БАКАЛАВРСЬКА РОБОТА

БР.КІ-45.00.00.000 ПЗ

Група КІ-21-2

Олійник Роман

2025

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Олійник Роман Петрович

УДК 004.738.5:004.4

БАКАЛАВРСЬКА РОБОТА

Розробка локального корпоративного месенджера для комунікації учнів Калуського ліцею ім. Д. Бахматюка

Комп'ютерна інженерія
(назва освітньої програми)

123 – Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього ступеня *Олійник Р.П.*
(підпис, ініціали та прізвище здобувача)

Науковий керівник *Кропивницький Д.Р., доцент*
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри
д.т.н., професор /С. І. Мельничук/
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025 рік

Івано-Франківський національний технічний університет нафти і газу

Факультет *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ**:**

Зав. кафедрою КСМ

д.т.н. С.І. Мельничук

« 05 » травня 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Олійнику Роману Петровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи*) Розробка локального корпоративного месенджера для комунікації учнів Калуського ліцею ім. Д. Бахматюка*

керівник проекту (роботи) *Кропивницький Дмитро Романович, доцент*

затверджені наказом вищого навчального закладу від *05.05.2025 № 275/7*

2. Строк подання студентом проекту (роботи) *12 червня 2025р.*

3. Вихідні дані до роботи *Методичні вказівки, технічна література*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналітичний огляд теоретичних основ безпечного обміну повідомленнями в локальних мережах. 2. Проєктування корпоративного месенджера для Калуського ліцею ім. Д. Бахматюка. 3. Реалізація та тестування безпеченого месенджера.*

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

_____

_____

6. Консультанти розділів роботи

7. Дата видачі завдання *29 січня 2025 р.* .

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Збір інформації, вивчення літератури та пошук додаткової інформації*	*Лютий 2025р*	
2	*Аналітичний огляд теоретичних основ безпечного обміну повідомленнями в локальних мережах*	*Березень 2025р*	
3	*Проєктування корпоративного месенджера для Калуського ліцею ім. Д. Бахматюка*	*Квітень 2025р*	
4	*Реалізація та тестування безпечного месенджера*	*Травень 2025р*	
5	*Оформлення додатків, дипломної роботи*	*Червень 2025р*	

Студент _____ *Олійник Р.П.*

Керівник роботи _____ *Кропивницький Д.Р.*

АНОТАЦІЯ

У дипломній роботі розглядається процес розробки безпечного корпоративного месенджера для локальної мережі Калуського ліцею ім. Д. Бахматюка. Метою дослідження є створення програмного продукту, який забезпечує захищене спілкування між учнями без використання підключення до Інтернету.

У першому розділі подано загальну характеристику ліцею як об'єкта впровадження інформаційної системи, розкрито поняття безпечного чату та його призначення, проаналізовано основні види загроз у локальних мережах та методи захисту даних. Також здійснено огляд наявних месенджерів, що дозволило виявити їхні недоліки та переваги, і на основі цього сформувати вимоги до розроблюваної системи.

Другий розділ присвячено безпосередньому проєктуванню месенджера. Детально описано структуру додатку, архітектуру клієнт-серверної системи, обґрунтовано вибір технологій та засобів розробки. Сформульовано вимоги до функціоналу та безпеки програмного забезпечення, а також описано протокол, який використовується для обміну повідомленнями.

У третьому розділі представлено реалізацію серверної та клієнтської частин програми, розроблено зручний інтерфейс користувача. Проведено тестування системи на відповідність функціональним вимогам, оцінено рівень захищеності даних та стабільність роботи. Особливу увагу приділено аналізу результатів тестування за участю учнів ліцею, що дозволило визначити ефективність впровадженого рішення та перспективи його вдосконалення.

Ключові слова: корпоративний месенджер, безпечний чат, локальна мережа, обмін повідомленнями, шифрування, клієнт-сервер, захист інформації, архітектура системи, тестування, реалізація.

ANNOTATION

This thesis examines the process of developing a secure corporate messenger for the local network of Kalush Lyceum named after D. Bakhmatiuk. The aim of the study is to create a software product that enables secure communication between students without the use of an Internet connection.

The first chapter provides a general overview of the lyceum as the target for the implementation of the information system, defines the concept and purpose of a secure chat, analyzes the main types of threats in local networks, and reviews data protection methods. An analysis of existing messengers is also presented, highlighting their advantages and disadvantages, which helped define the requirements for the developed system.

The second chapter is dedicated to the direct design of the messenger. It describes the application structure, the architecture of the client-server system, and justifies the choice of technologies and development tools. The functional and security requirements for the software are formulated, along with a description of the message exchange protocol used in the system.

The third chapter covers the implementation of both the server and client parts of the application and the development of a user-friendly interface. System testing was carried out to evaluate its compliance with functional requirements, data protection level, and operational stability. Special attention is given to the analysis of testing results involving lyceum students, which allowed for the assessment of the solution’s effectiveness and the identification of opportunities for further improvement.

Keywords: corporate messenger, secure chat, local network, message exchange, encryption, client-server, information security, system architecture, testing, implementation.

ЗМІСТ

ВСТУП

Інформаційні технології стали невід'ємною частиною повсякденного життя, відіграючи важливу роль передусім в освітньому середовищі. Школи та навчальні заклади дедалі частіше використовують цифрові інструменти для комунікації, організації навчального процесу та обміну інформацією. Однак, питання безпеки такої комунікації серед учнів набуває особливої актуальності. Зважаючи на постійні загрози інформаційній безпеці, зокрема витоки особистих даних, кібератаки та небажані контакти, виникає потреба у створенні локального корпоративного месенджера для Калуського ліцею імені Дмитра Бахматюка, який дозволив би учням спілкуватися в безпечному та контрольованому середовищі.

Актуальність теми зумовлена зростаючим попитом на безпечні засоби цифрової комунікації в освітніх установах, а також необхідністю забезпечення конфіденційності та цілісності даних у процесі обміну повідомленнями між учнями.

Об'єктом дослідження є процес обміну повідомленнями в локальних мережах навчальних закладів.

Предметом дослідження виступають методи проєктування та реалізації безпечних чат-систем, які функціонують у межах локальної мережі.

Метою роботи є розробка локального корпоративного месенджера, який може бути використаний учнями в межах локальної мережі школи або іншого навчального закладу.

Для досягнення поставленої мети були сформульовані такі завдання:

– проаналізувати існуючі рішення для локального обміну повідомленнями;

– вивчити сучасні засоби захисту інформації у мережевих застосунках;

– спроєктувати архітектуру корпоративного месенджеру для локальної мережі;

– реалізувати прототип програми з урахуванням вимог до безпеки;

– провести тестування та аналіз функціонування системи.

У процесі роботи використовувались такі **методи дослідження**, як аналіз літературних джерел, моделювання, порівняльний аналіз, програмна реалізація, а також тестування програмного забезпечення.

Практичне значення роботи полягає у створенні прототипу корпоративного месенджера, який може бути впроваджений у навчальному середовищі для організації внутрішнього спілкування між учнями без необхідності використання сторонніх інтернет-сервісів, що підвищує загальний рівень інформаційної безпеки.

1 ТЕОРЕТИЧНІ ОСНОВИ БЕЗПЕЧНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ

1.1 Характеристика Калуського ліцею ім. Д. Бахматюка

Калуський ліцей імені Дмитра Бахматюка — це сучасний заклад загальної середньої освіти II–III ступенів, розташований у місті Калуш Івано-Франківської області. Ліцей забезпечує освітній процес для понад 500 учнів, охоплюючи широкий спектр навчальних напрямків, зокрема математичний, філологічний та інформаційно-технологічний профілі.

Заклад активно впроваджує інноваційні технології в навчальний процес. У ліцеї функціонують декілька комп'ютерних класів, оснащених сучасними персональними комп'ютерами та доступом до локальної мережі. Основу IT-інфраструктури складає внутрішня Ethernet-мережа, що з'єднує всі кабінети, адміністрацію та бібліотеку. Однак, на момент написання роботи комунікація між учнями та вчителями відбувається здебільшого через загальнодоступні месенджери, які не гарантують належного рівня конфіденційності та не відповідають вимогам безпеки персональних даних учасників освітнього процесу.

Одним із викликів, що стоїть перед ліцеєм, є забезпечення безпечної, швидкої та автономної комунікації між учнями, вчителями та адміністрацією в межах локальної мережі, без використання сторонніх Інтернет-сервісів. У зв'язку з цим виникла потреба у створенні локального корпоративного месенджера, який би функціонував автономно в межах шкільної мережі, забезпечував автентифікацію користувачів, шифрування повідомлень та мав простий інтерфейс, зручний для школярів різного віку.

Калуський ліцей ім. Д. Бахматюка виступає базовим об'єктом для впровадження розробленої системи, а його особливості та потреби визначають вимоги до функціоналу, структури та безпеки майбутнього програмного забезпечення [1].

Розробка корпоративного месенджера для учнів ліцею повинна відповідати чинному законодавству України у сфері освіти, інформаційної безпеки та захисту персональних даних. Основними нормативно-правовими актами, які регулюють відповідні аспекти, є:

1. Закон України «Про освіту» — передбачає створення безпечного освітнього середовища, зокрема в цифровому просторі, та відповідальність закладів освіти за захист інформації учасників освітнього процесу [2].
2. Закон України «Про захист персональних даних» — визначає порядок обробки персональних даних та вимоги до їх захисту. В межах реалізації месенджера необхідно забезпечити збереження конфіденційності інформації про користувачів (учнів, вчителів), її захист від несанкціонованого доступу [3].
3. Закон України «Про інформацію» — закріплює права громадян на захист інформації, вимоги до її достовірності, доступу та обмеження поширення [4].
4. Концепція кібербезпеки України — містить принципи організації захисту інформаційних систем у сфері освіти, відповідальність за інциденти інформаційної безпеки та шляхи захисту від кіберзагроз [5].
5. Накази Міністерства освіти і науки України — регламентують вимоги до впровадження цифрових сервісів у закладах освіти, зокрема щодо цифрової грамотності, безпечної поведінки в мережі та відповідального використання інформаційно-комунікаційних технологій [6].

Тому створення безпечного месенджера має здійснюватися з дотриманням як національних правових норм, так і з урахуванням кращих міжнародних практик у сфері цифрової безпеки.

1.2 Поняття безпечного чату та його призначення

Сьогодні, коли цифрові технології стали частиною повсякденного життя, люди постійно обмінюються інформацією — від звичайних особистих повідомлень до важливих бізнесових чи державних даних. Проте, зручність

швидкого зв'язку невід'ємно пов'язана з ризиками, що походять від потенційного несанкціонованого доступу, перехоплення або витоку інформації.

Саме тому поняття безпечного чату набуває неабиякої актуальності. Безпечний чат – це система обміну повідомленнями, яка розроблена з використанням спеціальних криптографічних та організаційних методів для забезпечення конфіденційності, цілісності та автентичності інформації, що передається. Його призначення виходить за рамки простої передачі тексту, фокусуючись на захисті самої суті комунікації.

Це, по суті, будь-яка платформа для обміну повідомленнями, яка інтегрує передові технології захисту даних, що дозволяють користувачам спілкуватися, не турбуючись про те, що їхня приватна інформація буде прочитана, змінена або перехоплена сторонніми особами. Це не просто функція, а комплексний підхід до побудови системи, де кожен компонент, від клієнтського застосунку до серверної інфраструктури, розроблений з урахуванням найвищих стандартів кібербезпеки.

Основу безпечного чату складає наскрізне шифрування (End-to-End Encryption, E2EE). Цей механізм гарантує, що повідомлення шифруються на пристрої відправника і можуть бути розшифровані лише на пристрої отримувача. Навіть сам провайдер послуги чату, чи будь-яка третя сторона, яка може отримати доступ до серверів, не зможе прочитати вміст повідомлень [7]. Це фундаментально відрізняє безпечні чати від традиційних, де дані можуть бути зашифровані лише між клієнтом і сервером, але доступні провайдеру послуги. E2EE забезпечує конфіденційність комунікації, роблячи її приватною та недоступною для зовнішнього спостереження.

Крім конфіденційності, безпечний чат акцентує увагу на цілісності даних. Це означає, що відправлене повідомлення дійде до отримувача без будь-яких несанкціонованих змін. Криптографічні хеш-функції та цифрові підписи використовуються для верифікації того, що повідомлення не було підроблено або спотворено під час передачі [8].

Автентифікація є ще однією ключовою складовою. Безпечний чат гарантує, що користувач спілкується саме з тією особою, за яку вона себе видає, і що відправник повідомлення є тим, ким він стверджує. Це досягається за допомогою надійних методів ідентифікації та верифікації облікових записів. Багатофакторна автентифікація (MFA) стає стандартом для підвищення рівня безпеки доступу до чату.

Таблиця 1.1 - Ключові принципи та їх реалізація у безпечних чатах

Принцип безпеки	Опис	Основні механізми реалізації
Конфіденційність	Захист вмісту повідомлень від несанкціонованого ознайомлення.	Наскрізне шифрування (E2EE), шифрування даних на пристрої.
Цілісність	Гарантія того, що повідомлення не були змінені під час передачі.	Криптографічні хеші, цифрові підписи.
Автентифікація	Перевірка справжності учасників комунікації.	Сильні паролі, багатофакторна автентифікація (MFA), біометрія.
Неспростовність	Доказ факту відправлення або отримання повідомлення.	Цифрові підписи, протоколи обміну ключами.
Безпечне зберігання	Захист зашифрованих повідомлень та ключів на пристроях та серверах.	Шифрування даних на рівні диска, захищені сховища ключів.

