про стан перебігу передачі файлів між вузлами та іншого. Реалізовано також події, призначені для отримання відповіді від кінцевого користувача на запити доступу до передачі файлів або метаданих файлів. Таким чином, цей патерн створює надійний механізм, що надає можливість кінцевому користувачеві отримувати актуальні дані про стан системи та взаємодіяти з нею.

На рисунку 12 зображено діаграму архітектурного патерну «Спостерігач». На цій діаграмі подано лише частину класів, з обмеженою кількістю полів та методів, що стосуються цього поведінкового патерну.

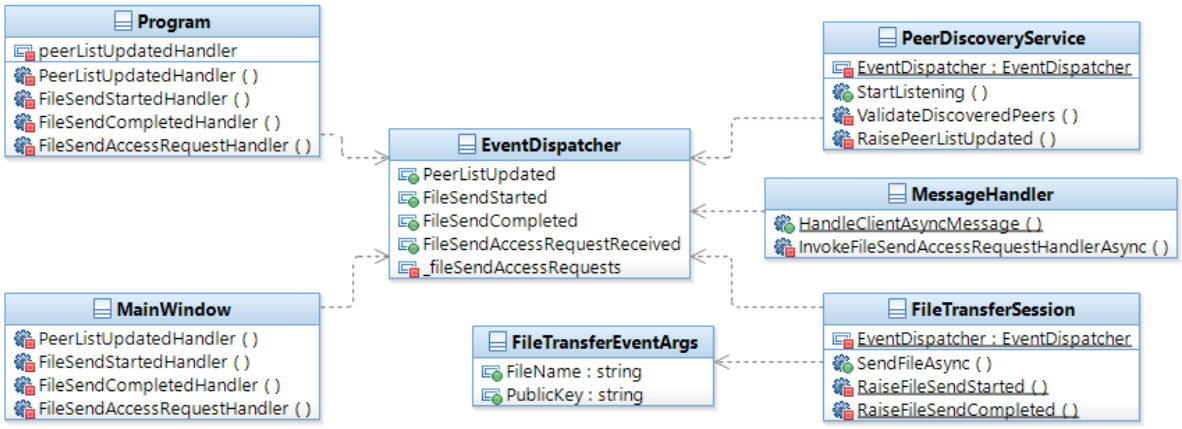

Рисунок 12. UML-діаграма ієрархії класів архітектурної моделі (шаблона «Спостерігач»)

Ще одним важливим патерном проєктування у даній реалізації є «Одинак». «Одинак» — це подорожувальний патерн, що забезпечує існування єдиного екземпляру класу та точки доступу до нього [31].

Цей шаблон проєктування застосовується до кількох класів. До таких класів належать DatabaseManager, FileManager, KeyManager та Logger. Таким чином, менеджер вузлів та менеджер файлів отримують доступ до єдиної бази даних, FileTransferService отримує доступ до менеджера файлів, а об’єкт класу KeyManager використовується в EncryptionService. Для забезпечення ізоляції під час тестування, в класі Logger, окрім звичайного екземпляра, реалізовано можливість створення тестового екземпляра класу.

Окрім описаних вище патернів, також було використано спрощену версію архітектурного патерну «Фабричний метод». Цей патерн проєктування не відіграє значну роль у роботі системи, а саме, надає вибір алгоритму хешування за допомогою інтерфейсу IHashingService та його реалізацій. Вибір алгоритму відбувається безпосередньо у класі FileTransferService безпосередньо перед використанням даного сервісу. На етапі розробки системи це архітектурне рішення планувалося як реалізація патерну «Стратегія». Однак, згодом, було виявлено що патерн не відповідає класичному визначенню та структурі цього патерну. Незважаючи на це, механізм вибору алгоритму хешування залишається доступною в цій версії застосунку.

Останнім кроком перед на етапі проєктуванні системи є проєктування діаграм послідовностей. Діаграми послідовностей призначена для демонстрації взаємодії об’єктів у послідовному порядку [32].

Таким чином, було прийнято рішення спроєктувати діаграму послідовності для відображення процесу надсилання файлу. Оскільки цей процес включає велику кількість взаємодій між об’єктами класів, діаграму було розділено на дві окремі діаграми послідовності.

Перша з них стосується етапу початкової ініціалізації. Підготовка до надсилання файлу супроводжується створенням прямого TCP з’єднання з вузлом, якщо на даний момент вільних активних з’єднань немає. Для цього менеджер

вузлів через сервіс виявлення вузлів отримує інформацію про активну IP-адресу і порт отримувача. Менеджер з'єднань ініціює з'єднання з вузлом, та одразу надсилає дані про власну IP-адресу та порт. Після цього здійснюється відправка запиту на передачу файлу. Цей запит містить інформацію про місце розташування файлу, публічний ключ відправника та отримувача. Обробник повідомлень очікує відповідь на запит. У випадку згоди отримувача, система переходить до процесу надсилання файлу. На рисунку 13 зображено діаграму послідовностей, що демонструє процес ініціалізації надсилання файлу.

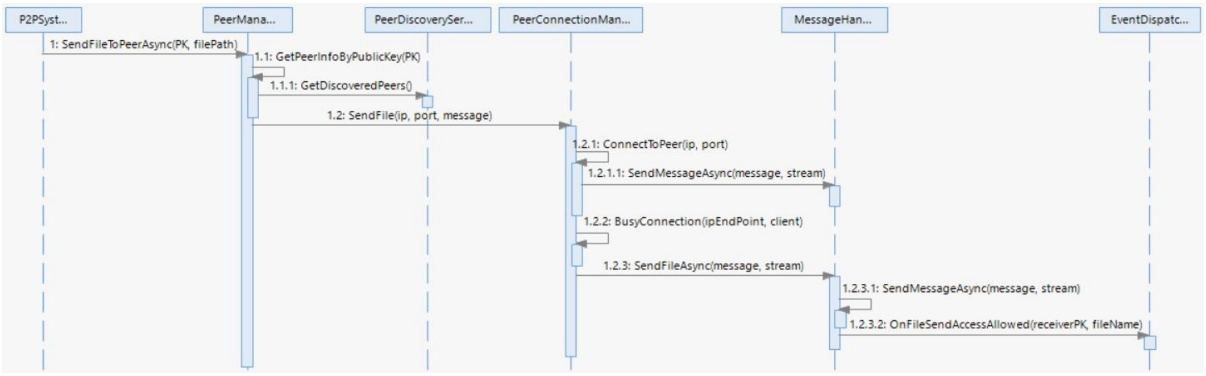

Рисунок 13. Діаграма послідовностей «Ініціалізація надсилання файлу»

Друга діаграма відображає безпосередній процес передачі файлу. Він складається з кількох етапів. Спершу відбувається підготовка файлу до передачі, а саме створення тимчасової копії файлу в поточному стані. Після цього обирається алгоритм хешування файлу. В даній реалізації, у більшості випадків обирається алгоритм SHA-512. Якщо архітектура поточної системи є 32-бітною, а розмір файлу перевищує 100 МБ, хешування виконується за алгоритмом SHA-256.

Наступним кроком є шифрування файлу за допомогою публічного ключа отримувача. Після завершення підготовки файлу, за відсутності вільної активної сесії, створюється нова сесія передачі файлу. Після цього відбувається процес передачі файлу, що супроводжується попереднім повідомленням, що містить технічну інформацію, зокрема публічний ключ відправника, дані про файл, хеш оригінального файлу та зашифрований AES-ключ. Більшість етапів у цьому ланцюжку супроводжуються викликами подій класу EventDispatcher, які

інформують користувача про перебіг передачі файлу. Для уникнення надмірної деталізації на діаграмі показано лише події початку та завершення процесу безпосереднього надсилання файлу.

На рисунку 14 зображено діаграму послідовностей, що демонструє процес надсилання файлу.