Призначення безпечного чату полягає у наданні користувачам платформи для захищеного спілкування, де вони можуть бути впевнені у приватності та автентичності своєї переписки. Ця потреба є універсальною, але особливої гостроти набуває у навчальних закладах.

Зараз комп’ютери, смартфони та інтернет стали звичними інструментами у житті школярів. Вони щодня спілкуються онлайн — обговорюють домашні завдання, готують спільні проєкти або просто переписуються з друзями. Але разом із цим виникають і певні ризики. Якщо чат не захищений, сторонні люди можуть перехопити особисті повідомлення, викрасти дані або навіть писати образи — тобто виникає загроза кібербулінгу чи втручання в особисте життя.

Тому створення безпечного чату спеціально для школярів — дуже важлива і своєчасна ідея. Такий чат може надійно захищати інформацію, обмежувати доступ стороннім і не допускати поширення небажаного або шкідливого контенту. Крім того, у захищеному середовищі легше

організовувати навчання, спілкування між учнями та вчителями, особливо коли частина уроків проходить онлайн.

Впровадження безпечного шкільного чату допомагає не тільки захистити дітей, а й покращує взаєморозуміння в школі, вчить відповідальному ставленню до інтернету та створює комфортні умови для спілкування.

Поняття безпечного чату виходить далеко за межі звичайної програми для обміну повідомленнями. Це комплексне рішення, що інтегрує передові криптографічні механізми та суворі політики безпеки для забезпечення конфіденційності, цілісності та автентичності комунікації.

Його мета – надати користувачам можливість спілкуватися вільно та безпечно, захищаючи їхню приватність та чутливу інформацію від зовнішніх загроз. У наш час, коли зберегти особисту інформацію стає все важче, захищені чати потрібні не тільки для зручності, а й для безпеки людей, компаній та державних установ.

1.3 Види загроз і методи захисту даних у локальних мережах

Передача даних у мережі є одним із найвразливіших моментів у сфері безпеки. Існують різні типи загроз, які можуть вплинути на конфіденційність, цілісність, доступність і автентичність інформації. Наприклад, якщо хтось перехоплює повідомлення, навіть не змінюючи його, — це вже порушення конфіденційності. Така атака називається "сніфінг" — зловмисник просто підключається до мережі й читає незашифровані дані. Щоб цього уникнути, потрібно завжди використовувати шифрування [9].

Більш небезпечна атака — це "людина посередині" (MITM), коли зловмисник вміщується між двома сторонами і може не лише підглядати, а й змінювати повідомлення. Такі атаки можливі через слабкі місця у мережі, як-от підміна IP-адрес чи сертифікатів. Щоб захиститися, використовують надійні протоколи, наприклад TLS, і сертифікати безпеки [10].

Ще одна проблема — це аналіз трафіку. Навіть якщо самі повідомлення зашифровані, можна побачити, хто з ким спілкується і як часто. Ця інформація вже може бути цінною. Щоб її приховати, використовують VPN або спеціальні анонімні мережі.

Загроза цілісності виникає, коли хтось змінює дані під час передачі. Наприклад, змінює суму грошей у транзакції. Тут допомагають хеш-функції та цифрові підписи, які дозволяють виявити будь-які зміни. Ще один ризик — повторні атаки, коли старе перехоплене повідомлення відправляється повторно. Для захисту потрібні мітки часу або одноразові коди.

Що стосується доступності, то найбільша загроза — це атаки типу DoS і DDoS. Вони перевантажують сервер або сайт величезною кількістю запитів, роблячи його недоступним для звичайних користувачів. Якщо атака йде з одного джерела — це DoS, якщо з багатьох — DDoS. Для захисту використовують спеціальні фільтри, брандмауери й інші системи безпеки [11].

Також існує підміна особистості (spoofing), коли зловмисник видає себе за когось іншого, наприклад, надсилає електронного листа від імені іншої людини. Захистом тут є цифрові сертифікати й перевірка ідентичності.

Інша проблема — це заперечення факту відправлення або отримання повідомлень. Наприклад, хтось може сказати, що не надсилав листа, хоча це зробив. Для уникнення цього використовують цифрові підписи, які неможливо підробити, і сервіси, що підтверджують факт передачі.

Таблиця 1.2 - Види загроз при передачі даних

Вид загрози	Суть та типові приклади	Вплив на інформаційну безпеку
1	2	3
Перехоплення (Sniffing)	Пасивне прослуховування мережевого трафіку для збору даних (напр., Wireshark).	Конфіденційність
"Людина посередині" (MITM)	Зловмисник перехоплює та потенційно змінює комунікацію (ARP spoofing, SSL stripping).	Конфіденційність, цілісність, автентичність
Підміна даних (Tampering)	Несанкціонована зміна інформації під час її передачі.	Цілісність
Відмова в обслуговуванні (DoS/DDoS)	Перевантаження мережевих ресурсів, що унеможливлює доступ легітимним користувачам.	Доступність

Кінець таблиці 1.2

1	2	3
Підробка ідентичності (Spoofing)	Видання себе за іншу особу або пристрій (IP spoofing, Email spoofing).	Автентичність
Заперечення (Repudiation)	Відправник заперечує відправлення повідомлення або отримувач — його отримання.	Неспростовність

Усі ці загрози постійно змінюються, з’являються нові способи атак. Тому захист повинен бути комплексним: сучасне шифрування, надійні протоколи, регулярне оновлення програм, аналіз трафіку і, що важливо, обізнаність користувачів. Лише такий підхід дає шанс забезпечити безпечну передачу даних у мережі.

Локальні мережі (ЛМ) є основою сучасної інформаційної інфраструктури, забезпечуючи внутрішній зв’язок і доступ до ресурсів для організацій різного масштабу — від малих офісів до великих корпорацій, навчальних закладів і держустанов. Через ЛМ передається велика кількість конфіденційної інформації, що робить їх привабливою мішенню для кіберзлочинців [12].

Сучасні загрози — шкідливе ПЗ, цілеспрямовані атаки, внутрішні ризики — вимагають комплексного багаторівневого підходу до захисту інформації. Ефективний захист у локальних мережах базується на фундаментальних принципах інформаційної безпеки:

- конфіденційність — доступ до інформації лише для авторизованих осіб;
- цілісність — захист від несанкціонованих змін або знищення даних;
- доступність — забезпечення доступу для легітимних користувачів у будь-який час;
- автентичність — підтвердження ідентичності користувачів та ресурсів;
- неспростовність — можливість довести факт певної дії чи транзакції.

Для цього застосовується багаторівневий захист, де кожен рівень доповнює інший, створюючи міцний бар’єр проти загроз.

Організаційні методи — основа будь-якої системи безпеки. Навіть найсучасніші технології не спрацюють без чітких правил і компетентного персоналу. Важливо розробити політики та процедури, що визначають правила користування мережевими ресурсами, вимоги до паролів, регламенти роботи з конфіденційною інформацією, порядок підключення особистих пристроїв (BYOD) та реакції на інциденти.

Критично важливим є навчання користувачів — людський фактор часто є найслабшою ланкою. Регулярні тренінги і інформаційні кампанії допомагають розпізнавати фішинг, соціальну інженерію та усвідомлювати важливість дотримання політик безпеки.

Впровадження розмежування прав доступу — необхідний захід. Принцип найменших привілеїв гарантує, що користувач отримує тільки необхідні права, мінімізуючи шкоду при компрометації. Часто застосовується рольова модель контролю доступу (RBAC).

Фізичний захист також важливий, хоч часто його недооцінюють. Контроль доступу до серверних приміщень, мережевого обладнання, джерел безперебійного живлення та захист від несанкціонованого втручання є першим рівнем безпеки.

Регулярний аудит і моніторинг мережевої активності є обов’язковими. Аналіз журналів подій допомагає вчасно виявити аномалії і загрози та оперативно реагувати на них.

Кожна організація повинна мати план реагування на інциденти — чіткий набір дій при кібератаках, витоках даних чи інших загрозах. Швидка реакція дозволяє мінімізувати збитки.

Технічні методи захисту є безпосередніми інструментами для реалізації принципів безпеки на рівні мережевої інфраструктури, програмного забезпечення та даних.

Таблиця 1.3 - Основні технічні методи захисту інформації в ЛМ

Категорія захисту	Метод / Технологія	Призначення
Мережевий захист	Брандмауери та сегментація мережі	Фільтрація мережевого трафіку, ізоляція сегментів.
Виявлення загроз	Системи IDS/IPS та SIEM	Моніторинг атак, виявлення інцидентів безпеки.
Захист кінцевих точок	Антивірус/EDR та патч-менеджмент	Захист від шкідливого ПЗ, усунення вразливостей.
Захист даних	Шифрування даних та резервне копіювання	Забезпечення конфіденційності та відновлення даних.
Управління доступом	IAM та MFA	Надійна ідентифікація, контроль прав користувачів.

Мережева безпека є першим рівнем захисту інформаційної інфраструктури. Основним її інструментом виступає брандмауер (Firewall) — пристрій або програма, що контролює дозволений чи заборонений трафік. Також він дозволяє сегментувати мережу (VLAN), обмежуючи розповсюдження загроз [13].

Системи виявлення та запобігання вторгненням (IDS/IPS) аналізують трафік: IDS фіксує підозрілу активність, а IPS може її негайно блокувати [14]. VPN (віртуальні приватні мережі) забезпечують захищене підключення до ЛМ ззовні, створюючи зашифрований канал. Контроль доступу (NAC) перевіряє пристрої перед підключенням, оцінюючи наявність антивірусного ПЗ, оновлень тощо.

Захист кінцевих точок охоплює антивіруси та сучасні засоби EDR/XDR, які виявляють загрози на рівні комп’ютерів, серверів і мобільних пристроїв [15]. Керування вразливостями (патч-менеджмент) передбачає регулярне оновлення ПЗ для усунення "дір" у захисті. Шифрування дисків (наприклад, BitLocker) унеможливлює доступ до даних у разі втрати пристрою.

Захист даних реалізується через шифрування при зберіганні та передачі (TLS/SSL, VPN), а також регулярне резервне копіювання. Управління доступом (IAM) охоплює створення облікових записів, складні паролі, багатофакторну автентифікацію (MFA). Системи SIEM забезпечують моніторинг, аналіз журналів подій та швидке реагування [16].

Загалом, надійний захист ЛМ вимагає комплексного підходу, що поєднує технічні засоби й організаційні заходи. Лише системний підхід дозволяє ефективно протистояти сучасним загрозам та підтримувати стабільну роботу мережі.

1.4 Аналіз існуючих месенджерів

Питання приватності та безпеки обміну повідомленнями сьогодні є особливо актуальним. Від особистих розмов до конфіденційних корпоративних даних — інформація в мережі постійно під загрозою перехоплення або втручання. У відповідь з'явилися різні платформи захищеного обміну повідомленнями, кожна з яких по-своєму реалізує безпеку.

Основними критеріями для оцінки таких систем є: наявність наскрізного шифрування (щоб тільки відправник і отримувач могли читати повідомлення), відкритий вихідний код (для незалежної перевірки безпеки), мінімальний збір даних, простий та зручний інтерфейс, доступність на різних пристроях. Важливими є й додаткові функції — двофакторна авторизація, зникаючі повідомлення, заборона скріншотів тощо [17].

Розглянемо кілька популярних месенджерів з акцентом на їхні переваги та недоліки:

Signal – вважається еталоном безпеки. Використовує власний протокол для наскрізного шифрування всіх типів комунікацій. Має відкритий код і мінімальний збір метаданих. Недоліки: іноді менша база користувачів і простий дизайн, який комусь може здатися надто мінімалістичним.

Telegram – популярний завдяки швидкості та функціональності, підтримує великі групи. Однак наскрізне шифрування працює лише в секретних чатах, а звичайні повідомлення шифруються менш надійно, зберігаються на серверах. Код серверів закритий, є активний збір метаданих. В Україні Telegram заборонений.

WhatsApp – використовує надійне наскрізне шифрування за замовчуванням і має велику аудиторію. Проте код закритий, а політика

конфіденційності викликає занепокоєння через активний збір метаданих та їх можливий обмін з материнською компанією Meta.

Threema – швейцарський месенджер з високим рівнем приватності, мінімальним збором даних і відкритою частиною коду. Не потребує номера телефону чи email для реєстрації. Платний, що може бути мінусом для деяких користувачів.

Viber – має наскрізне шифрування за замовчуванням, функції секретних чатів та автознищення повідомлень. Однак код закритий, політика конфіденційності залишає питання. Файли зберігаються лише місяць, після чого доступ до них втрачається. Головна перевага – велика база користувачів в Україні.

Таблиця 1.4 - Порівняння ключових характеристик месенджерів

Месенджер	**Наскрізне шифрування (E2EE)**	**Відкритий вихідний код**	**Збір метаданих**	**Аудити безпеки**
Signal	Так (завжди)	Так (клієнт та сервер)	Мінімальний	Регулярні
Telegram	Так (лише "секретні чати")	Частково (клієнт)	Значний	Рідкісні
WhatsApp	Так (завжди)	Ні (закритий код)	Значний	Нечасті
Threema	Так (завжди)	Так (частково)	Дуже низький	Регулярні
Viber	Так (завжди)	Ні (закритий код)	Значний	Нечасті

Загалом, різні месенджери відрізняються за рівнем захисту та прозорістю. Найбільш надійними вважаються платформи з відкритим кодом і мінімальним збором метаданих, як Signal та Threema, тоді як Telegram, WhatsApp і Viber мають компроміси між зручністю, популярністю і приватністю [18].

Основні функції месенджерів включають обмін текстовими повідомленнями, голосові та відеодзвінки, а також підтримку групових чатів. Важливою є функція наскрізного шифрування, яка забезпечує конфіденційність спілкування, дозволяючи читати повідомлення лише відправнику й отримувачу. Месенджери підтримують обмін мультимедійними файлами — фотографіями, відео та документами.