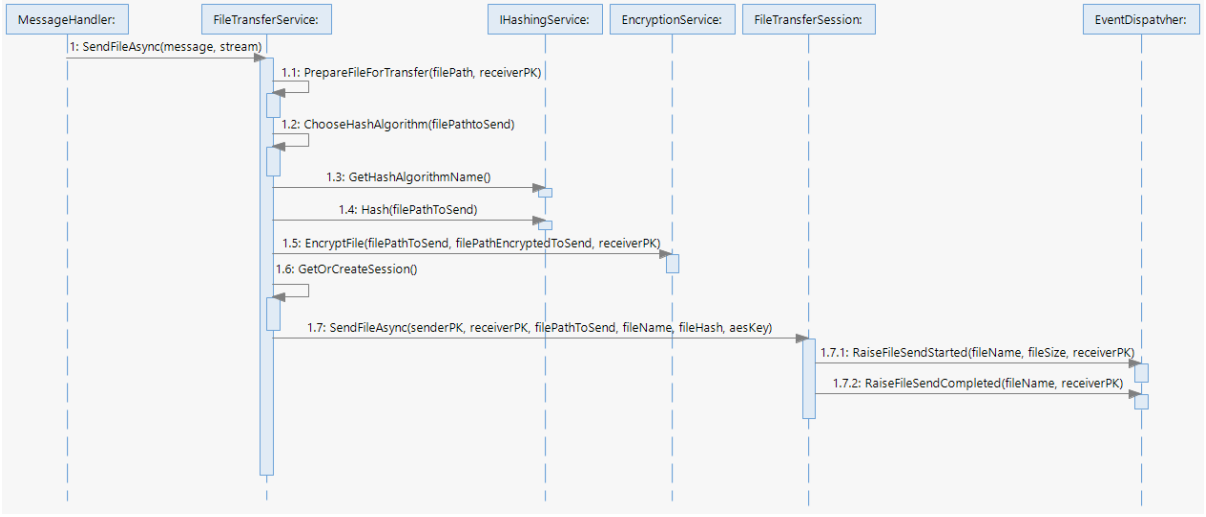

Рисунок 14. Діаграма послідовностей «Надсилання файлу»

Отже, на цьому завершено етап проєктування системи. У процесі розробки архітектура системи зазнавала деяких змін, незважаючи на це у записці наведено її остаточний варіант, актуальний для поточної версії застосунку.

РОЗДІЛ 3. КОНСТРУЮВАННЯ P2P-СИСТЕМИ ДЛЯ ОБМІНУ ФАЙЛАМИ

Конструювання програмної системи є одним з найважливіших етапів життєвого циклу розробки програмного забезпечення. Цей процес складається з багатьох завдань.

До основних завдань конструювання належить:

- Проектування та написання класів
- Вибір структур, узгодження формату імен змінних та констант
- Блокове та інтеграційне тестування, відладка коду
- Форматування та коментування програмного коду
- Інтеграція програмних компонентів
- Оптимізація програмної реалізації [33]

Підготовка до етапу конструювання включає вибір інструментів розробки, до яких належить мова програмування, платформа, середовище розробки, інструмент управління версіями та інші. Також варто визначити яка частина застосунку буде розроблена заздалегідь, а яка під час написання коду; визначити специфічні методики кодування, що стосуються архітектури програмного комплексу; організувати спільну роботу в команді [33].

Деякі з цих елементів підготовки, зокрема вибір технологій та інструментів розробки, були розглянуті в розділі 1 – «Огляд предметної області». Тому, в цьому розділі кваліфікаційної роботи буде зосереджено увагу на процесі кодування програмної системи, проведені тестування, а також буде здійснено верифікацію вимог до системи.

3.1 Реалізація ключових класів

Поточна реалізація системи складається з великого набору класів, кожен з яких втілює логіку різноманітної складності. Від тривіальної реалізації бази даних до складної організації процесу шифрування та передачі файлів. Отже, надалі буде подано опис найбільш цінних, з точки зору реалізації програмного рішення, елементів програмного коду.

Сервіс виявлення вузлів PeerDiscoveryService відіграє одну з найважливіших ролей в системі. Його основним завданням є розпізнавання вузлів та поширення інформації про власний вузол. Для цього в архітектурі програмного рішення передбачено клас PeerInfo, що містить технічну інформацію про вузли. Його структура зображена на рисунку 15.

```
public string Name { get; set; }
public string Ip { get; set; }
public int Port { get; set; }
public DateTime LastSeen { get; set; }
public string PublicKey { get; set; }
```

Рисунок 15. Поля класу PeerInfo

Таким чином, в ньому зберігається інформація про ім’я вузла, IP, порт, час останньої активності та його публічний ключ. Всі ці дані, за винятком часу останньої активності поширюються у мережі від імені вузла. Інформація про поточний вузол зберігаються у змінній _myPeerInfo.

Клас PeerDiscoveryService працює працює за протоколом Multicast, використовує IP-адресу "239.255.10.10" або "239.255.11.11" (якщо це тестовий об’єкт класу) та порт 11000. Для поширення даних про вузол використовується приватний метод StartBroadcasting, що працює в окремому потоці. Цей метод надсилає повідомлення у мережу кожні півтори секунди, що забезпечує актуальність інформації про активні вузли. Також перед відправкою повідомлення

здійснюється перевірка наявності підключення до локальної мережі. Фрагмент коду даного методу подано на рисунку 16.

```
while (!_cts.Token.IsCancellationRequested)
{
    try
    {
        await HandleNetworkConnectionAsync();

        var message :byte[] = System.Text.Encoding.UTF8.GetBytes(
            s: $"{_myPeerInfo.Ip}:{_myPeerInfo.Port}:{_myPeerInfo.PublicKey}:{_myPeerInfo.Name}");
        await _broadcastClient.SendAsync(message, message.Length,
            new IPEndPoint(_multicastAddress, DiscoveryPort)); // Task<int>
        await Task.Delay(1500, _cts.Token);
    }
    catch (Exception ex)
    {
        Logger.LogError("Failed to broadcast message.", ex);
        throw;
    }
}
```

Рисунок 16. Фрагмент коду методу StartBroadcasting

Сканування мережі на наявні вузли відбувається за допомогою методу StartListening. Цей метод також працює в окремому потоці та циклі. На кожній ітерації відбувається перевіряка на наявність з’єднання з локальною мережею, здійснюється валідація вузлів та очікується повідомлення у мережі. Під час валідації здійснюється фільтрування вузлів, які були активні більшу 5-ти секунд тому. Такі вузли видаляються зі списку, а всі наявні з’єднаннями з ними примусово завершуються. Після цього протягом 1,5 секунди система очікує на появу повідомлення у мережі. Якщо повідомлення виявлено, воно розбирається на частини, що містять інформацію про вузол, що його відправив. Якщо система розпізнає повідомлення від власного вузла, воно буде пропущене. На рисунку 17 подано перший фрагмент коду цього методу.

```
while (!_cts.Token.IsCancellationRequested)
{
    if (_lostConnection)
    {
        await Task.Delay(5000, _cts.Token);
        continue;
    }

    await Task.Delay(100, _cts.Token);
    ValidateDiscoveredPeers();

    var receiveTask = _discoveryServer.ReceiveAsync();

    if (await Task.WhenAny(receiveTask, Task.Delay(1500, _cts.Token)) == receiveTask)
    {
        var result = await receiveTask;
        var message = System.Text.Encoding.UTF8.GetString(result.Buffer);
        var parts :string[] = message.Split(':');

        if (parts.Length >= 4 && int.TryParse(parts[1], out int port))
        {
            var remoteIp :string = parts[0];
            var remoteId :string = parts[2];
            var remoteName = parts[3];

            // Skip our own broadcasts
            if (remoteId == _myPeerInfo.PublicKey)
                continue;
```

Рисунок 17. Фрагмент коду методу StartListening (1)

Якщо виявлений вузол є відмінним від власного, виконується перевірка на співпадіння порту, відкритого на нові TCP з’єднання. За потреби, система вирішує конфлікт портів (здебільшого, це корисно під час інтеграційного тестування). Після цього здійснюється перевірка наявності вузла у локальному списку активних в мережі вузлів. Якщо такий вузол вже присутній у списку, система оновлює дані про нього та передає їх у менеджер вузлів через відповідну подію. Якщо вузол виявлено вперше, він додається у список та зберігається у базі даних (також через менеджер вузлів). У випадку появи нового вузла або зміни імені наявного вузла, ініціюється оновлення списку доступних вузлів на інтерфейсі користувача через диспетчер подій. На рисунку 18 подано другу частину фрагмента коду методу StartListening.

```
if (port == _myPeerInfo.Port)
{
    if (!portConflict)
    {
        portConflict = true;
        _myPeerInfo.Port = NetworkUtils.ChangeLocalPort();
        PortChanged?.Invoke( sender: this, _myPeerInfo.Port);
        portConflict = false;
    }
}

var existingPeer = _discoveredPeers.FirstOrDefault(d :PeerInfo => d.PublicKey == remoteId);
if (existingPeer != null)
{
    var usernameChanged :bool = existingPeer.Name != remoteName;
    existingPeer.Name = remoteName;
    existingPeer.LastSeen = DateTime.Now;
    existingPeer.Port = port;
    PeerDiscovered?.Invoke( sender: this, existingPeer);
    if (usernameChanged)
        _eventDispatcher.OnPeerListUpdated( e: EventArgs.Empty);
}
else
{
    var peer = new PeerInfo(remoteIp, port, remoteName,  publicKey: remoteId, DateTime.Now);
    _discoveredPeers.Add(peer);
    NewPeerDiscovered?.Invoke( sender: this, peer);
    _eventDispatcher.OnPeerListUpdated( e: EventArgs.Empty);
}
```

Рисунок 18. Фрагмент коду методу StartListening (2)