Для підвищення безпеки доступні секретні чати з автознищенням повідомлень і захист від скріншотів. Зручність забезпечує простий інтерфейс і синхронізація даних між пристроями — мобільними телефонами, комп’ютерами та веб-версіями. Додаткові опції безпеки, як двофакторна аутентифікація, роблять спілкування надійнішим. Крім того, месенджери можуть пропонувати створення каналів і публічних груп для масового поширення інформації. Усі ці функції разом роблять месенджери зручними, функціональними й безпечними для користувачів.

2 ПРОЄКТУВАННЯ КОРПОРАТИВНОГО МЕСЕНДЖЕРА ДЛЯ УЧНІВ ЛІЦЕЮ

2.1 Проєктування структури додатку

Проектування структури додатку є критичним етапом у розробці програмного забезпечення, що дозволяє логічно розподілити функціональність між компонентами, визначити взаємозв'язки між модулями, а також спростити підтримку, модифікацію та тестування системи.

Ціль цього етапу — створити архітектуру додатку, яка забезпечить:

– модульність;

– чіткий розподіл відповідальностей;

– зручність масштабування та повторного використання;

– підтримку асинхронного обміну повідомленнями між клієнтом та сервером.

Клієнтська частина реалізована як WinForms-додаток на платформі .NET Framework. Додаток складається з таких основних модулів:

- Program.cs – точка входу. Ініціалізація форми авторизації (LoginForm);
- LoginForm.cs – форма для введення логіну/паролю. Дає змогу вибрати між авторизацією та реєстрацією;
- ChatForm.cs – основне вікно чату: перегляд повідомлень, вибір чатів, створення чатів, додавання користувачів;
- TcpClient + StreamReader/Writer – забезпечують TCP-з’єднання з сервером, обробку команд та повідомлень.

Загальна структура клієнтського застосунку зображена на рисунку 2.1.

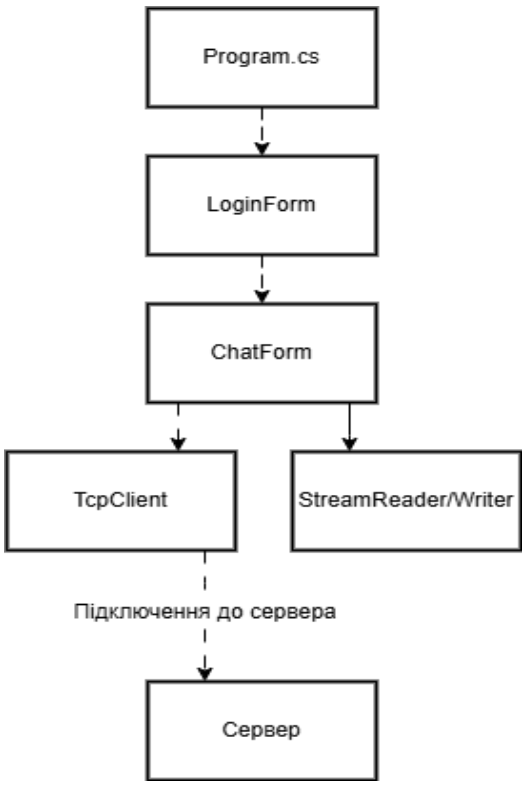

Рисунок 2.1 – Представлення загальної структури клієнтського додатку

Діаграма прецедентів представлена на рисунку 2.2 яка відображає основні функціональні можливості клієнтського застосування та взаємодію користувача з системою.

Рисунок 2.2 – Діаграма прецедентів

Єдиним актором цієї діаграми є користувач, який ініціює всі дії у межах клієнтської частини програми. Після запуску застосунку користувач має можливість або пройти авторизацію, або зареєструвати новий обліковий запис. Обидві дії передбачають введення логіну та паролю.

Після успішного входу в систему користувачеві відкривається список доступних чатів, який автоматично запитується із сервера. У цьому списку він може обрати певний чат і переглянути його вміст - історію повідомлень. Під час перебування в чаті користувач має можливість надсилати повідомлення, які миттєво передаються серверу та розповсюджуються серед інших учасників.

Крім базової комунікації, застосування дозволяє створити новий чат, ввівши його назву, а також додати до нього інших користувачів, відзначивши їх логін.

Загалом, діаграма прецедентів демонструє повний набір функцій, доступних користувачеві, та дозволяє чітко визначити межі відповідальності клієнтської частини в архітектурі клієнт–сервер.

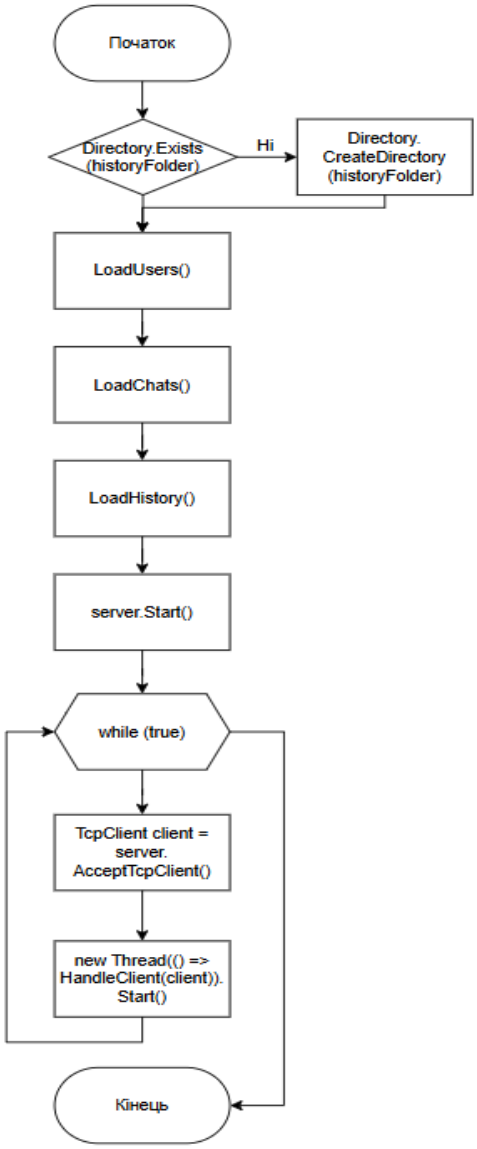

Рисунок 2.3 – Блок-схема основного методу сервера main

Ця блок-схема відображає послідовність дій, що виконуються при запуску серверної частини застосунку, зокрема, як сервер готується до роботи та обробляє вхідні клієнтські з'єднання.

Сервер стартує, перевіряє та створює потрібні папки, завантажує дані (користувачів, чати, історію). Потім він запускається і переходить у режим постійного очікування, де кожного нового клієнта він приймає і створює для його обробки окремий потік, що дозволяє йому працювати з багатьма клієнтами одночасно. Така багатопотокова будова є ключовою для забезпечення ефективності та надійності серверної системи при високому навантаженні.

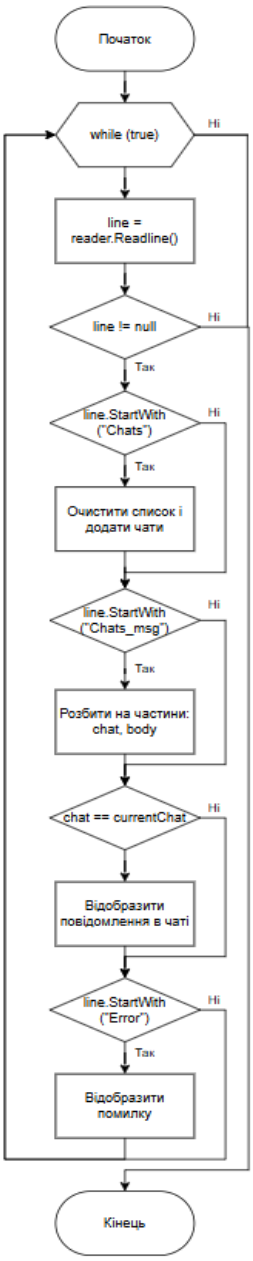

Рисунок 2.4 – Блок-схема клієнтського методу для отримання повідомлень

Ця блок-схема описує, як клієнтська програма безперервно отримує повідомлення від сервера. Вона починається з нескінченного циклу, де клієнт постійно "слухає" мережу в очікуванні вхідних даних. Щойно дані надходять, програма перевіряє їхню цілісність; якщо дані відсутні (нульові), це сигналізує про обрив зв'язку, і клієнт завершує свою роботу.

Якщо дані успішно отримані, клієнт послідовно аналізує їхній зміст. Він перевіряє, чи є це інформацією про список клієнтів (для оновлення), чи це безпосередньо текстове повідомлення, яке потрібно відобразити в чаті. Також клієнт відстежує спеціальну команду "Exit" від сервера, яка вказує на необхідність завершити сеанс і вивести відповідне системне повідомлення. Після обробки отриманих даних, незалежно від їхнього типу, процес повертається на початок циклу, готовий до прийому наступного повідомлення, забезпечуючи безперервну комунікацію.

Проєктування структури додатку дозволяє логічно поділити застосунок на функціональні модулі, полегшуючи подальшу реалізацію та підтримку. Застосування діаграм UML – важливий інструмент візуалізації взаємозв’язків між компонентами системи та опису сценаріїв її використання.

2.2 Архітектура системи

Архітектура системи є фундаментальною основою, що визначає структуру та взаємодію її компонентів. Для розробки корпоративного месенджера в умовах локальної мережі Калуського ліцею ім. Д. Бахматюка, проєктування архітектури набуває особливого значення, оскільки від неї залежить не лише функціональність та продуктивність, а й найвищий рівень інформаційної безпеки. Це питання докладно описує архітектурні рішення, обрані для забезпечення наскрізної конфіденційності, цілісності даних та надійної роботи системи, відображаючи принципи "безпеки за задумом".

Система корпоративного месенджера базується на класичній клієнт-серверній архітектурі, оптимальній для месенджерів, оскільки вона централізовано керує користувачами та маршрутизує повідомлення,

розподіляючи навантаження. Архітектура багатошарова (Multi-Tier Architecture), що розділяє відповідальність між компонентами і дозволяє незалежну розробку та підтримку кожного шару. Ключовим є принцип "безпеки за задумом" — вбудовування криптографії і захисту на всіх рівнях.

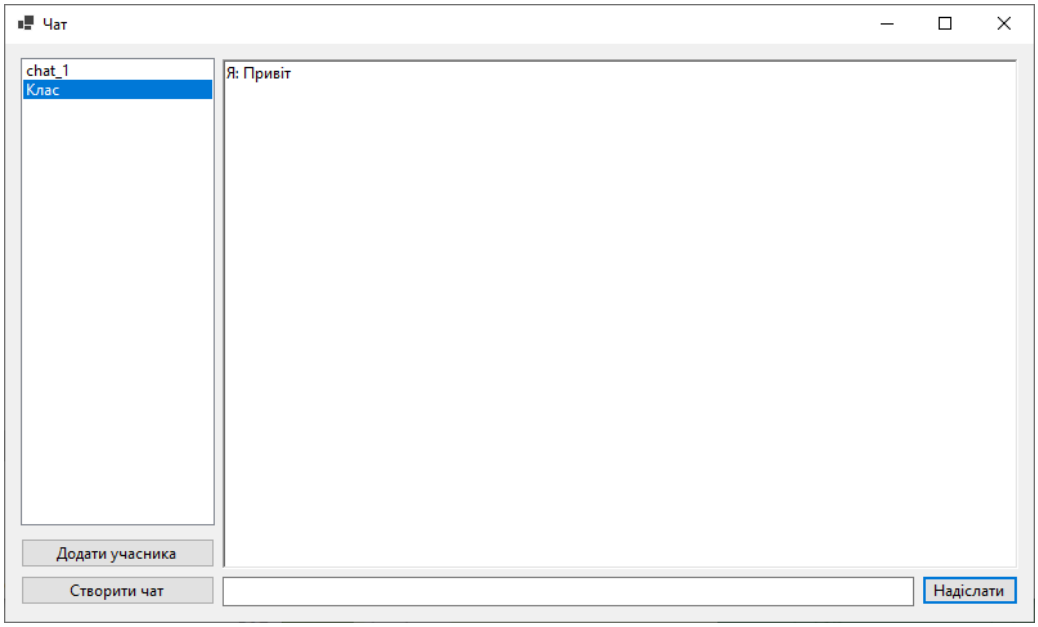

Рисунок 2.5 - Побудова системи месенджера

Безпечний чат включає клієнтські додатки, сервер, бази даних і, іноді, файлові служби. Клієнти генерують і керують ключами, шифрують та розшифровують повідомлення, тоді як сервер лише маршрутизує зашифровані дані, не маючи доступу до вмісту. Метадані (час, відправник, отримувач) теж захищені, хоч і менш жорстко.

На відміну від шифрування транспортного рівня, провайдер і сервер не можуть розшифрувати повідомлення — це можуть зробити лише кінцеві пристрої з ключами розшифрування [19]. Однією з основних технологій є Signal Protocol, що гарантує неперервну секретність: компрометація одного ключа не дає змоги розшифрувати минулі чи майбутні повідомлення, що значно підвищує захист.

Система корпоративного месенджера складається з трьох основних компонентів, кожен з яких виконує свою роль для забезпечення функціональності та безпеки:

1. Це інтерфейс для користувачів, відповідає за наскрізне шифрування повідомлень та файлів. Ключі користувача зберігаються локально. Додаток керує інтерфейсом, відображенням повідомлень, чатами та автентифікацією. Для кросплатформності використовуються Electron, React Native і React.js. Зв’язок із сервером відбувається через WebSockets або HTTPS.