Для керування з’єднаннями з вузлами створено клас PeerConnectionManager. Для менеджменту TCP з’єднань створено словники _activeConnections та _activeListeners, що зберігають інформацію про IP-адресу та порт клієнта, статус з’єднання, а також об’єкт класу TcpClient, що містить інформацію про з’єднання. В ньому також реалізовано однойменний метод StartListening. Принцип його дії полягає у очікуванні на вхідне з’єднання та обробки повідомлення іцініалізації, що відповідно містить дані про IP-адресу та порт клієнта. Якщо повідомлення проходить валідацію, з’єднання зберігається у словник та запускається процес прослуховування цього клієнта в окремому потоці за допомогою методу ListeningMonitor. На рисунку 19 зображено фрагмент коду цього методу.

```
var client = await _listener.AcceptTcpClientAsync(_cts.Token);
var stream = client.GetStream();

var ipEndPoint = await MessageHandler.ProcessInitialMessage(stream);
if (ipEndPoint == null)
{
    Logger.LogWarning($"Failed to process initial message from " +
                      $"{client.Client.RemoteEndPoint}. Closing connection.");
    client.Close();
    continue;
}

AddListener(ipEndPoint, client);
Logger.LogInformation($"Accepted connection from {client.Client.RemoteEndPoint}.");
_ = ListeningMonitoring(ipEndPoint, client);
```

Рисунок 19. Фрагмент коду методу StartListening (PeerConnectionManager)

Метод моніторингу повідомлень (ListeningMonitor) очікує на будь-які вхідні повідомлення від клієнта, що обробляються у методі HandleClientAsyncMessage класу MessageHandler. Якщо метод повертає значення “false”, з’єднання для прослуховування, а також окреме з’єднання (за наявності) закриваються. На рисунку 20 подано лістинг коду цього методу.

```
private async Task ListeningMonitoring(IPEndPoint ipEndPoint, TcpClient client)
{
    var stream = client.GetStream();
    var continueListening = true;
    while (continueListening)
    {
        if (stream.DataAvailable)
        {
            continueListening = await MessageHandler.HandleClientAsyncMessage(stream, ipEndPoint);
        }
        else
        {
            await Task.Delay(100, _cts.Token);
        }
    }

    StopListening(ipEndPoint.Address.ToString(), ipEndPoint.Port);
    Disconnect(ipEndPoint.Address.ToString(), ipEndPoint.Port);
}
```

Рисунок 20. Лістинг коду методу ListeningMonitoring

Обробкою повідомлень займається окремий статичний клас MessageHandler. Для наочної демонстрації роботи системи, надалі буде розглянуто поетапний

процес отримання файлу від іншого користувача P2P-системи від моменту отримання відповідного запиту. Процес ініціалізації надсилання файлу було продемонстровано на діаграмі послідовності на рисунку 13. На ній подано сценарій, коли отримувач підтвердив відповідний запит. На рисунку 21 зображено фрагмент коду, що містить обробку даного запиту від користувача. Для обробки використовується делегат та диспетчер подій, за допомогою якого запит передається на інтерфейс користувача для отримання відповіді.

```
else if (message.StartsWith("SEND_FILE"))
{
    var fileName = message.Split('|')[1];
    Logger.LogInformation($"Receiving file: {fileName} from client {ipEndPoint}");
    if (!_testMode)
    {
        var result :KeyValuePair<string,bool>  = await VerifyUser(ipEndPoint);
        bool accessResult = await EventDispatcher.GetInstance().InvokeFileReceiveAccessRequestHandlerAsync(result.Key, fileName);
        if (!accessResult)
        {
            var response :string  = message.Replace("SEND_FILE", "ACCESS_TO_SEND_FILE_DENIED");
            await SendMessageAsync(response, stream);
            Logger.LogInformation($"Access denied for file: {fileName} transfer request from client {ipEndPoint}");
            return false;
        }
    }
    await SendMessageAsync("ACCESS_TO_SEND_FILE_ALLOWED", stream);
}
```

Рисунок 21. Фрагмент коду методу HandleClientAsyncMessage

У диспетчері подій створюється завдання на отримання відповіді від користувача відповідно до отриманого запиту. Таким чином, обробка повідомлень у поточному TCP-з’єднанні буде призупинена до моменту отримання відповіді. Опрацювання запиту від делегата “VerifyUser” здійснюється класом PeerManager, що володіє доступом до бази даних. Проте, в архітектурі застосунку не передбачено пряме з’єднання між цими класами, тому ця обробка виконується через посередника PeerConnectionManager, що володіє власним делегатом з ідентичною назвою.

У наведеному фрагменті, на рисунку 21, делегат використовується виключно для отримання інформації про вузол, оскільки отримання файлу від вузла завжди вимагає підтвердження, незалежно від статусу відправника (довірений чи ні).

Процес отримання файлу розпочинається після надходження відповідного повідомлення. Формат цього повідомлення має наступну структуру: “FILE

{senderPublicKey}|{fileName}|{fileSize}|{fileHash}|{aesKey}\n”. Це повідомлення обробляє MessageHandler та викликає метод ReceiveFileAsync статичного класу FileTransferService.

Метод відповідає за ініціалізацію сесії прийому файлу та здійснює його обробку. Спершу виконується парсинг початкового повідомлення на окремі елементи. Далі, створюється нова або використовується наявна активна сесія, після чого отримується файл у зашифрованому вигляді.

За допомогою переданого ключа шифрування та наявного приватного криптографічного ключа здійснюється його дешифрування. Наступним кроком є хешування файлу. Назва алгоритму та оригінальний хеш отримуються з початкового повідомлення. Якщо після порівняння оригінальний хеш відмінний від поточного, процес передачі, розшифрування та хешування повторюється до трьох разів. У випадку співпадіння, система надсилає позитивну відповідь, файл записується у базу даних як отриманий файл. Поточна сесія залишається активною та може бути використана для наступних передач файлів.

Кожен з етапів процесу прийому файлу супроводжується викликом відповідної події, що інформує кінцевого користувача про поточний стан передачі. Повний лістинг коду даного методу подано у додатку Б.

Отже, залишилось детальніше розглянути реалізацію системи безпеки у застосунку. Як згадувалось вище, для розшифрування файлу використовується приватний криптографічний ключ та переданий AES-ключ шифрування. Відправник шифрує цей AES-ключ публічним ключем отримувача за допомогою методу шифрування RSA. Таким чином, лише за наявності відповідного приватного криптографічного ключа можливо отримати оригінальний AES-ключ.

На рисунку 22 зображено фрагмент коду методу DecryptFile класу EncryptionService.

```
privateKey ??= KeyManager.GetInstance().GetPrivateKey();
var privateAesKeyBytes :byte[] = DecryptKey(aesKey, privateKey);

var decryptPath = Path.ChangeExtension(filePath, extension: ".decrypted");

using var inputStream = new FileStream(filePath, FileMode.Open, FileAccess.Read,
    FileShare.Read, bufferSize: 4096, FileOptions.SequentialScan);
using var outputStream = new FileStream(decryptPath, FileMode.Create, FileAccess.Write,
    FileShare.None, bufferSize: 4096, FileOptions.SequentialScan);

// Read IV
var iv = new byte[16];
inputStream.ReadExactly(iv, offset: 0, count: iv.Length);

// Create an Aes object
// with the specified key and IV.
using var aes = Aes.Create();
aes.Key = privateAesKeyBytes;
aes.IV = iv;

// Create a decryptor to perform the stream transform.
var decryptor :ICryptoTransform = aes.CreateDecryptor(aes.Key, aes.IV);

// Calculate buffer size based on file size
var fileSize :long = inputStream.Length;
var bufferSize = (int)Math.Min(Math.Max(fileSize / 1000, 8 * 1024), 1 * 1024 * 1024);

// Create the streams used for decryption.
using var csDecrypt = new CryptoStream(inputStream, decryptor, CryptoStreamMode.Read);
csDecrypt.CopyTo(outputStream, bufferSize);
return decryptPath;
```

Рисунок 22. Фрагмент коду методу DecryptFile

Реалізація цього методу є модифікованим (з допомогою GitHub Copilot) варіантом прикладу використання AES, наведеного в офіційний документації Microsoft Docs [34].

В цьому методі здійснюється розшифрування AES-ключа приватним криптографічним ключем поточного користувача, Створюються файлові потоки для читання поточного та створення нового (розшифрованого) файлу. Перші 16 байт зашифрованого файлу містить унікальний вектор ініціалізації, що використовується алгоритмом шифрування відповідно до зазначеної вище документації. Після цього на основі ключа та вектора створюється декриптор та обчислюється розмір буфера, що залежить від розміру файлу. Як результат, метод повертає шлях до розшифрованого файлу на пристрої.