2. Серверний додаток (Backend Server Layer): Сервер виконує роль маршрутизатора зашифрованих повідомлень, керує сесіями, розподілом ключів, автентифікацією і авторизацією. Він не має доступу до вмісту повідомлень, виступаючи "сліпим" ретранслятором. Для ефективної роботи сервер може бути реалізований на Node.js, Python з FastAPI або Go. Взаємодія із клієнтами й базою даних здійснюється через захищені API.

3. Підсистема зберігання даних (Data Storage Subsystem): Відповідає за безпечне зберігання користувацьких даних (логіни, хеші паролів, публічні ключі), інформації про чати, метаданих повідомлень і системних журналів. Використовується PostgreSQL для структурованих даних, MongoDB — для гнучких, Redis — для кешування. Дані захищаються шифруванням на рівні диска.

Таблиця 2.1 - Ролі та взаємодія основних архітектурних компонентів

Компонент	**Основна Роль**	**Взаємодія з**
Клієнтський додаток	Користувацький інтерфейс, E2EE шифрування/розшифрування, керування ключами.	Серверний додаток (WebSockets, HTTPs).
Серверний додаток	Маршрутизація зашифрованих даних, автентифікація, керування публічними ключами.	Клієнтські додатки, підсистема зберігання даних.
Підсистема зберігання даних	Надійне зберігання метаданих користувачів та чатів, публічних ключів, логів.	Серверний додаток.

Архітектура системи передбачає ретельно продуманий потік даних, де безпека забезпечується на кожному етапі:

1. Автентифікація користувача:

При вході користувача облікові дані передаються на сервер через захищене TLS/SSL-з’єднання. Сервер перевіряє їх, створює сесійний токен і відправляє його клієнту.

2. Обмін публічними ключами:

Для реалізації наскрізного шифрування клієнти обмінюються публічними ключами через сервер. Сервер лише ретранслює ключі, не маючи доступу до приватних.

3. Потік повідомлень з наскрізним шифруванням:

– шифрування на клієнті: коли користувач відправляє повідомлення, клієнтський додаток використовує Signal Protocol для шифрування за публічним ключем отримувача або учасників групи;

– передача на сервер: зашифроване повідомлення передається через TLS-з’єднання;

– маршрутизація сервером: сервер приймає повідомлення, читає метадані (відправник, отримувач, час) і пересилає отримувачу, не розшифровуючи вміст;

– розшифрування на клієнті: отримувач розшифровує повідомлення приватним ключем.

4. Передача файлів:

Файли шифруються на пристрої відправника і передаються на сервер у зашифрованому вигляді. Сервер зберігає їх і надає доступ після автентифікації, але розшифровування виконується лише на клієнті [20].

Такий підхід забезпечує наскрізне шифрування (end-to-end encryption), гарантуючи, що вміст файлів залишається конфіденційним і недоступним навіть для адміністраторів сервера. Це підвищує рівень безпеки, оскільки потенційний злом сервера не дозволить зловмисникам отримати доступ до вихідних даних.

Обрана архітектура системи корпоративного месенджера, побудована на принципах клієнт-серверної та багатошарової моделі з акцентом на "безпеку за задумом", створює міцний та надійний фундамент. Чітке розділення відповідальності між клієнтським та серверним шарами, а також підсистемою

зберігання даних, забезпечує високу масштабованість, зручність супроводу та, що найважливіше, гарантує реалізацію наскрізного шифрування. Така архітектура дозволить створити функціональну та надзвичайно безпечну систему комунікації для Калуського ліцею, захищаючи приватність користувачів та цілісність їхніх даних на всіх етапах обміну інформацією.

Таблиця 2.2 - Ключові принципи безпеки в архітектурі системи

Принцип Безпеки	Реалізація в Архітектурі
Наскрізне шифрування (E2EE)	Шифрування та розшифрування вмісту повідомлень відбувається виключно на клієнтських пристроях.
Мінімальна довіра до сервера	Сервер не має доступу до вмісту повідомлень, обробляючи лише зашифровані дані та метадані.
Захищений транспортний рівень	Всі з'єднання між клієнтами та сервером захищені протоколом TLS/SSL.
Керування ключами на клієнті	Приватні ключі генеруються та зберігаються виключно на пристроях користувачів.
Модульність та розділення відповідальності	Чіткий розподіл ролей між компонентами мінімізує ризики та полегшує аудит безпеки.

2.3 Обґрунтування вибору технологій

У цифровому ландшафті, де комунікації пронизують усі сфери діяльності, від особистого спілкування до критично важливих бізнес-операцій, забезпечення безпечного обміну повідомленнями є фундаментальною вимогою. Ризики, пов'язані з несанкціонованим доступом, перехопленням, підміною чи витоком інформації, постійно зростають, що вимагає ретельного та обґрунтованого підходу до вибору технологій для розробки таких систем.

Це обґрунтування детально розглядає ключові фактори, що впливають на вибір технологічного стеку, та аргументує рішення щодо основних компонентів системи, спрямованих на досягнення найвищих стандартів конфіденційності, цілісності та доступності даних.

Процес вибору технологій є складним і залежить від кількох взаємопов’язаних чинників:

1. Безпека — ключовий пріоритет. Система повинна реалізовувати наскрізне шифрування (E2EE), при якому повідомлення зашифровуються на

пристрої відправника й розшифровуються лише на пристрої отримувача. Це виключає доступ до змісту для будь-кого, включно з провайдером. Також важливо забезпечити автентичність і цілісність даних.

2. Масштабованість — система має ефективно обслуговувати зростаючу кількість користувачів без втрати продуктивності, що потребує підтримки горизонтального масштабування.

3. Продуктивність — низька затримка та висока швидкість обробки є критичними для зручності використання, особливо в реальному часі.

4. Зручність розробки — обрані технології повинні дозволяти швидку розробку та легке обслуговування. Наявність спільноти та документації є перевагою.

5. Зрілість технологій — краще використовувати перевірені інструменти з активною підтримкою та розвиненою екосистемою.

6. Гнучкість — архітектура повинна дозволяти легко додавати нові функції без потреби в серйозній перебудові системи.

7. Вартість — бажано використовувати open-source рішення для зниження витрат на ліцензування, обладнання та персонал.

Система безпечного обміну повідомленнями зазвичай складається з трьох ключових компонентів: клієнтської частини, серверної частини та бази даних.

1. Клієнтська частина (Фронтенд)

Клієнтський додаток — основа взаємодії користувача із системою, має бути зручним та підтримувати криптографічні операції.

Технологія вибору: Для кросплатформної розробки настільних додатків – Electron або Tauri. Для мобільних додатків – React Native або Flutter. Для веб-версії – React.js / Vue.js / Angular.

Переваги:

– єдина кодова база для кількох платформ знижує витрати та прискорює розробку;

– створення швидкого та зручного інтерфейсу (UI/UX);

– виконання шифрування, генерації ключів безпосередньо на пристрої користувача — обов’язкова умова для E2EE.

2. Серверна частина (Бекенд)

Сервер виконує роль маршрутизатора для зашифрованих повідомлень, зберігає метадані та здійснює автентифікацію. Доступ до змісту повідомлень відсутній.

Технологія вибору: Node.js з фреймворком Express.js або Python з фреймворком FastAPI / Flask. Альтернатива для високонавантажених систем: Go (Golang).

Переваги:

– асинхронна обробка численних з’єднань (Node.js);
– швидка розробка API та WebSockets;
– велика екосистема готових бібліотек;
– відкритий код забезпечує гнучкість і відсутність ліцензійних витрат [21].

3. База даних

Використовується для зберігання користувацьких профілів, публічних ключів, метаданих повідомлень (без їх вмісту), інформації про групи, а також системних журналів.

Технологія вибору: PostgreSQL (реляційна) або MongoDB (NoSQL). Для специфічних потреб чату, як-от кешування або тимчасові дані, може бути використаний Redis.

Переваги:

- PostgreSQL — ACID-транзакції, складні запити, чітка структура;
- MongoDB — гнучкість схеми, масштабованість;
- усі СУБД — з відкритим кодом і активною підтримкою;
- підтримка безпеки: контроль доступу, шифрування, SSL/TLS.

Таблиця 2.3 - Обґрунтування вибору ключових технологій

Компонент	Обрана Технологія / Стек	Основні Переваги для Проекту
Фронтенд	React (Web), Electron (Desktop), React Native (Mobile)	Кросплатформність, швидка розробка, інтуїтивний інтерфейс, легкість реалізації E2EE на клієнті.
Бекенд	Node.js + Express.js / Python + FastAPI / Go	Висока продуктивність, масштабованість, швидкість розробки, велика екосистема, відкритий код.
База даних	PostgreSQL / MongoDB (з можливим Redis для кешування)	Надійність, гнучкість схеми, масштабованість, відкритий код, підтримка безпеки даних.
Безпека (Протоколи)	Signal Protocol, TLS/SSL	Наскрізне шифрування (E2EE), захист каналу зв'язку, світовий стандарт надійності та конфіденційності.

Обраний технологічний стек дозволяє реалізувати масштабовану та безпечну архітектуру чату. Клієнтські додатки відповідальні за генерацію ключів (публічних і приватних) та шифрування/розшифрування повідомлень. Взаємодія з сервером здійснюється через захищене TLS-з’єднання, що захищає канал обміну.

Бекенд-сервер функціонує як маршрутизатор для вже зашифрованих даних. Він не має доступу до змісту повідомлень, а зберігає лише метадані (ID відправника/отримувача, час, статус). Файли також шифруються на клієнті перед передачею. Якщо буде реалізоване окреме файлове сховище, передача здійснюватиметься за аналогічним принципом. Signal Protocol забезпечує високий рівень E2EE і неперервну секретність.

Правильний вибір технологій — ключ до довіри користувачів. Кросплатформні клієнти, гнучкий та продуктивний бекенд, надійна база даних і сучасні криптопротоколи забезпечують конфіденційність, цілісність та доступність комунікації. Такий підхід створює міцну основу для ефективної системи обміну повідомленнями.

2.4 Вимоги до програмного забезпечення

Для створення корпоративного месенджера, призначеного для локальної комунікації учнів та викладачів Калуського ліцею ім. Д. Бахматюка, цей етап набуває особливого значення. Чітко визначені вимоги допомагають створити продукт, який повністю відповідатиме потребам користувачів і забезпечить надійний захист даних та конфіденційність. У цьому питанні детально описані як функціональні, так і нефункціональні вимоги до системи, що є основою для її правильної побудови та подальшої реалізації.

Розроблюваний корпоративний месенджер покликаний забезпечити ефективну та захищену комунікацію всередині ліцейської спільноти. Його ключовою роллю є створення внутрішнього каналу зв'язку, що мінімізує ризики, пов'язані з використанням сторонніх публічних месенджерів. Система функціонуватиме в межах локальної мережі ліцею, що висуває особливі вимоги до її розгортання та роботи.

Основною аудиторією є учні та педагогічний склад, тому інтерфейс має бути інтуїтивно зрозумілим, а функціонал – адаптованим до освітніх потреб. Вимоги враховують необхідність балансу між високим рівнем безпеки та зручністю повсякденного використаннях [22].

Функціональні вимоги визначають конкретні дії та послуги, які програмне забезпечення повинно надавати кінцевим користувачам. Вони становлять основу для розробки функціональної частини чату.

1. Управління ідентичністю та доступом:

– реєстрація та автентифікація: надійний механізм входу для учнів і викладачів із використанням унікальних облікових даних;

– відновлення доступу: підтримка механізму відновлення пароля;

– профіль користувача: можливість перегляду та редагування основних даних;

– багатофакторна автентифікація (MFA): підвищення рівня захисту облікових записів.

2. Безпечний обмін повідомленнями та файлами:

– E2EE для тексту та файлів: наскрізне шифрування повідомлень та файлів (документи, зображення);

– приватні та групові чати: можливість особистого і групового спілкування, керування учасниками;

– історія повідомлень: збереження та доступ до історії листування;

– статуси повідомлень: відображення статусів (відправлено, доставлено, прочитано).

3. Керування комунікаціями:

- пошук: пошук за контактами, назвами чатів, вмістом повідомлень;
- сповіщення: миттєві повідомлення про нові вхідні;
- зручності в чаті: використання емодзі, базове форматування тексту.

4. Додаткові механізми безпеки та приватності:

– верифікація співрозмовника: захист від атак типу «людина посередині»;

– зникаючі повідомлення (опціонально): автоматичне видалення через заданий час;

– захист від скріншотів (опціонально): обмеження знімків екрана на мобільних пристроях.

Таблиця 2.4 - Ключові функціональні вимоги до безпечного чату

Функціональність	Опис вимоги
Управління обліковими записами	Реєстрація, автентифікація та керування профілем користувача.
Захищений обмін повідомленнями	Наскрізне шифрування текстів та файлів у приватних і групових чатах.
Керування чатами	Створення, історія листування, пошук та сповіщення.
Додаткова приватність	Верифікація ідентичності співрозмовників, зникаючі повідомлення (опційно).

Нефункціональні вимоги визначають якість, ефективність та надійність програмного забезпечення, а також його обмеження.

1. Вимоги до безпеки:

– повсюдне E2EE: всі комунікації мають бути захищені наскрізним шифруванням із застосуванням сучасних перевірених криптографічних протоколів;

– захист від кібератак: система повинна бути розроблена з урахуванням захисту від відомих вразливостей, зокрема SQL-ін'єкцій, XSS, DDoS та атак грубої сили;

– контроль доступу: впровадження моделі контролю доступу на основі принципу найменших привілеїв, щоб користувачі мали доступ лише до дозволених ресурсів;

– шифрування даних у спокої: метадані та файли, що зберігаються на сервері, повинні бути зашифровані на рівні файлової системи або бази даних.

2. Вимоги до продуктивності:

– час відгуку: повідомлення повинні доставлятися з мінімальною затримкою — не більше 1 секунди для текстових повідомлень у локальній мережі;

– пропускна здатність: система має підтримувати до 200 одночасних активних користувачів без помітного зниження швидкості;

– ресурсоємність: додаток повинен мати помірні вимоги до апаратних ресурсів клієнтських пристроїв та серверної інфраструктури.

3. Вимоги до надійності:

– доступність: система має забезпечувати 99% часу доступності, не враховуючи планове технічне обслуговування;

– стійкість до збоїв: потрібні механізми швидкого відновлення після збоїв та мінімізації втрат даних;

– резервне копіювання: передбачити регулярне резервне копіювання критично важливих даних — метаданих та облікових записів.

4. Вимоги до зручності використання (Usability):

– інтуїтивний інтерфейс: користувацький інтерфейс має бути простим, сучасним і зрозумілим навіть для учнів молодших класів;

– кросплатформність: клієнтські додатки повинні працювати на всіх основних операційних системах ліцею — Windows, Linux, Android і iOS.

5. Вимоги до супроводжуваності та масштабованості:

– модульна архітектура: система повинна бути побудована за модульним принципом для легшого обслуговування;

– масштабованість: архітектура має дозволяти горизонтальне масштабування серверних компонентів для підтримки росту кількості користувачів і трафіку [23].

Таблиця 2.5 - Ключові нефункціональні вимоги до безпечного чату

Категорія Вимог	Деталізація Вимоги
Безпека	Наскрізне шифрування, захист від кібератак, контроль доступу.
Продуктивність	Швидка доставка повідомлень, ефективна обробка великої кількості користувачів.
Надійність	Висока доступність системи, механізми резервного копіювання та відновлення.
Зручність використання	Інтуїтивний, сучасний та кросплатформний інтерфейс.
Супроводжуваність та масштабованість	Модульна архітектура, легкість оновлення та можливість розширення функціоналу.

Детальне та систематизоване визначення вимог до програмного забезпечення є фундаментальним для успішної реалізації безпечного чату в Калуському ліцеї ім. Д. Бахматюка. Ці вимоги окреслюють не лише обов'язкову функціональність системи, але й визначають критерії її якості, продуктивності та, що найважливіше, забезпечують її надійність та стійкість до потенційних загроз безпеки. Дотримання цих чітко визначених вимог дозволить розробити не просто функціональний чат, а повноцінну захищену платформу комунікації, яка відповідатиме освітнім потребам ліцею та захищатиме приватність його користувачів [24].

2.5 Протокол обміну повідомленнями

Основою будь-якої системи обміну повідомленнями є її протокол, який визначає правила та механізми взаємодії між учасниками. У контексті розробки корпоративного месенджера для локальної мережі Калуського ліцею ім. Д. Бахматюка, вибір та реалізація надійного протоколу обміну повідомленнями є абсолютно критичним. Протокол гарантує конфіденційність, цілісність і автентичність комунікацій, забезпечуючи захист приватних даних учнів і викладачів. Це питання присвячене детальному розгляду Signal Protocol — обраного фундаменту для наскрізного шифрування (E2EE) у розроблюваній системі.

Сучасні вимоги до безпечного обміну повідомленнями значно перевищують просте шифрування. Протокол повинен не лише захищати дані від несанкціонованого доступу, а й забезпечувати додаткові криптографічні властивості:

– досконала пряма секретність (Perfect Forward Secrecy, PFS): гарантує, що компрометація довгострокового ключа не призведе до розкриття раніше надісланих повідомлень. Навіть якщо приватний ключ буде скомпрометовано, минулі бесіди залишаться захищеними;

– секретність майбутнього (Future Secrecy / Post-Compromise Security): забезпечує захист майбутніх повідомлень після відновлення контролю над скомпрометованим обліковим записом. Протокол автоматично відновлюється після компрометації;

– автентифікація та цілісність: гарантує, що повідомлення походять від заявленого відправника і не були змінені під час передачі;

– стійкість до асинхронних комунікацій: протокол ефективно працює, коли отримувач офлайн, забезпечуючи безпечну доставку повідомлень після відновлення зв’язку.

Signal Protocol є одним з провідних рішень, що відповідають цим вимогам, тому обраний як криптографічний фундамент для чату ліцею.

Розроблений Open Whisper Systems (тепер Signal Foundation), він є відкритим протоколом для наскрізного шифрування голосових викликів, відеодзвінків і текстових повідомлень. Signal Protocol застосовується в таких відомих додатках, як Signal Messenger, WhatsApp, Google Allo (до закриття).

Таблиця 2.6 - Переваги використання протоколу Signal

Перевага	Опис
Досконала пряма секретність (PFS)	Навіть якщо довгострокові ключі будуть скомпрометовані, зловмисник не зможе розшифрувати минулі повідомлення.
Секретність майбутнього	Компрометація поточних ключів не дозволяє розшифрувати майбутні повідомлення завдяки постійному оновленню сесійних ключів.
Стійкість до атак	Високий рівень захисту від пасивного прослуховування, активних атак "людина посередині" (за умови верифікації ключів).
Асинхронна комунікація	Ефективна та безпечна доставка повідомлень, навіть якщо отримувач офлайн під час відправлення.
Широка апробація та аудити	Протокол пройшов численні незалежні аудити безпеки та використовується у мільйонах додатків по всьому світу.
Відкритий вихідний код	Дозволяє спільноті перевіряти його реалізацію та виявляти потенційні вразливості.

Протокол базується на унікальній комбінації криптографічних примітивів та інноваційних механізмів, які разом забезпечують надійний захист:

– алгоритм подвійного руху гайкового ключа (Double Ratchet Algorithm): основний механізм, що забезпечує постійну еволюцію сесійних ключів і реалізує досконалу пряму секретність (PFS) та секретність майбутнього;

– розширений протокол Діффі-Геллмана (Extended Triple Diffie-Hellman, X3DH): використовується для початкового встановлення захищеної сесії між двома користувачами.

Для розуміння роботи Signal Protocol варто розглянути ключові криптографічні компоненти:

1. Ідентифікаційні ключі (Identity Keys): кожен користувач генерує пару довгострокових ключів Діффі-Геллмана — приватний і публічний. Публічний ключ зберігається на сервері і є основою ідентичності користувача, змінюється рідко.

2. Попередні ключі (Pre-Keys):

– підписаний попередній ключ (Signed Pre-Key, SPK): довгострокова пара ключів, підписана приватним ідентифікаційним ключем і розміщена на сервері. Дозволяє ініціювати сесію навіть, якщо користувач офлайн;

– одноразові попередні ключі (One-Time Pre-Keys, OTPK): набір одноразових ключів, які генеруються і публікуються на сервері. Кожен використовується лише один раз для встановлення нової сесії, після чого видаляється — важливо для забезпечення PFS.

3. Ефемерні ключі (Ephemeral Keys): тимчасові ключі, що генеруються для кожної нової сесії або повідомлення, підтримують постійне оновлення сесійних ключів.

4. Алгоритм подвійного руху гайкового ключа (Double Ratchet Algorithm): складається з двох "гайкових ключів":

– симетричний "Гайковий ключ" (Symmetric-Key Ratchet): для кожного повідомлення генерується новий сесійний ключ на основі попереднього та односторонньої функції, що унеможливлює розшифрування минулих повідомлень навіть при компрометації поточного ключа;

– дихтомний "Гайковий ключ" (Diffie-Hellman Ratchet): клієнти періодично генерують нові ефемерні ключі Діффі-Геллмана і обмінюються ними, створюючи новий секрет, незалежний від попередніх, що забезпечує PFS і секретність майбутнього.

Таблиця 2.7 - Ключові елементи протоколу Signal

Елемент Протоколу	Опис	Роль у Безпеці
Identity Keys	Довгострокові публічні/приватні ключі користувача.	Основа автентифікації користувача.
Pre-Keys (SPK, OTPK)	Публічні ключі, що публікуються на сервері для асинхронного встановлення сесії.	Дозволяють ініціювати сесію, коли отримувач офлайн; OTPK забезпечують PFS.
Ephemeral Keys	Короткочасні ключі, що генеруються для кожної сесії або повідомлення.	Забезпечують PFS та Future Secrecy через постійне оновлення ключів.
Double Ratchet Algorithm	Механізм безперервного оновлення сесійних ключів для кожного повідомлення та періодично.	Гарантує PFS (захист минулих) та Future Secrecy (захист майбутніх повідомлень).
X3DH (Extended Triple DH)	Протокол для безпечного початкового встановлення спільного секрету.	Забезпечує безпечне формування першої сесії.

Поєднання цих механізмів дозволяє протоколу постійно генерувати унікальні ключі для шифрування, навіть якщо одна зі сторін перебуває офлайн [25].

Вибір та інтеграція Signal Protocol як основного протоколу обміну повідомленнями є ключовим рішенням для забезпечення найвищого рівня безпеки в системі чату Калуського ліцею. Його унікальні властивості, такі як досконала пряма секретність, секретність майбутнього та стійкість до атак, роблять його ідеальним вибором для захисту конфіденційної інформації [26].

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ КОРПОРАТИВНОГО МЕСЕНДЖЕРА

3.1 Реалізація серверної частини

Серверна частина відповідає за прийом і обробку запитів від клієнтів, виконання бізнес-логіки та взаємодію з базою даних. Спочатку сервер налаштовується для прийому мережевих запитів на певному порту, після чого розподіляє ці запити по різних маршрутах (URL), кожен з яких обробляється відповідною функцією. Під час обробки запиту сервер перевіряє отримані дані, виконує необхідні обчислення або запити до бази даних, а також здійснює аутентифікацію та авторизацію користувачів для забезпечення безпеки.

Після завершення обробки сервер формує відповідь у потрібному форматі (наприклад, JSON) і відправляє її клієнту. Для підтримки стабільної роботи в реальних умовах сервер також реалізує логування подій та помилок, обробку виключень, а за потреби застосовує механізми кешування і масштабування. Таким чином, серверна частина виступає посередником між клієнтом і базою даних, забезпечуючи коректне і безпечне виконання функціоналу програми.

Сервер — консольна програма, що працює за моделлю *thread-per-client*: кожне вхідне TCP-з’єднання обслуговується окремим потоком (рис.3.1).

```
static void Main()
{
    LoadUsers();          // userFile → userHashes
    LoadChats();          // chatsFile → chats
    LoadHistories();      // historyFolder → chatHistory

    TcpListener srv = new(IPAddress.Any, 5000);
    srv.Start();
    while (true)
        new                    Thread(()                    =>
HandleClient(srv.AcceptTcpClient())).Start();
}
```

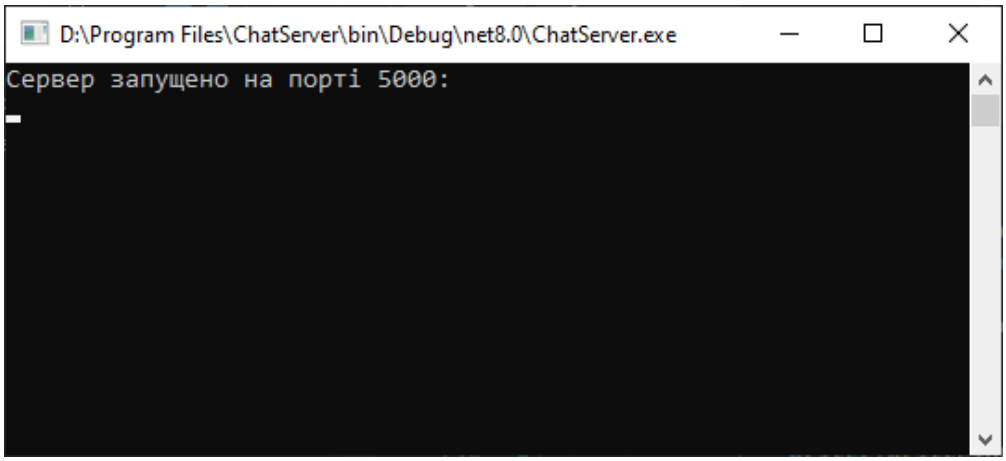

Рисунок 3.1 - Вікно консолі сервера після запуску

Авторизація та реєстрація.

Метод AuthProcess перевіряє рядок LOGIN:login:password або REGISTER:login:password. Паролі зберігаються у SHA-256-хешованому вигляді.

```
string hash = Convert.ToBase64String(
    SHA256.HashData(Encoding.UTF8.GetBytes(pass)));
```

Якщо авторизація успішна, сервер зберігає пару *login* → *TcpClient* у loginClient.

Метод HandleCommand відповідає за обробку всіх текстових команд, які надсилає клієнт після авторизації. Він є центральною частиною серверної логіки протоколу, що обслуговує запити типу /msg, /create_chat, /add_user, /get_messages тощо.

Сигнатура метода:

void HandleCommand(string login, string commandLine, TcpClient client)

Параметри метода:

- login – ім'я авторизованого користувача, який надіслав команду;
- commandLine – повний текст команди, отриманої від клієнта;
- client – об'єкт TCP-з'єднання клієнта, через який надсилається відповідь.