Останнім методом, що буде розглянуто в рамках цієї роботи є метод шифрування приватного криптографічного ключа. Оскільки цей ключ є найважливішим елементом безпеки застосунку, йому потрібно забезпечити відповідний рівень захисту. На рисунку 23 зображено фрагмент коду методу для шифрування ключа за допомогою PIN-коду — EncryptKey.

```
var salt :byte[] = GenerateSalt();
var key :byte[] = GetAesKey(salt, pin);

// Create an Aes object
// with the specified key and IV.
using var aes = Aes.Create();
aes.Key = key;
aes.GenerateIV();

// Create an encryptor to perform the stream transform.
var encryptor :ICryptoTransform = aes.CreateEncryptor(aes.Key, aes.IV);

// Create the streams used for encryption.
using var msEncrypt = new MemoryStream();
using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
{
    using (var bwEncrypt = new BinaryWriter(csEncrypt))
    {
        //Write all data to the stream.
        bwEncrypt.Write(privateKey);
    }
}

var ciphertext :byte[] = msEncrypt.ToArray();
return salt.Concat(aes.IV).Concat(ciphertext).ToArray();
```

Рисунок 23. Фрагмент коду методу EncryptKey

Цей метод також використовує AES-шифрування. Однак, в цьому випадку ключ шифрування створюється на основі PIN-коду та сольового хешу, що генерується випадковим чином. У результаті виконання методу формується байтовий рядок, що складається з сольового хешу, унікального вектору ініціалізації та безпосередньо зашифрованого ключа об’єднаних послідовно.

3.2 Розробка GUI

Як згадувалось у попередньому розділі, архітектура програмного рішення містить дві підсистеми графічного інтерфейсу. Оскільки система розроблялась поступово, починаючи з бекенд-частини, створення інтерфейсу командного рядка було необхідністю для повноцінного тестування роботи внутрішньої логіки застосунку.

В цьому підрозділі буде описано розробку саме графічного інтерфейсу користувача, оскільки графічний інтерфейс не лише значно підвищує зручність використання системи, але часто є невід'ємним елементом програмного забезпечення у межах кваліфікаційної роботи.

Поточна програмна реалізація фронтенд частини складається з кількох ключових компонентів. Головне вікно організовано на основі динамічного завантаження вмісту, містить меню навігації, список сповіщень та поточних сесій передач файлів. Основний вміст вікна завантажується відповідно до дій користувача через елемент ContentControl. Тож користувач має доступ до основних елементамів програми незалежно від поточного стану застосунку. Зовнішній вигляд домашньої сторінки подано на рисунку 24.

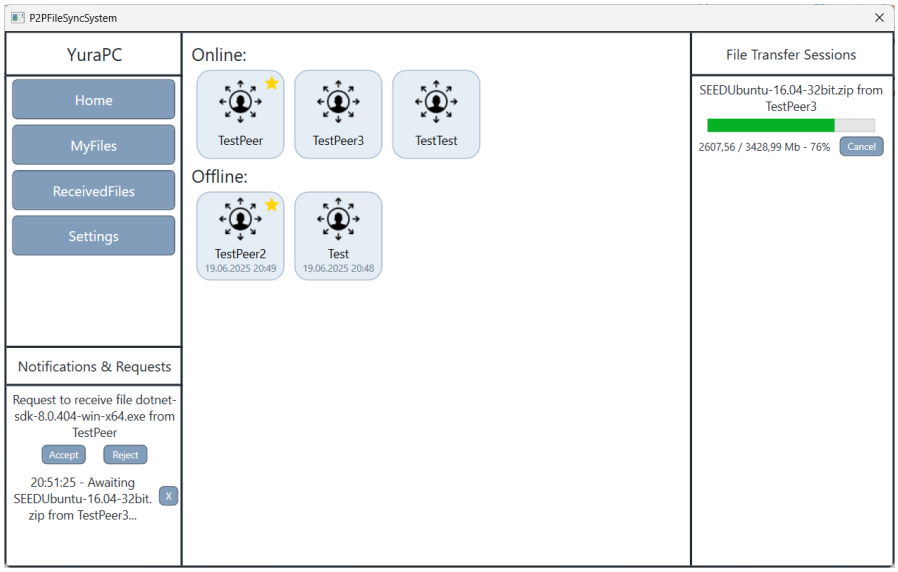

Рисунок 24. Зовнішній вигляд екрану HomeView

Як видно на рисунку 24, користувач має можливість перемикатися між різними екранами інтерфейсу, інтерактивно керувати сповіщеннями та запитами, а також скасовувати передачі файлів. Довірені вузли позначені відповідним символом, для вузлів, що мають статус “Offline” додатково відображається час останнього візиту.

Серед екранів, передбачених у застосунку є дім, мої файли, отримані файли та системні налаштування. Також окрім головного вікна передбачено вікно автентифікації, що у свою чергу теж наповнюється контентом з екрану реєстрації або авторизації. Реєстрація виконується у два етапи: введення імені користувача та введення PIN-коду для входу в застосунок.

На рисунку 25 подано зовнішній вигляд екрану авторизації.

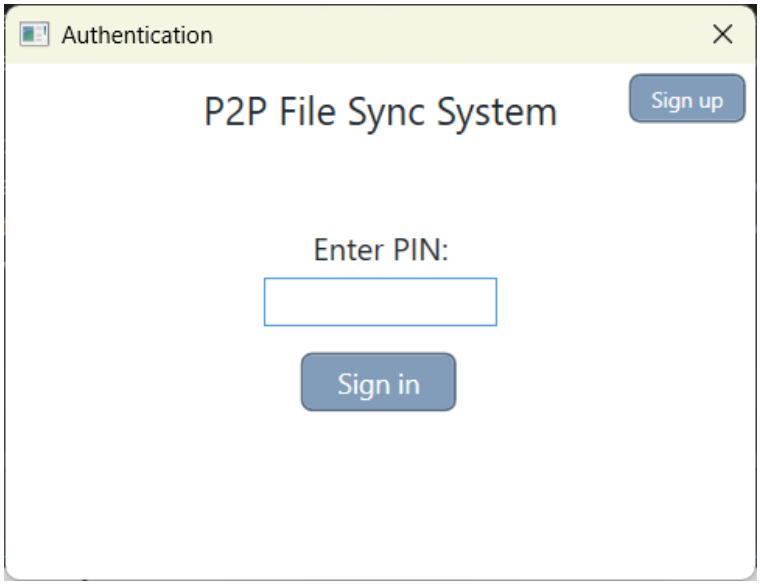

Рисунок 25. Зовнішній вигляд екрану LoginView

Таким чином, запуск програми супроводжується автентифікацією користувача, з можливістю перереєстрації за бажанням. Повторна реєстрація користувача зберігає дані про файли та вузли, та лише оновлює ключі безпеки.

На час написання записки до даної роботи підсистема графічного інтерфейсу користувача залишається на стадії розробки. Її завершення заплановано до моменту фінальної здачі кваліфікаційної роботи.

3.3 Тестування та верифікація вимог

Тестування є одним із завершальних етапів життєвого циклу розробки програмного забезпечення. Тестуванням називають процес перевірки коректності роботи системи, виявлення проблем та помилок. Тестування здійснюється в різних умовах, з використанням варіативної кількості та якості вхідних даних. Існують різні методи тестування, зокрема функціональне, навантажувальне, тестування безпеки та регресії [35].

В межах цієї роботи було проведено виключно функціональне тестування. Кожна ітерація (спринт) включала розробку відповідних модульних або інтеграційних тест-кейсів. Модульне тестування передбачає перевірку роботи окремих модулів системи, в той час як інтеграційне тестування має на меті перевірити взаємодію модулів [35].

Як згадувалось раніше, для тестування використовується фреймворк xUnit та бібліотека Moq. Фреймворк забезпечує зручну інфраструктуру для створення тестів, а бібліотека надає можливості імітації об’єктів, потрібних для роботи модуля. На рисунку 26 зображено «backlog» тест-кейсів програмного рішення.

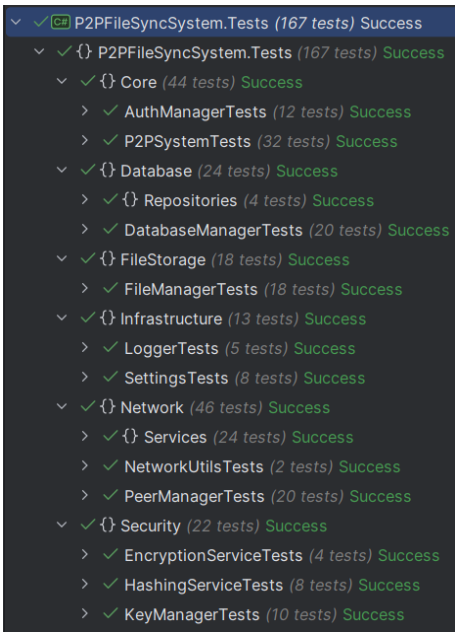

Рисунок 26. Backlog усіх тест-кейсів P2P-системи

Таким чином, основна частина бекенд логіки застосунку була охоплена різноманітними тестами. Для тестування у частині класів передбачено тестові режими, зокрема бази даних, файлового менеджера, логувальника, менеджера авторизації, та класів, пов’язаних із мережею.

Один з прикладів модульного тестового сценарію наведено на рисунку 27.

```
[Fact]
public async Task SendFileAsync_ValidFile()
{
    // Arrange
    const string fileName = "testFile.txt";
    var tempFilePath = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString(), fileName);
    Directory.CreateDirectory(Path.GetDirectoryName(tempFilePath)!);
    const string fileContent = "Test file content";
    await File.WriteAllTextAsync(tempFilePath, contents: fileContent);

    const string senderPublicKey = "testUser";
    const string receiverPublicKey = "testUser2";

    var message = $"FILE {senderPublicKey}|{receiverPublicKey}|{tempFilePath}";
    var stream = new MemoryStream();

    // Act
    await FileTransferService.SendFileAsync(message, stream);

    // Assert
    stream.Position = 0;
    var reader = new StreamReader(stream);
    var receivedMessageParts :string[] = (await reader.ReadToEndAsync()).Split('\n');

    Assert.Equal(fileContent, receivedMessageParts[1].Trim());

    // Cleanup
    if (File.Exists(tempFilePath))
        File.Delete(tempFilePath);
}
```

Рисунок 27. Лістинг тест-кейсу, що перевіряє метод надсилання файлу

Даний тестовий сценарій належить до класу FileTransferServiceTests, що тестує функціональні можливості сервісу, який повністю керує життєвим циклом сесій передачі файлів. Попередньо, в конструкторі цього класу ініціалізується необхідні компоненти у тестовому режимі, зокрема логувальник, файловий менеджер, директорія з файлами, що додаються у тестову файлову систему та підписка на подію.

У самому тесті створюються тимчасові вхідні дані, а також потік даних, що працює з оперативною пам’яттю (MemoryStream). Тест викликає метод SendFileAsync статичного класу FileTransferService із заданими параметрами, а саме форматованим за структурою стрічки повідомленням та потік, через який передаватиметься файл. Після завершення передачі у тесті перевіряється вміст потоку та порівнюється з вмістом початкового файлу.

Прикладом інтеграційного тесту тестовий сценарій на рисунку 28.

```
[Fact]
public async Task CancelFileTransferWithoutVerification_ShouldReturnTrue_WhenSuccessful()
{
    // Arrange
    _p2PSystem.Start();
    _p2PSystem2.Start();
    await Task.Delay(_delay * 2);
    await _p2PSystem2.TrustPeerAsync("testPublicKey");

    var filePath = Path.Combine(_testDirectory, "testLargeFile.txt");
    await File.WriteAllTextAsync(filePath, contents: new string( c: 'A', count: 50_000_000));

    // Act
    var result :bool  = await _p2PSystem.SendFileToPeerAsync("testPeerPublicKey", filePath);

    // Wait for the event to be triggered
    var cancelResult = false;
    if (result)
    {
        await Task.WhenAny(_fileSendStarted.Task, Task.Delay(_timeout * 5));
        cancelResult = await _p2PSystem.CancelFileTransfer( chosenPeerPublicKey: "testPeerPublicKey",  fileName: "testLargeFile.txt");
    }

    // Assert
    Assert.True(result);
    Assert.True(cancelResult);
    Assert.True(await Task.WhenAny(_fileSendCancelled.Task, Task.Delay(_timeout)) == _fileSendCancelled.Task);
    Assert.True(await Task.WhenAny(_fileReceiveCancelled.Task, Task.Delay(_timeout)) == _fileReceiveCancelled.Task);
}
```

Рисунок 28. Лістинг тест-кейсу, що перевіряє метод скасування передачі файлу

В цьому тестовому сценарії використовуються об’єкти основного класу підсистеми Core, що є фасадом до усього функціоналу системи, окрім автентифікації (необхідна для шифрування файлів під час передачі).

Отже, в цьому тестовому класі ініціалізується ряд об’єктів P2PSystem, що представляють повноцінних користувачів системи. Для їх ініціалізації створюються імітації об’єктів налаштувань за допомогою бібліотеки Moq. Крім цього, задаються тестові режими для статичних класів, створюються тимчасові директорії для тестів, завдання та підписки на події.

В ході тестового сценарію завантажуються дві системи, встановлюється рівень довіри, записується тестовий файл більшого розміру та здійснюється передача цього файлу. Після цього розпочинається очікування отримання сповіщення про початок надсилання файлу. Коли процес розпочато, відправник викликає метод, що скасовує передачу. Результат тесту визначається як значеннями, що повертають методи, так і фактом спрацювання подій, що сигналізують про скасування передачі файлу.

Отже, у результаті розробки проєкту реалізовано значну частину поставлених вимог (див. розділ 1.4).

Функціональна вимога щодо підключення пристроїв у локальній мережі повністю виконана та верифікована рядом модульних тестів та ручного тестування системи. Однак, тестування мережевої взаємодії застосунку, в локальній мережі на особистому обладнанні між ноутбуком та стаціонарним комп’ютером не дало очікуваних результатів, пристрої не розпізнаються. Тим не менш, тестування в іншому середовищі між двома ноутбуками дало позитивні результати. Підключення через глобальну мережу є однією з тих функцій, що закладає фундамент на майбутнє глобальне оновлення застосунку.

Додавання спільних файлів та папок реалізовано частково, обмежуючись виключно файлами. Перегляд метаданих доступних файлів та передача файлів реалізована у повній мірі, що підтверджено модульними та інтеграційними тестами, а також ручним тестуванням застосунку. Вимога щодо контролю доступу була дещо змінена. В процесі розробки було прийнято рішення обмежитись лише двома варіантами: надання доступу до метаданих лише обраним пристроям або нікому (без запиту). До того ж, рівень доступу визначається для усіх доданих спільних файлів одночасно. Вимога, що стосується логування дій виконана повністю, однак потребує вдосконалення.

Система більшою мірою задовольняє вимогу продуктивності, однак потребує подальшої оптимізації та докладнішого тестування. Робота системи перевірена здебільшого (але не виключно) на одній робочій машині, на якій розроблялась, з різними вхідними даними та умовами. Система забезпечує

надійний механізм передачі файлів, що включає перевірку цілісності та повторну передачу за потреби. Цей механізм був протестований модульним та ручним тестуванням. Опціональна вимога щодо відновлення перерваної передачі не була виконана через брак часу та складність реалізації.

Система відповідає усім визначеним вимогами, щодо безпеки. Користувач визначає рівень довіреності користувачів. Файли передаються у зашифрованому вигляді, користувач автентифікується за допомогою PIN-коду. Зручність системи забезпечена наявністю графічного інтерфейсу користувача.

Таким чином, було завершено етап конструювання та тестування P2P-системи для децентралізованого обміну файлами. Завершальний етап життєвого циклу розробки програмного забезпечення — розгортання та впровадження системи — не розглядається в цій роботі. На цьому розробка поточної версії проєкту вважається завершеною.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ранах

Кожна людина зазнавала у своєму житті поранення різного ступеня важкості. Зазвичай, рани «з’являються» випадковим чином: через необережність, недбалість, рідше як результат навмисних дій, спрямованих на нанесення фізичної шкоди людині. Тому, завжди слід бути готовим до надання першої допомоги в таких випадках. Спершу слід дати точне визначення що таке рана.

Відповідно до матеріалів лекцій з безпеки життєдіяльності та основи охорони праці, рана — механічне пошкодження цілісності судин шкіри, слизових оболонок чи органа тіла, яке супроводжуються болем і кровотечею; порушення цілісності шкіряних покривів, слизових оболонок, глибоких тканин і поверхні внутрішніх органів в результаті механічної або іншої дії [36].

Таким чином, можна зробити висновок, що існують різні види ран, кожен з яких вимагає відповідне медичне втручання.

Спершу слід звернути увагу на рани, які супроводжуються кровотечею. При кровотечі з такої рани, за можливості чітко її візуалізувати, необхідно здійснити максимально можливий тиск на рану руками та накласти пов’язку, що тисне. Загалом, за необхідності, накладається кровоспинний джгут, що повинен запинити кровотечу, однак, якщо накладання й другого кровоспинного джгута буде неефективним, слід здійснювати прямий тиск на рану до приїду персоналу медичної допомоги або виконати тампонування рани [37].

Однією з ознак при підозрі на травму голови є рани в області голови та обличчя. В такому випадку визначено наступні правила надання допомоги: на рану потрібно накласти марлеву пов’язку та зафіксувати її не створюючи надмірний тиск. Заборонено проводити пальпацію рани та тиснути в ній. Також не можна використовувати антисептики при обробці такої рани та вправляти кісткові уламки [37].