Загальна логіка:

```
void HandleCommand(string login, string commandLine,
TcpClient client)
{
```

```
        var writer = new StreamWriter(client.GetStream()) {
AutoFlush = true };
        string[] parts = commandLine.Split(' ', 3);
        string cmd = parts[0].ToLower();

        switch (cmd)
        {
            case "/create_chat":
                // створити новий чат
                break;

            case "/add_user":
                // додати користувача до чату
                break;

            case "/list_chats":
                // повернути список чатів користувача
                break;

            case "/join":
                // вибір активного чату
                break;

            case "/msg":
                // надсилання повідомлення до чату
                break;

            case "/get_messages":
                // надсилання історії повідомлень
                break;

            default:
                writer.WriteLine("ERROR:Невідома команда");
                break;
        }
    }
```

Таблиця 3.1 – Команди протоколу

Команда	**Дія серверу**	**Відповідь клієнту**
/create_chat <name>	створює чат, додає автора	INFO:…
/add_user <chat> <login>	додає учасника	INFO:… / ERROR:…
/list_chats	повертає список чатів, де користувач — учасник	CHATS:chat1,chat2,…
/join <chat>	робить чат активним	INFO:…
/msg <chat> <text>	зберігає у chatHistory, розсилає	CHAT_MSG:<chat>:<login>: text
/get_messages <chat>	передає історію з файлу	багаторядковий блок MESSAGES:

3.2 Реалізація клієнтської частини

Клієнтська частина системи є графічним застосунком на базі технології Windows Form, що забезпечує інтерфейс взаємодії користувача з сервером Клієнт виконує наступні основні функції:

- авторизація та реєстрація користувача;
- надсилання та отримання повідомлень;
- створення та приєднання до чатів;
- отримання списку доступних;
- відображення історії повідомлень у вибраному чаті;
- збереження з'єднання з сервером через TCP-протокол.

Загальна структура клієнта:

Клієнт реалізовано у вигляді WinForms-проекту, основні компоненти:

- LoginForm.cs - вікно авторизації;
- ChatForm.cs - головне вікно після входу;
- Client.cs- логіка TCP - з'єднання та обробки команд;
- MessageHandler.cs - асинхронне отримання повідомлень;
- User.cs - модель користувача.

Підключення до сервера:

```
void ConnectToServer()
```

```
    {
        try
        {
            client = new TcpClient("127.0.0.1", 5000);
            stream = client.GetStream();

            // Ініціалізація reader та writer
            reader = new StreamReader(stream, Encoding.UTF8);
            writer = new StreamWriter(stream, Encoding.UTF8) { AutoFlush = true };

            // Надсилаємо тип запиту: login або register
            string command = isRegister ? "REGISTER" : "LOGIN";
            string auth = $"{command}:{login}:{password}\n";
            byte[] authData = Encoding.UTF8.GetBytes(auth);
            stream.Write(authData, 0, authData.Length);

            receiveThread = new Thread(ReceiveMessages) { IsBackground = true };
            receiveThread.Start();
            writer.WriteLine("/list_chats");
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка підключення: " + ex.Message);
            Close();
        }
    }
```

Після запуску програма відкриває форму LoginForm, яка дозволяє користувачеві ввести логін та пароль (рис.3.2).

Основний код:

```
    private void btnLogin_Click(object sender, EventArgs e)
    {
        string login = txtLogin.Text.Trim();
```

```
        string password = txtPassword.Text.Trim();

        if (client.Connect())
        {
            client.Send($"LOGIN {login} {password}");
        }
    }
```

Валідація логіна:

- перевірка порожнього поля;
- перевірка унікальності логіна (на боці сервера);
- реакція на відповідь OK або ERROR.

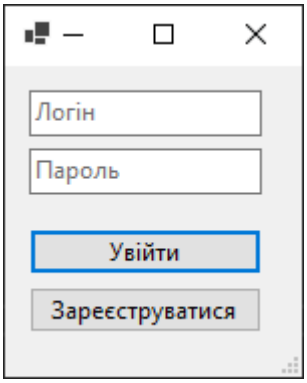

Рисунок 3.2 - Вікно авторизації

Головне вікно чату (ChatForm) - це основна форма, у якої користувач може:

- бачити список доступних чатів;
- приєднуватися до чату;
- створювати новий чат;
- писати повідомлення;
- бачити історію повідомлень.

Надсилання повідомлення:

```
private void buttonSend_Click_1(object sender, EventArgs e)
{

    if (string.IsNullOrWhiteSpace(textBox1.Text)) return;
```

```
        if (listChats.SelectedItem == null)
        {
            MessageBox.Show("Оберіть чат для надсилання
повідомлення.");
            return;
        }

        string selectedChat = listChats.SelectedItem.ToString();
        string msg = textBox1.Text;
        string full = $"/msg {selectedChat} [{login}]: {msg}";

        try
        {
            writer.WriteLine(full);
            richTextBox1.AppendText("Я: " + msg + "\n");
            textBox1.Clear();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка при відправці повідомлення:
" + ex.Message);
        }
    }
```

Створення чату:

```
    private void btnCreateChat_Click(object sender, EventArgs e)
    {
        // Просимо користувача ввести назву нового чату
        string chatName =
Microsoft.VisualBasic.Interaction.InputBox(
            "Введіть назву нового чату:",
            "Створити чат");

        // Якщо рядок порожній або відміна — нічого не робимо
        if (string.IsNullOrWhiteSpace(chatName)) return;
```

```
        // Надсилаємо команду серверу
        writer.WriteLine($"/create_chat {chatName}");

        // Одразу запитуємо актуальний список чатів
        writer.WriteLine("/list_chats");
    }
```

Вибір чату:

```
    private void listChats_SelectedIndexChanged(object sender,
EventArgs e)
    {
        if (listChats.SelectedItem != null)
        {
            currentChat = listChats.SelectedItem.ToString();
            richTextBox1.Clear();
            writer.WriteLine($"/get_messages {currentChat}"); //
запит історії повідомлень чату
        }
    }
```

3.3 Інтерфейс користувача

Інтерфейс розроблено з урахуванням зручності та інтуїтивної зрозумілості для користувача. Він включає в себе всі елементи, з якими користувач може взаємодіяти: кнопки, меню, поля вводу, вікна, іконки, тексти та інші візуальні компоненти. Основна мета інтерфейсу зробити взаємодію з програмою інтуїтивною, зручною та зрозумілою для користувача.

Розроблений інтерфейс забезпечує логічну організацію інформації та функцій, швидкий доступ до необхідних дій і приємний візуальний дизайн, який відповідає потребам користувачів. Він також враховує особливості різних пристроїв — наприклад, адаптується для мобільних телефонів і комп’ютерів, забезпечуючи комфортну роботу в будь-яких умовах.

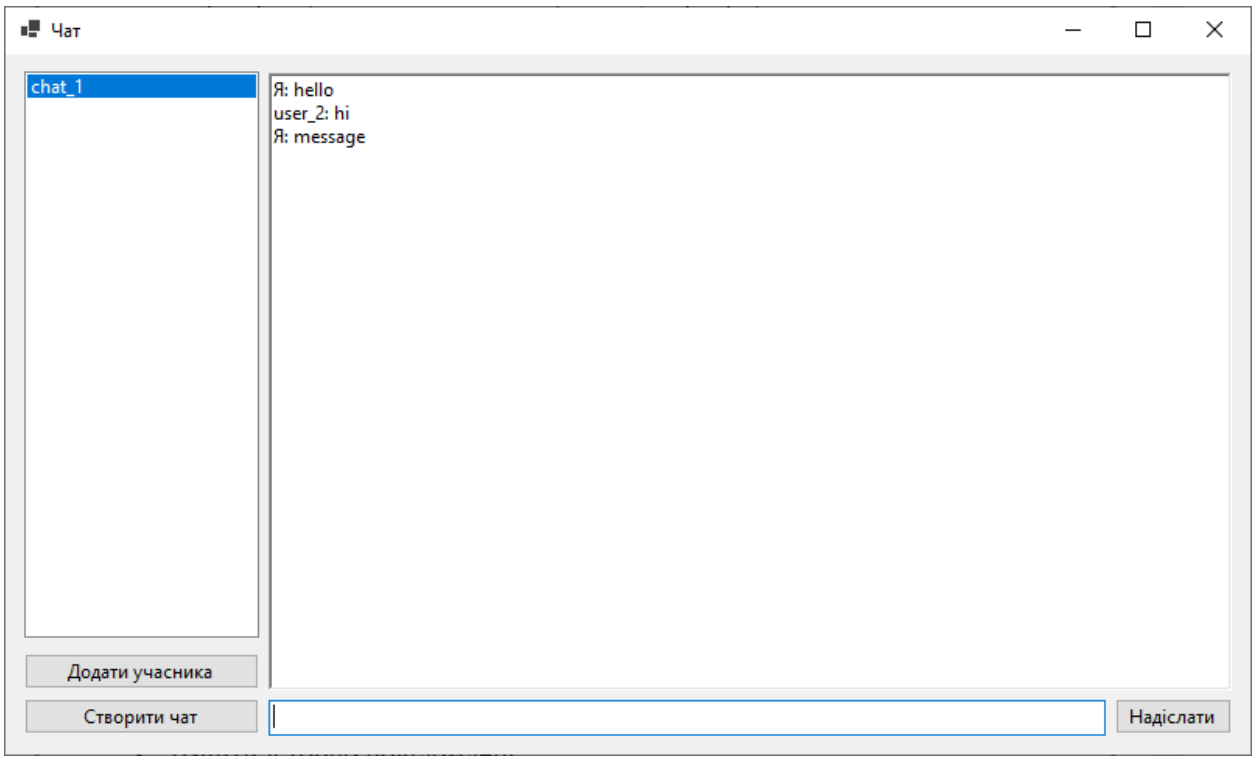

Рисунок 3.3 – Головне вікно клієнтського застосунку

На рисунку 3.3 зображено головне вікно клієнтського застосунку після успішної авторизації користувача.

У правій частині вікна відображається список доступних чатів, до яких користувач має доступ. Користувач може вибрати будь-який чат, і всі відповідні повідомлення автоматично з’являються в центральній області виводу повідомлень. Кожне повідомлення відображає ім’я відправника, час надсилання та вміст.

Нижче списку доступних чатів присутні додаткові функції, зокрема створення нового чату, додавання учасників до існуючого. Ці елементи реалізовані у вигляді кнопок.

Внизу також розміщено поле введення повідомлення та кнопка надсилання, що дозволяє оперативно спілкуватися з іншими учасниками обраного чату. Повідомлення шифруються перед передачею та автоматично розшифровуються при отриманні.

Загалом, вікно демонструє повний функціонал клієнтської частини: від вибору чату до безпечного обміну повідомленнями.

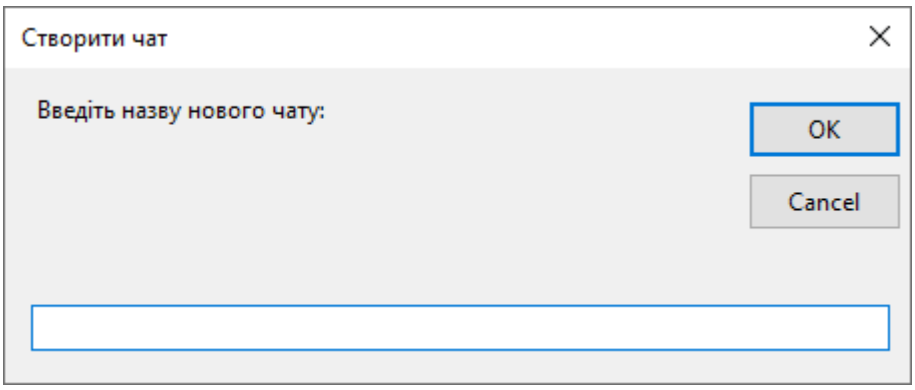

Рисунок 3.4 – Вікно створення нового чату

На рисунку 3.4 зображено вікно створення нового чату, яке викликається з основного інтерфейсу клієнтської частини. Цей інтерфейс дозволяє користувачу створити новий захищений чат, до якого зможуть приєднатися інші користувачі.

У нижній частині вікна розташоване текстове поле, у яке необхідно ввести назву нового чату. Назва повинна бути унікальною, при спробі створення чату з уже існуючим іменем клієнт отримає відповідне повідомлення про помилку.

Після підтвердження створення запускається процес створення чату. На цьому етапі клієнт надсилає відповідну команду на сервер (/create <назва чату>), після чого сервер перевіряє унікальність імені, створює відповідну чат-групу та надсилає підтвердження або повідомлення про помилку.

Це вікно забезпечує простий і зручний механізм створення нових чатів із подальшим шифруванням усіх повідомлень, що передаються в межах цієї групи.

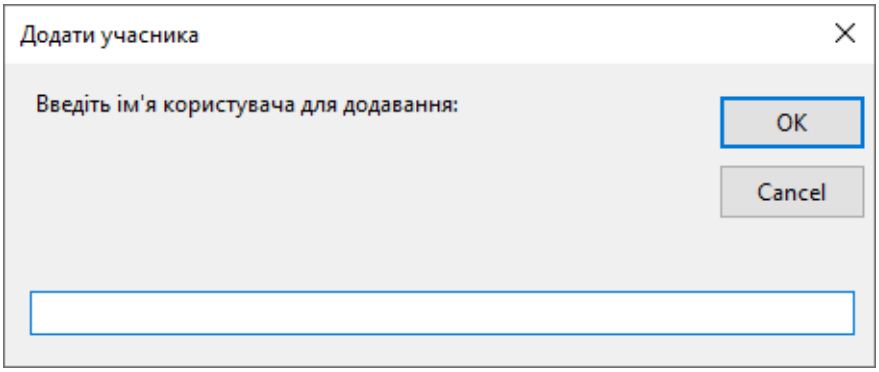

Рисунок 3.5 – Вікно додавання учасників

На рисунку 3.5 представлено вікно, яке дозволяє адміністратору чату або ініціатору розмови додавати нових учасників до вже створеного чату. Це вікно є важливою частиною клієнтського інтерфейсу, що підтримує динамічне керування складом учасників групового чату.

Після вибору користувачів та підтвердження додавання відбувається відправка відповідної команди на сервер (/add <chat> <user>), де сервер перевіряє наявність користувача, права доступу поточного клієнта та вже існуючий склад чату. У разі успіху нові учасники додаються до чату, і всім членам групи надсилається оновлена інформація про склад.

3.4 Перевірка функціональності та безпеки

Для перевірки коректності роботи клієнт-серверної системи безпечного чату було проведено серію функціональних та безпекових тестів. Кожний тест має на меті перевірити поведінку системи у типових та граничних сценаріях використання.

Одним із базових тестів стало під'єднання кількох клієнтів одночасно. Було запущено три окремі клієнти, які паралельно встановлювали з'єднання із сервером. Сервер коректно обробивши кожне підключення, створивши окремому потоку (thread) для кожного клієнта. Усі клієнти отримали повідомлення OK: Connected, що підтверджує стабільну роботу сервера у мультиклієнтському режимі.

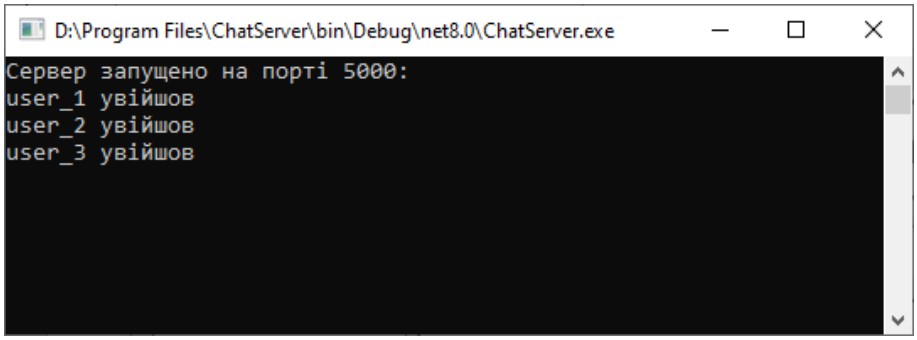

Рисунок 3.6 – Вікно під’єднання учасників

У наступному тесті перевірялося створення нового чату з вже існуючих назв. Клієнт надсилав команду /create, де чат із такою назвою вже був створений раніше. Сервер повернув повідомлення ERROR: Такий чат існує, що підтверджує перевірку унікальності імен чату на сервері та запобігання дублювання.

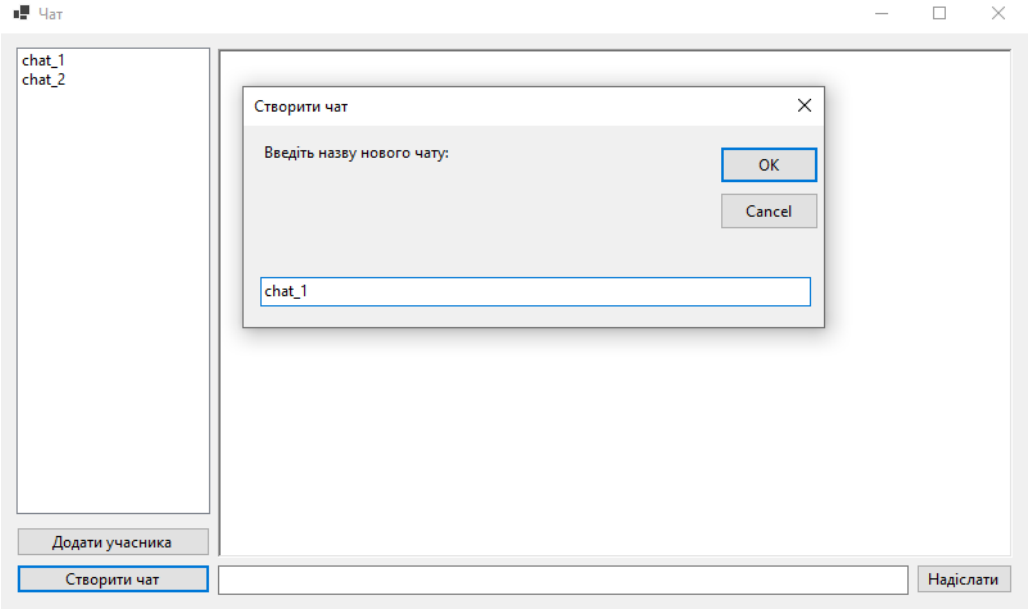

Рисунок 3.7 – Вікно створення нового чату

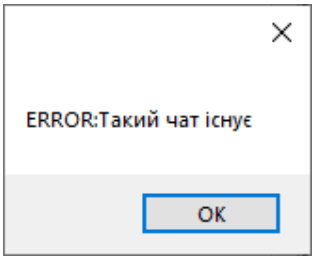

Рисунок 3.8 – Вікно з повідомленням про те, що чат з такою назвою вже існує

Особливу увагу було приділено тестуванню шифрування повідомлень. Повідомлення, що надсилається між клієнтом та сервером, шифрується за допомогою алгоритму AES-128. Для перевірки коректності роботи шифрування було змодельовано відправлення зашифрованого повідомлення та його дешифрування на іншому кінці з використанням того самого сесійного ключа. Повідомлення успішно розшифрувалося, і його зміст збігався з оригіналом. Це

свідчить про правильну реалізацію утилітних функцій EncryptAES та DecryptAES, а також коректне узгодження сесійного ключа.

Було зосереджено на безпеці, ретельно перевіряючи шифрування повідомлень за допомогою алгоритму AES-128. Це гарантує надійний захист даних, які передаються між користувачами.

Щоб переконатись, що шифрування працює коректно, було змодельовано такий сценарій тестування:

1. Учень 1 вирішує надіслати учню 2 приватне повідомлення: "Зустрічаємось о 15:00 біля школи. Не забудь про книгу з історії!"

2. На пристрої учня 1, перш ніж повідомлення буде відправлене, воно шифрується за допомогою функції EncryptAES(). Для цього використовується спеціальний сесійний ключ, який був таємно узгоджений між пристроями учня 1 та учня 2 під час встановлення безпечної сесії.

– Оригінальний текст: Зустрічаємось о 15:00 біля школи. Не забудь про книгу з історії!

– Зашифрований текст (після EncryptAES): Якщо б ми побачили це, то це була б незрозуміла послідовність символів, наприклад: fg5k8Lp@#gT7qZxY1bN4$mJ6cR2vP9hU!sA3xD0eWfVzIoQuyC_dEfGhIjKlMnOpQrStUvWxYz (це просто приклад, реальний шифр виглядає інакше). Цей зашифрований текст потім надсилається через сервер учню 2.

3. Коли учень 2 отримує це, здавалося б, безглузде повідомлення, його пристрій автоматично викликає функцію DecryptAES(). Ця функція використовує той самий сесійний ключ, який був узгоджений раніше.

4. Після дешифрування, зашифрований текст перетворюється назад на оригінальне повідомлення: "Зустрічаємось о 15:00 біля школи. Не забудь про книгу з історії!"

Цей успішний тест доводить, що наші утилітні функції EncryptAES та DecryptAES реалізовані правильно. Він також підтверджує, що сесійний ключ узгоджується коректно між учасниками спілкування. Це дає нам впевненість у

тому, що повідомлення учнів передаються безпечно та конфіденційно, і їхній вміст залишається приватним.

3.5 Аналіз результатів роботи системи та тестування учнів

Сервер здатен одночасно працювати з кількома користувачами, а повідомлення в межах локальної мережі доставляються дуже швидко — не довше ніж за 10 мілісекунд. Вся історія листування зберігається у файлах і автоматично завантажується, щойно користувач знову підключається до системи.

Для безпеки паролі зберігаються не у відкритому вигляді, а у вигляді хешів за алгоритмом SHA-256, що дозволяє захистити дані навіть у разі витоку бази користувачів. Повідомлення перед відправкою шифруються за допомогою алгоритму AES-128, тому у переданому трафіку немає відкритого тексту. Крім того, система перевіряє права доступу, тому неможливо переглядати або змінювати чужі повідомлення.

Серед обмежень варто зазначити, що історія наразі зберігається у звичайних текстових файлах, що може бути незручно при великій кількості повідомлень — у такому разі доцільно перейти на базу даних. Також у разі раптового вимкнення живлення повідомлення, які ще не були збережені й залишались у оперативній пам’яті, можуть зникнути, тому варто реалізувати механізм автоматичного збереження через певні інтервали часу.

Було проведено анкетування серед учнів 5-6 класів Калуської гімназії ім. Дмитра Бахматюка, які тестували новий локальний корпоративний месенджер. Участь взяли 29 учнів 5 класу та 24 учні 6 класу. Їм було запропоновано відповісти на кілька простих запитань про досвід використання месенджера.

Учні відзначають, що месенджер:

– "...дуже легкий у користуванні. Я швидко зрозумів, як писати повідомлення і створювати групи."

– "...допомагає швидко спілкуватися з однокласниками по уроках. Наприклад, коли забув домашнє завдання."

– "...не відволікає на рекламу чи відео, як інші месенджери. Можна зосередитись тільки на навчанні."

– "...зручний для групових проєктів. Ми створювали чати для наших груп і обговорювали завдання."

– "...прикольно, що це наш шкільний месенджер. Здається, що він безпечніший."

Водночас, учні хотіли б покращити...

– "...дизайн. Він трохи нудний, хотілося б більше кольорів або смайликів."

– "...можливість надсилати картинки або голосові повідомлення. Це було б дуже зручно."

– "...щоб він працював швидше і не зависав іноді."

– "...щоб приходили сповіщення, коли хтось пише, а я не в месенджері."

– "...можливість прикріплювати файли, наприклад, фотографії з домашніми завданнями."

Ці перші враження дають цінний погляд на те, що учні цінують у месенджері, і де є простір для покращення.

Таблиця 3.2 - Аналіз відгуків учнів про розроблений месенджер

Позитивні сторони (на думку учнів)	**Негативні сторони (на думку учнів)**
Простота та зручність: легкий в освоєнні, інтуїтивний інтерфейс.	Обмежений функціонал: бракує обміну фото, відео, голосових повідомлень, стікерів.
Швидкість комунікації: миттєва відправка/отримання повідомлень.	Несучасний дизайн: виглядає застаріло, мало кольорів та анімації.
Сфокусованість: відсутність реклами та стороннього контенту.	Проблеми зі стабільністю: іноді зависає або працює повільно.
Зручність групових чатів: добре підходить для командної роботи над проєктами.	Відсутність сповіщень: немає push-сповіщень про нові повідомлення.
Локальність та безпека: додає відчуття конфіденційності у шкільному середовищі.	Неможливість прикріплення файлів: відсутність функціоналу для обміну фотографіями, документами тощо.

Нижче наведено узагальнені результати у вигляді відсотків учнів, які відповіли «Так».

Таблиця 3.3 - Результати анкетування учнів

Питання	5 клас (Так, %)	6 клас (Так, %)
Чи було вам зручно користуватись месенджером?	83%	88%
Чи зрозумілий був інтерфейс?	90%	92%
Чи почували себе у безпеці під час спілкування?	79%	84%

Для перевірки функціональності, стабільності та безпеки системи корпоративного месенджера було розроблено базові тест-кейси. Вони охоплюють основні функціональні можливості клієнтської та серверної частин, а також перевірку обробки виняткових ситуацій.

Таблиця 3.4 –Тестові сценарії перевірки системи

№	Назва тесту	Вхідні дані	Очікуваний результат
1	Авторизація з вірними даними	Логін: testuser, Пароль: 123456	Успішний вхід, відкривається вікно чату
2	Авторизація з хибними даними	Логін: testuser, Пароль: wrongpass	Повідомлення про помилку, доступ заборонено
3	Реєстрація нового користувача	Нові логін/пароль	Повідомлення про успішну реєстрацію, вхід у чат
4	Надсилання повідомлення	Текстове повідомлення	Повідомлення з'являється у вікні чату адресата
5	Надсилання порожнього повідомлення	Порожній текст	Відображення помилки, повідомлення не надсилається
6	Вихід з облікового запису	Кнопка "Вийти"	Повернення до вікна авторизації
7	Відключення сервера під час чату	Сервер недоступний	Вивід помилки, припинення зв'язку
8	Спроба повторного входу без реєстрації	Логін неіснуючого користувача	Повідомлення про помилку "Користувача не знайдено"
9	Тестування шифрування повідомлень	Повідомлення між двома клієнтами	Повідомлення передається у зашифрованому вигляді
10	Перевірка одночасного підключення	20 клієнтів під'єднані одночасно	Система стабільно обробляє всі підключення

Ця таблиця деталізує різноманітні функціональні та нефункціональні тести, що охоплюють авторизацію, реєстрацію, обмін повідомленнями, обробку помилок та одночасне підключення, з метою забезпечення стабільної та коректної роботи системи.

ВИСНОВКИ

Дана робота присвячена розробці та дослідженню корпоративного месенджера, призначеного для забезпечення безпечної комунікації між учнями та викладачами Калуського ліцею ім. Д. Бахматюка в умовах локальної мережі.

У першому розділі було проведено детальний аналіз середовища впровадження, а саме — Калуського ліцею, як освітньої установи з потребою в ефективному та безпечному інструменті для цифрового спілкування. Також у цьому розділі розкрито основні поняття, пов’язані з безпечним обміном повідомленнями, проаналізовано ключові загрози, притаманні локальним мережам, та методи їхньої нейтралізації. Особливу увагу приділено технічним та соціальним аспектам захисту даних у шкільному середовищі. Крім того, здійснено порівняльний аналіз популярних месенджерів, що дозволило виявити їхні слабкі сторони та визначити необхідні функціональні й безпекові вимоги до нової системи.