Якщо є ознаки відкритого перелому кісток кінцівок, зокрема, наявність рани в місці перелому, кровотеча та біль в області рани, порушення функцій ушкодженої кінцівки та інші, варто накласти стерильну, чисту пов’язку на рану. У випадку кровотечі з такої рани, слід діяти відповідно до порядку надання домедичної допомоги постраждалим при масивній зовнішній кровотечі [37].

Зважаючи на воєнний стан, що триває в Україні через повномасштабне вторгнення російської федерації, особливо важливим є вміння надання домедичної допомоги постраждалим в умовах бойових дій та воєнного стану. Таким чином, варто ознайомитись з основними положеннями, що регламентують порядок дій при травмах та пораненнях у вищезазначених умовах. Відповідно до порядку, затвердженого Міністерством охорони здоров’я від 9 березня 2022 року, визначено алгоритм дій в умовах прямої та непрямої загрози.

Що стосується дій з ранами та пораненнями, у випадку перебування в зоні прямої загрози, слід обмежитись наступними діями: гасіння вогню на тілі постраждалого; зупинка масивної зовнішньої кровотечі, прямим натиском на рану або використовуючи доступні кровоспинні джгути (у випадку виникнення труднощів з виявленням країв ран, джгут накладається якомога вище); якщо свідомість потерпілого порушена та немає змоги перейти в укриття, необхідно повернути постраждалого на живіт або стабільне бокове положення [37].

В зоні непрямої загрози алгоритм дій дещо відрізняється. Спершу варто оцінити наявність масивної кровотечі. У випадку виявлення такої, здійснити процедуру, що описана вище. Після цього слід оцінити свідомість постраждалого та прохідність дихальних шляхів. За необхідності, розпочати серцево-легеневу реанімацію, в іншому випадку, підтримувати прохідність дихальних шляхів. Для цього варто забезпечити стабільне бокове положення пораненого або здійснювати цей процес мануально (руками). Якщо виявлено масивні травми обличчя, слід оглянути грудну клітку, а також оглянути постраждалого з голови до ніг. При виявленні кровотечі, слід її зупинити, накладанням пов’язки на рани. Також положенням визначено застереження від застосування тиснучих пов’язок на око, тампонування чи надмірного натиску на рани голови, а також тампонування рани

грудної клітини чи животі. За можливості, слід викликати екстрену медичну допомогу і дотримуватись вказівок диспетчера [37].

4.2 Загальні вимоги безпеки до обладнання та технологічних процесів

Кожне підприємство володіє різноманітним обладнанням, від станків, верстатів, до різного типу устаткування. Усі типи обладнань розробляються відповідно до певних стандартів та безпекових вимог. Дотримання цих вимог забезпечує мінімальний ризик виникнення негативних наслідків для користувача устаткування.

Відповідно до наказу про затверджених вимог безпеки та захисту здоров'я під час використання виробничого обладнання працівниками визначено ряд мінімальних вимог до безпеки виробничого обладнання. Далі перелічено деякі з них. Пристрої керування виробничим обладнанням, що впливають на безпеку, мають бути чітко видимі, ідентифіковані та належним чином позначені. Виробниче обладнання облаштовується пристроєм аварійної зупинки залежно від небезпеки, пов’язаної з обладнанням. Якщо існує небезпека через механічний контакт із рухомими частинами виробничого обладнання, що може призвести до нещасних випадків, його частини обладнуються захисними огородженнями чи пристроями [38].

Також вказано вимоги до захисних пристроїв. Вони повинні бути міцними та не становити будь-якої додаткової небезпеки. Розташовуватись на достатній відстані від небезпечної зони, а також таким чином, що унеможливлює їх зняття або виведення з ладу. Захисні пристрої повинні не обмежувати спостереження за робочим циклом обладнання. Необхідно щоб пристрої були зручними для проведення операцій із встановлення або заміни частин виробничого обладнання та для технічного обслуговування, обмежуючи доступ тільки до тієї зони, в якій

має виконуватися робота, та (якщо можливо) без зняття їх захисних огороджень і пристроїв [38].

Крім вимог до самого обладнання також існують безпекові вимоги до технологічних процесів. Безпека робітника під час виробництва є найвищим пріоритетом підприємства. Більшість летальних випадків, та таких, що негативно впливають на здоров'я користувача устаткування трапляється саме через нехтування технікою безпеки під час виробничих процесів.

Безпечні умови праці (безпека праці) – стан умов праці, при яких вплив на працівника небезпечних і шкідливих виробничих факторів не перевищує гранично допустимих значень [39].

На даний момент немає діючого документу, що регламентує загальні вимоги до виробничих процесів. Однак, вищезгаданий наказ містить вимоги безпеки щодо використання виробничого обладнання, що частково стосуються поставленого питання. Далі наведено деякі з них. Виробниче обладнання має бути розташоване таким чином, щоб зменшити ризики для операторів та інших працівників. Тобто, забезпечити достатню відстань від рухомих до нерухомих частинок приладу, відвести усіх види енергії та речовини, що використовуються або виробляються. Керувати самохідним обладнанням повинні лише ті працівники, що мають право ним керувати, дотримуючись правил руху, що визначені роботодавцем [40].

Також є ряд законів, що регламентують поводження працівників під час переміщення вантажів та роботи на висоті. Обладнання, що здійснює підйом вантажів повинно бути надійно закріплене, враховуючи властивості ґрунту. Під час перебування працівників та такому обладнанні, на посту керівника постійно повинен перебувати оператор. Працівники повинні бути забезпечені надійними засобами зв'язку, на випадок небезпеки потрібно передбачити заходи евакуації. Роботу на висоті слід проводити лише за належних ергономічних умов, з відповідної поверхні, в інших випадках слід обирати найкраще виробниче обладнання, що забезпечить безпечні умови роботи. Пріоритетними є заходи колективного, а не індивідуального захисту. У разі потреби встановлюються

захисні засоби для запобігання падінню та травмуванню працівників. Ці засоби мають підходити за конфігурацією та бути достатньо міцними. Драбини встановлюються із забезпеченням їх стійкості під час використання [40].

ВИСНОВКИ

У даній кваліфікаційній роботі бакалавра було реалізовано спрощену P2P-систему для децентралізованого обміну файлами в межах локальної мережі з використанням технології .Net. Розроблене програмне рішення відповідає поставленим вимогам та може бути використано для зручного файлообміну в домашній та корпоративній мережі.

Було проведено аналіз предметної області системи для розробки системи для децентралізованого обміну файлами, визначено основні інструменти та технології, що використовуватимуться під час роботи над проєктом. Здійснено порівняння конкурентних програмних рішень, сформовано вимоги до власної спрощеної версії P2P-системи.

Розроблено архітектурну модель рішення, зокрема модель предметної області та бізнес-модель, що включає акторів з набором варіантів використання. Архітектурне проєктування системи стало однією з найвагоміших частин роботи. Починаючи з проєктування UML-діаграми підсистем класів, ієрархії класів, завершуючи UML-діаграмами ієрархії класів архітектурних моделей шаблонів та UML-діаграмами послідовностей. Такий підхід до проєктування забезпечив швидке конструювання системи та зменшив ймовірність виникнення критичних помилок.

Було здійснено конструювання окремих підсистем рішення, зокрема мережевої, файлової, підсистеми безпеки та інших. Розроблено ряд компонентів мережевої підсистеми, що включає сервіс виявлення вузлів, менеджер підключень та сервіс передачі файлів. Забезпечено базовий рівень безпеки передачі файлів, який передбачає використання алгоритмів шифрування та перевірку цілісності переданих файлів за допомогою алгоритму хешування. Для взаємодії між компонентами системи впроваджено диспетчер подій. Реалізовано підсистему графічного інтерфейсу користувача, що забезпечує зручну взаємодію із ключовими функціями системи.

Проведено модульне тестування основної логіки та інтеграційне тестування для перевірки взаємодії кількох екземплярів системи одночасно. Такий підхід дозволив перевірити як роботу окремих функцій системи, так і узгодженість роботи основних компонентів системи, що вимагають наявність кількох вузлів у мережі. Зокрема, взаємодію під час з’єднання між вузлами та функції передачі файлів.

Отже, розроблена система відповідає основній частині поставлених вимог і є готовою до використання в реальних умовах. Поточна реалізація P2P-системи для обміну файлами може бути вдосконалена та має потенціал стати ефективним інструментом у сфері децентралізованого файлообміну.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Cohen B. The BitTorrent Protocol Specification, 2008. URL: https://www.bittorrent.org/beps/bep_0003.html (дата звернення: 23.05.2025).

2. "Five Best BitTorrent Clients". Lifehacker.com. URL: https://lifehacker.com/five-best-bittorrent-clients-5813348 (дата звернення: 23.05.2025).

3. Logo of µTorrent. URL: https://commons.wikimedia.org/wiki/File:%CE%9CTorrent_logo.svg (дата звернення: 19.05.2025).

4. Forget online drives, sync directly with BitTorrent Sync – CNET. URL: https://www.cnet.com/tech/computing/forget-online-drives-sync-directly-with-bittorrent-sync/ (дата звернення: 22.05.2025).

5. Current Logo for BitTorrent Sync. URL: https://commons.wikimedia.org/wiki/File:Sync_2.0_Logo.png (дата звернення: 19.05.2025).

6. Syncthing 1.0.0 released as open-source P2P sync tool, finally leaves beta. URL: https://betanews.com/2019/01/03/syncthing/ (дата звернення: 23.05.2025).

7. Block Exchange Protocol v1 — Syncthing documentation. URL: https://docs.syncthing.net/specs/bep-v1.html (дата звернення: 23.05.2025).

8. Understanding Device IDs — Syncthing documentation. URL: https://docs.syncthing.net/dev/device-ids.html (дата звернення: 23.05.2025).

9. Syncthing logo. URL: https://commons.wikimedia.org/wiki/File:SyncthingLogoHorizontal.svg#/media/File:SyncthingLogoHorizontal.svg (дата звернення: 23.05.2025).

10. Introduction to .NET - .NET | Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/core/introduction (дата звернення: 24.05.2025).

11. Overview - A tour of C# | Microsoft Learn. URL: https://learn.microsoft.com/ en-us/dotnet/csharp/tour-of-csharp/overview (дата звернення: 24.05.2025).

12. Biclique Cryptanalysis of the Full AES. URL: https://web.archive.org/web/20160306104007/http:/research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf (дата звернення: 31.05.2025).

13. Rivest R., Shamir A., Adleman L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems // Communications of the ACM. — 1978. — Vol. 21, No. 2. — P. 120–126. URL: https://web.archive.org/web/20230127011251/http://people.csail.mit.edu/rivest/Rsapaper.pdf (дата звернення: 31.05.2025).

14. RFC 3962 - Advanced Encryption Standard (AES) Encryption for Kerberos 5. URL: https://datatracker.ietf.org/doc/html/rfc3962 (дата звернення: 03.06.2025).

15. Penard, W., van Werkhoven, T. "On the Secure Hash Algorithm family". URL: https://web.archive.org/web/20160330153520/https://www.staff.science.uu.nl/~werkh108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf (дата звернення 17.06.2025).

16. What is Windows Presentation Foundation - WPF | Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/ (дата звернення: 31.05.2025).

17. xUnit.net. URL: https://xunit.net (дата звернення: 31.05.2025).

18. devlooped/moq: The most popular and friendly mocking framework for .NET. URL: https://github.com/devlooped/moq (дата звернення: 31.05.2025).

19. Serilog — simple .NET logging with fully-structured events. URL: https://serilog.net (дата звернення: 31.05.2025).

20. Rider: The Cross-Platform .NET IDE from JetBrains. URL: https://www.jetbrains.com/rider/ (дата звернення: 31.05.2025).

21. Git. URL: https://git-scm.com (дата звернення: 31.05.2025).

22. About GitHub and Git - GitHub Docs. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git (дата звернення: 31.05.2025).

23. Що таке Scrum | HURMA. URL: https://hurma.work/blog/scrum-pidkhid-dlya-biznesu/ (дата звернення: 31.05.2025).

24. Scrum with GitHub No Jira Required! – YouTube. URL: https://www.youtube.com/watch?v=Is9KZFkHmpk (дата звернення: 05.03.2025).

25. Understanding Epics in Agile for Managing and Chunking Work Items. URL: https://resources.scrumalliance.org/Article/epic-agile (дата звернення: 05.06.2025).

26. Agile Planning: Step-By-Step Guide | monday.com Blog. URL: https://monday.com/blog/rnd/agile-planning/ (дата звернення: 05.06.2025).

27. Fibonacci Agile Estimation: What Is It and Why Does it Work? URL: https://www.parabol.co/blog/fibonacci-estimation/ (дата звернення: 16.06.2025).

28. Курс «Аналіз вимог до програмного забезпечення (SE322)» — ТНТУ ім. І. Пулюя. URL: https://dl.tntu.edu.ua/bounce.php?course=1559 (дата звернення: 06.06.2025).

29. Курс «Моделювання та аналіз програмного забезпечення» — ТНТУ ім. І. Пулюя. URL: https://dl.tntu.edu.ua/bounce.php?course=1351 (дата звернення: 09.06.2025).

30. Component diagrams - IBM Documentation. URL: https://www.ibm.com/docs/en/dma?topic=diagrams-component (дата звернення: 11.06.2025).

31. Refactoring.Guru. URL: https://refactoring.guru (дата звернення: 13.06.2025).

32. Explore the UML sequence diagram - IBM Developer. URL: https://developer.ibm.com/articles/the-sequence-diagram/ (дата звернення: 13.06.2025).

33. Курс «Конструювання програмного забезпечення (SE211)» — ТНТУ ім. І. Пулюя. URL: https://dl.tntu.edu.ua/bounce.php?course=1737 (дата звернення: 14.06.2025).

34. Aes Class (System.Security.Cryptography) | Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes (дата звернення: 03.06.2025).

35. Що таке тестування програмного забезпечення: види, етапи, інструменти. URL: https://university.sigma.software/what-is-software-testing/ (дата звернення: 17.06.2025).

36. Охорона праці в галузі: метод. вказівки до виконання кваліфікаційної роботи / уклад. : І. Б. Гевко, Ю. О. Соколовська. – Тернопіль : ТНТУ, 2023. – 36 с. URL: https://dl.tntu.edu.ua/content.php?cid=299862 (дата звернення: 02.06.2025).

37. Про затвердження порядків надання домедичної допомоги особам при невідкладних станах: наказ від 09.03.2022 № 441. URL: https://zakon.rada.gov.ua/laws/show/z0356-22 (дата звернення 07.06.2025).

38. Мелех Л. В. Безпека життєдіяльності та охорона праці: навч. посіб. / Л. В. Мелех. – Львів : ЛьвДУВС, 2022. – 228 с. URL: https://dspace.lvduvs.edu.ua/bitstream/1234567890/4863/1/Мелех%20Л.В.%20БЖД%20та%20ОП%20навчальний%20посібник%202022_%20(2).pdf (дата звернення: 02.06.2025).

39. Охорона праці в галузі: метод. вказівки до виконання диплом. проєктів для студентів спец. 133 "Галузеве машинобудування" / уклад. : І. Б. Гевко, Ю. О. Соколовська. – Тернопіль : ТНТУ, 2022. – 32 с. URL: https://dl.tntu.edu.ua/content.php?cid=289118 (дата звернення: 02.06.2025).

40. Про затвердження Порядку проведення навчання і перевірки знань з питань охорони праці: наказ Мінсоцполітики України від 29.01.2018 № 33. URL: https://zakon.rada.gov.ua/laws/show/z0097-18 (дата звернення: 02.06.2025).

41. Петрик М.Р. Проєктування програмного забезпечення на основі аналізу вимог та інструментальних засобів розробки IBM Rational Software

Architect (від Вимог до коду) Науково-методичний посібник. Тернопіль: Вид-во ТНТУ ім. Івана Пулюя.-2022.- 560с.

42. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

ДОДАТКИ

ДОДАТОК А

Тези конференції

VIII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 621.326
Олійник Ю. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

ЕЛЕМЕНТИ ПРОЄКТУВАННЯ ДЛЯ СПРОЩЕНОЇ ОДНОРАНГОВОЇ P2P-СИСТЕМИ ОБМІНУ ФАЙЛАМИ

Науковий керівник: д.ф.-м.н., професор Петрик М.Р.

Oliinyk Y.
Ternopil Ivan Puluj National Technical University

DESIGN ELEMENTS FOR A SIMPLIFIED P2P FILE SHARING SYSTEM

Supervisor: Ph.D.-M.Sc., professor Petryk M.R.

Ключові слова: децентралізація, P2P, обмін файлами, проєктування,
Keywords: decentralization, P2P, file sharing, system design

P2P (Peer-to-Peer) системи в сучасному світі знаходять свою цінність як надійна та безпечна альтернатива стандартній клієнт-серверній архітектурі. У чистому вигляді, перевагою однорангової архітектури є її повна незалежність від сервера чи певного користувача. Кожен клієнт мережі є одночасно й сервером, що дозволяє створювати довільні зв'язки в мережі. З'єднання та обмін даними між вузлами відбувається за протоколом TCP [1]. Визначення вузлів у мережі відбувається шляхом використання протоколу UDP.

В даному типі систем встановлюється періодичне надсилання вузлами деяких даних про себе у мережу. Комунікація може відбуватись за моделлю unicast (one-to-one), multicast (many-to-many) або broadcast (one-to-all), хоча остання модель не рекомендується через надмірне навантаження на мережу. Для обміну файлами доцільно використовувати саме multicast. Ця модель також ділиться на різні типи зв'язків [2]. Система обміну файлами, що розглядається в даній роботі використовує зв'язок 1-to-M для встановлення з'єднання, а також N-to-M для передачі файлів між вузлами. Це забезпечує паралельний прийом та передачу довільної кількості файлів у мережі.

На відміну від інших P2P систем файлообміну, в даній реалізації не передбачено swarming-механізм розбиття файлів на частинки (як у BitTorrent). Дане рішення є спрощеною версією класичної однорангової системи, що володіє властивістю децентралізації та рівноправності вузлів, проте, з прямою передачею файлів між ними. Крім цього, запропонована реалізація працює в межах локальної мережі, що суттєво знижує складність системи. Тому, поточне рішення орієнтоване на використання в межах домашнього або корпоративного середовища.

Проєктування архітектури є одним з ключових етапів у життєвому циклі програмного забезпечення. Запропонована система будується на основі модульної архітектури, що забезпечує розбиття рішення на окремі модулі, що спрощує процес розробки та підтримки. У процесі розробки значну роль у формулюванні ідей, оптимізації етапів проєктування та покращенні якості окремих компонентів системи відіграла підтримка штучного інтелекту.

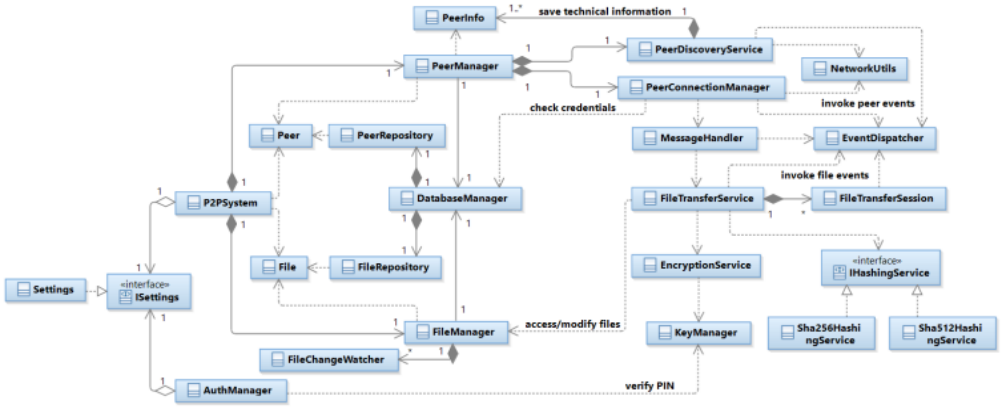

Рисунок 1. Загальний вигляд діаграми класів P2P-системи для обміну файлами

P2P-система обміну файлами складається з кількох модулів. Модуль Network становить ключову частину архітектури. Його завданням є визначення вузлів у мережі, управління з'єднаннями між ними, а також обмін даними між користувачами. Network не є повністю автономним, оскільки частково залежить від модуля бази даних Database Manager та файлової системи File Storage, що обробляє інформацію пов'язану з файлами. Обидва модулі реалізують патерн одинак, що забезпечує доступ до них з мережевого модуля. Окремо виділяється безпековий модуль Security, що є представленим як набір компонентів, що забезпечують шифрування, хешування, та роботу з ключами доступу.

Особливістю цього архітектурного рішення є використання кількох патернів проєктування. У модулі Core реалізовано патерн фасад, що застосовується для забезпечення централізованого керування модулями Network та File Storage. Network містить додатковий фасад, що виконує роль менеджера вузлів, координуючи роботу сервісу визначення вузлів та менеджера з'єднань. Натомість, модуль UI використовує Core та AuthManager як інтерфейси для роботи з усією системою. Сервіс EventDispatcher, що реалізує патерн одинак, задовольняє потребу в комунікації між модулем Network та UI, надаючи можливість оновлення поточного стану системи на інтерфейсі користувача. Ще одним патерном проєктування в цьому рішенні є стратегія. Модуль Security дозволяє обирати тип хешування даних, що забезпечує адаптивність відповідно до умов роботи системи. На завершення, відповідно до сучасних стандартів розробки ПЗ, впроваджено модуль Tests для тестування компонентів системи.

Підсумовуючи, пропоноване рішення задовольняє основні критерії однорангової P2P-системи, що побудована на сучасних архітектурних підходах, які забезпечують гнучкість, надійність та ефективність під час розробки та підтримки програмного забезпечення.

Література

1. Lui, Carrie & Kwok, Sai. (2002). Interoperability of peer-to-peer file sharing protocols. SIGecom Exchanges. 3. 25-33. 10.1145/844339.844350.
2. Masinde, Newton & Graffi, Kalman. (2020). Peer-to-Peer-Based Social Networks: A Comprehensive Survey. SN Computer Science. 1. 299. 10.1007/s42979-020-00315-8.

ДОДАТОК Б

Лістинг коду методу FileTransferService.ReceiveFileAsync

```
public static async Task<bool> ReceiveFileAsync(string message, Stream stream, bool bypassEncryption = false)
{
    // Validate message format
    message = message.Substring(5);

    // Parse the message
    var messageParts :string[] = message.Split('|');
    var senderPublicKey :string = messageParts[0];
    var fileName = messageParts[1];
    var fileSize :long = long.Parse(messageParts[2]);
    var fileHash :string = messageParts[3];
    var aesKey :string = messageParts[4];

    FileTransferSession? session = null;
    try
    {
        session = GetOrCreateSession(stream, senderPublicKey);

        // Receive the file
        var filePath = Path.Combine(FileManager.GetInstance().GetDefaultPath(), fileName);
        var result :string? = await session.ReceiveFileAsync(senderPublicKey, filePath, fileSize);
        if (result == null)
        {
            RemoveSession(senderPublicKey, session);
            session.Dispose();
            return false;
        }

        // Decrypt the file if not bypassing encryption
        if (!bypassEncryption)
        {
            RaiseFileDecryptionStarted(fileName, senderPublicKey);
            var decryptionResult :string = EncryptionService.DecryptFile(result, aesKey)
                                ?? throw new Exception("File decryption failed");
            RaiseFileDecryptionCompleted(fileName, senderPublicKey);

            File.Delete(result);
            File.Move( sourceFileName: decryptionResult,  destFileName: result);
        }

        // Validate the file hash
        var hashAlgorithm :string =
            fileHash.StartsWith("SHA256") ? "SHA256" :
            fileHash.StartsWith("SHA512") ? "SHA512" :
            throw new Exception("Hash algorithm not supported");

        var expectedHash :string = fileHash.Substring(7);

        // Hash the received file
        RaiseFileReceiveHashingStarted(fileName, senderPublicKey);
        var calculatedHash :string = hashAlgorithm == "SHA256"
            ? new Sha256HashingService().Hash(result)
            : new Sha512HashingService().Hash(result);
        RaiseFileReceiveHashingCompleted(fileName, senderPublicKey);
```

```
        if (expectedHash != calculatedHash)
        {
            var attempt = 3;
            while (attempt > 0)
            {
                // Send a retry signal
                await stream.WriteAsync( buffer: [0],  offset: 0,  count: 1);
                await stream.FlushAsync();

                // Retry receiving the file
                result = await session.ReceiveFileAsync(senderPublicKey, filePath, fileSize)
                if (result == null)
                {
                    RemoveSession(senderPublicKey, session);
                    session.Dispose();
                    return false;
                }

                if (!bypassEncryption)
                {
                    // Decrypt the file again
                    RaiseFileDecryptionStarted(fileName, senderPublicKey);
                    var decryptionResult :string  = EncryptionService.DecryptFile(result, aesKe
                                                throw new Exception("File decryption failed");
                    RaiseFileDecryptionCompleted(fileName, senderPublicKey);

                    File.Delete(result);
                    File.Move( sourceFileName: decryptionResult,  destFileName: result);
                }

                // Hash the received file
                RaiseFileReceiveHashingStarted(fileName, senderPublicKey);
                calculatedHash = hashAlgorithm == "SHA256"
                    ? new Sha256HashingService().Hash(result)
                    : new Sha512HashingService().Hash(result);
                RaiseFileReceiveHashingCompleted(fileName, senderPublicKey);

                // Compare the hashes
                if (expectedHash == calculatedHash)
                    break;

                attempt--;
            }
            if (attempt == 0)
                throw new Exception("File transfer failed after 3 attempts");
        }

        // Send a success signal
        await stream.WriteAsync( buffer: [1],  offset: 0,  count: 1);
        await stream.FlushAsync();

        // Add the received file to the file manager
        await FileManager.GetInstance().AddReceivedFileAsync( path: result, senderPublicKey);
        RaiseFileReceiveCompleted(fileName,  destinationPath: result, senderPublicKey);
        return true;
    }
    catch (Exception e)
    {
        Logger.LogError($"Error receiving file {fileName}", e);
        RaiseFileReceiveFailed(fileName, senderPublicKey, e.Message);
        if (session == null) return false;
        RemoveSession(senderPublicKey, session);
        session.Dispose();
        return false;
    }
}
```

ДОДАТОК В

Диск