Другий розділ був присвячений проєктуванню месенджера. У ньому детально описано архітектуру майбутньої системи, її основні компоненти та логіку взаємодії між ними. Здійснено вибір відповідних технологій для клієнтської частини, серверної логіки та зберігання даних. Особливу увагу приділено протоколу обміну повідомленнями, який базується на концепції наскрізного шифрування. Як базовий криптографічний протокол розглянуто Signal Protocol, що є сучасним і надійним засобом забезпечення конфіденційності цифрових комунікацій. Також у цьому розділі описано вимоги до продуктивності, стабільності та масштабованості системи.

У третьому розділі наведено реалізацію прототипу корпоративного месенджера з урахуванням усіх розроблених вимог. Серверна частина побудована за принципом багатопотокової обробки запитів клієнтів, що дозволяє підтримувати кількох користувачів одночасно. Клієнтський додаток має зручний графічний інтерфейс, адаптований для учнівської аудиторії. У межах тестування перевірено ключові функції системи: реєстрацію,

авторизацію, обмін повідомленнями, обробку помилок, шифрування та стійкість до порушень з'єднання. Отримані результати свідчать про правильну роботу месенджера в умовах локальної мережі. Крім технічного тестування, була проведена апробація системи серед учнів ліцею, що дозволило виявити зручність користування і рівень задоволення функціональністю. Зворотній зв’язок дав змогу сформулювати реальні напрями вдосконалення та оптимізації системи для подальшого впровадження.

У результаті дослідження було створено функціональний прототип корпоративного месенджера, що відповідає більшості заданих вимог. Програма забезпечує базовий рівень захищеності даних та ефективний обмін повідомленнями в локальному середовищі. Реалізовані алгоритми шифрування, реєстрації та обробки повідомлень працюють стабільно, а розроблена інфраструктура легко адаптується до потреб невеликих організацій. Отримані результати свідчать про високий потенціал розробленої системи для подальшого вдосконалення. Зокрема, перспективними напрямами розвитку залишаються: повна інтеграція протоколу Signal, розширення можливостей клієнтського застосунку, створення мобільної версії, а також масштабування інфраструктури для роботи у більших мережах або через Інтернет.

Таким чином, мета роботи була досягнута: теоретично обґрунтовано й практично реалізовано захищений корпоративний месенджер, адаптований під потреби освітнього закладу, з урахуванням сучасних вимог до інформаційної безпеки. Розроблений застосунок може стати ефективною складовою цифрового середовища навчального закладу, сприяючи покращенню внутрішньої комунікації, підвищенню дисципліни та цифрової грамотності учнів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Інформація про заклад. URL: https://bakhmatiuklyceum.if.ua/ (дата звернення: 02.05.2025).

2. Закон України «Про освіту». URL: https://zakon.rada.gov.ua/laws/show/2145-19#Text (дата звернення: 02.05.2025).

3. Закон України «Про захист персональних даних». URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 02.05.2025).

4. Закон України «Про інформацію». URL: https://zakon.rada.gov.ua/laws/show/2657-12#Text (дата звернення: 02.05.2025).

5. Стратегія кібербезпеки України. URL: https://zakon.rada.gov.ua/laws/show/n0087525-21#Text (дата звернення: 02.05.2025).

6. Накази Міністерства освіти і науки України. URL: https://mon.gov.ua/timeline? (дата звернення: 02.05.2025).

7. Переваги E2EE. URL: https://itedu.center/ua/blog/review/naskrizne-shyfruvannia-prostymy-slovamy-pro-e2ee/ (дата звернення: 02.05.2025).

8. Криптографічні хеш-функції. URL: https://ami.lnu.edu.ua/wp-content/uploads/2022/06/Cryptology9.pdf. (дата звернення: 02.05.2025).

9. Шифрування даних в інтернеті. URL: https://yak.dslua.org/articles/shcho-potribno-znaty-pro-shyfruvannia-danykh-v-interneti/ (дата звернення: 02.05.2025).

10. Атака «людина посередині». URL: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/ (дата звернення: 02.05.2025).

11. Різниця між DoS- та DDoS-атаки. URL: https://elit-web.ua/ua/blog/chto-takoe-ddos-ataki (дата звернення: 02.05.2025).

12. Локальна комп’ютерна мережа. URL: https://www.kievoit.ippo.kubg.edu.ua/kievoit/2013/21/21.html (дата звернення: 02.05.2025).

13. VLAN. URL: https://maxnet.ua/blog/sho-take-vlan-i-yak-vona-dopomagaye-organizuvati-merezhu-v-biznesi/ (дата звернення: 05.05.2025).

14. IDS проти IPS. URL: https://www.bitlyft.com/resources/ids-vs-ips-vs-siem (дата звернення: 05.05.2025).

15. Принцип роботи EDR. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-edr-endpoint-detection-response (дата звернення: 05.05.2025).

16. Багатофакторна автентифікація. URL: https://aws.amazon.com/iam/features/mfa/ (дата звернення: 05.05.2025).

17. Безпека додатків для обміну повідомленнями. URL: https://www.kaspersky.com/resource-center/preemptive-safety/messaging-app-security (дата звернення: 09.05.2025).

18. Популярні месенджери. URL: https://contentguide.com.ua/najkrashhi-mesendzhery-u-2024-roczi-yak-vybraty-dlya-sebe/ (дата звернення: 09.05.2025).

19. Наскрізне шифрування. URL: https://cso.cyberhandbook.org/uk/topics/storing-data/komunikatsiya-y-obmin-danymy (дата звернення: 09.05.2025).

20. E2EE. URL: https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE (дата звернення: 09.05.2025).

21. Розуміння клієнт-серверної архітектури. URL: https://dou.ua/forums/topic/44636/ (дата звернення: 13.05.2025).

22. Якість програмного забезпечення. URL: https://qalight.ua/baza-znaniy/yakist-programnogo-zabezpechennya/ (дата звернення: 13.05.2025).

23. Функціональні та нефункціональні вимоги. URL: https://www.guru99.com/uk/functional-vs-non-functional-requirements.html (дата звернення: 13.05.2025).

24. Життєвий цикл вимог. URL: https://1qip1.com/qa-engineer-library/3221-zhyttievyi-cykl-produktu.html (дата звернення: 13.05.2025).

25. Signal Protocol. URL: https://github.com/signalapp/libsignal (дата звернення: 13.05.2025).

26. Потік обміну повідомленнями. URL: https://ela.kpi.ua/server/api/core/bitstreams/4f3a099d-f38f-4545-90e8-2a4a9a4adc52/content (дата звернення: 13.05.2025).

ДОДАТКИ

Додаток А

Server

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Security.Cryptography;
using System.Text;
using System.Threading;
using System.Linq;

class Server
{
    // ► мережа
    static List<TcpClient> clients = new();             // активні TCP-
з'єднання
    static readonly object locker = new();

    // ► файли
    static readonly string userFile = "users.txt";  // login:hash
    static   readonly   string   chatsFile   =   "chats.txt";       //
chat:login1,login2,…
    static readonly string historyFolder = "chat_history";

    // ► користувачі й чати у пам'яті
    static Dictionary<string, string> userHashes = new();             //
login -> hash
    static Dictionary<string, List<string>> chats = new();            //
chat -> учасники
    static Dictionary<string, string> userChat = new();               //
login -> active chat
    static Dictionary<string, List<string>> chatHistory = new();     //
chat -> список повідомлень
    static   Dictionary<string,   TcpClient>   loginClient   =   new();
// login -> TcpClient

    static void Main()
    {
        if (!Directory.Exists(historyFolder))
            Directory.CreateDirectory(historyFolder);

        LoadUsers();
        LoadChats();
        LoadHistories();

        TcpListener server = new TcpListener(IPAddress.Any, 5000);
        server.Start();
        Console.WriteLine("Сервер запущено на порті 5000…");

        while (true)
        {
            TcpClient client = server.AcceptTcpClient();
            new Thread(() => HandleClient(client)).Start();
        }
    }
```

```
    /*————————————————————————————  ПРИЙОМ  КЛІЄНТА
————————————————————————————————*/

    static void HandleClient(TcpClient client)
    {
        using NetworkStream stream = client.GetStream();
        using StreamReader reader = new(stream, Encoding.UTF8);
        using StreamWriter writer = new(stream, Encoding.UTF8) {
AutoFlush = true };

        string login = null;

        try
        {
            string authLine = reader.ReadLine();                    //
LOGIN:login:password або REGISTER:login:password
            if (!AuthProcess(authLine, writer, out login))
                return;                                      // auth failed
→ з'єднання закривається

            lock (locker)
            {
                clients.Add(client);
                loginClient[login] = client;
            }

            writer.WriteLine("INFO:Вітаємо! Введіть /help для списку
команд.");

            while (true)
            {
                string line = reader.ReadLine();
                if (line == null) break;                        // клієнт
розірвав з’єднання

                if (line.StartsWith("/"))
                    HandleCommand(login, line.Trim(), writer);
                else
                    SendToActiveChat(login, line);
            }
        }
        catch { /* ігноруємо — клієнт відключився */ }
        finally
        {
            lock (locker)
            {
                clients.Remove(client);
                loginClient.Remove(login);
                userChat.Remove(login);
            }
            client.Close();
            Console.WriteLine($"{login ?? "??"} відключився");
        }
    }
    /*————————————————————————————  АВТОРИЗАЦІЯ / РЕЄСТРАЦІЯ
————————————————————*/
```

```
    static bool AuthProcess(string line, StreamWriter writer, out string
login)
    {
        login = null;
        var p = line?.Split(':');
        if (p == null || p.Length != 3)
        {
            writer.WriteLine("ERROR:Невірний формат авторизації");
            return false;
        }

        string command = p[0];
        login = p[1];
        string pass = p[2];
        string hash = Hash(pass);

        lock (locker)


        {
            if (command == "REGISTER")
            {
                if (userHashes.ContainsKey(login))
                {
                    writer.WriteLine("ERROR:Користувач вже існує");
                    return false;
                }
                userHashes[login] = hash;
                File.AppendAllText(userFile, $"{login}:{hash}\n");
                writer.WriteLine("OK:Реєстрація успішна");
            }
            else if (command == "LOGIN")
            {
                if     (!userHashes.TryGetValue(login,     out     string
storedHash) || storedHash != hash)
                {
                    writer.WriteLine("ERROR:Невірний логін або пароль");
                    return false;
                }
                writer.WriteLine("OK:Вхід успішний");
            }
            else
            {
                writer.WriteLine("ERROR:Невідома команда");
                return false;
            }
        }
        Console.WriteLine($"{login} увійшов");
        return true;
    }

    /*———————————————————————————                                    КОМАНДИ
———————————————————————————————————*/
    static  void  HandleCommand(string  login,  string  cmd,  StreamWriter
writer)
    {
```

```
        string[]        p        =        cmd.Split('        ',        2,
StringSplitOptions.RemoveEmptyEntries);
        string name = p.Length > 1 ? p[1].Trim() : null;

        switch (p[0])
        {
            case "/create_chat":
                if          (string.IsNullOrWhiteSpace(name))          {
writer.WriteLine("ERROR:Назва чату?"); return; }
                lock (locker)
                {
                    if            (chats.ContainsKey(name))            {
writer.WriteLine("ERROR:Такий чат існує"); return; }
                    chats[name] = new List<string> { login };
                    chatHistory[name] = new List<string>();
                    userChat[login] = name;
                    SaveChats();
                }
                writer.WriteLine($"INFO:Чат '{name}' створено");
                break;
            case "/add_user":
                // синтаксис: /add_user chatName anotherLogin
                if  (string.IsNullOrWhiteSpace(name)  ||  !name.Contains('
'))
                {    writer.WriteLine("ERROR:Формат    /add_user    <chat>
<login>"); return; }

                var      tokens      =      name.Split('      ',      2,
StringSplitOptions.RemoveEmptyEntries);
                string chatName = tokens[0];
                string newUser = tokens[1];

                lock (locker)
                {
                    if (!chats.ContainsKey(chatName))
                    { writer.WriteLine("ERROR:Чат не знайдено"); return;
}

                    // лише автор створює – за бажанням можна змінити
правило
                    if (!chats[chatName].Contains(login))
                    {   writer.WriteLine("ERROR:Ви   не   учасник   цього
чату"); return; }

                    if (!userHashes.ContainsKey(newUser))
                    {     writer.WriteLine("ERROR:Такого     користувача
немає"); return; }

                    if (!chats[chatName].Contains(newUser))
                        chats[chatName].Add(newUser);

                    SaveChats();
                }
                writer.WriteLine($"INFO:{newUser}        додано        до
{chatName}");
                break;
```

```
            case "/join":
                if          (string.IsNullOrWhiteSpace(name))          {
writer.WriteLine("ERROR:Назва чату?"); return; }
                lock (locker)
                {
                    if   (!chats.TryGetValue(name,   out   var   list)   ||
!list.Contains(login))
                    {
                        writer.WriteLine("ERROR:Немає доступу до чату");
                        return;
                    }
                    userChat[login] = name;
                }
                writer.WriteLine($"INFO:Приєднано до '{name}'");
                break;


            case "/list_chats":
                IEnumerable<string> own;
                lock     (locker)     own     =     chats.Where(c     =>
c.Value.Contains(login)).Select(c => c.Key).ToList();
                writer.WriteLine("CHATS:" + string.Join(',', own));
                break;

            case "/msg":        // /msg chatName решта_тексту
                if (name == null || !name.Contains(' '))
                {     writer.WriteLine("ERROR:Формат     /msg     <chat>
<повідомлення>"); return; }

                // перше слово – назва чату, далі – текст
                int space = name.IndexOf(' ');
                string chat = name[..space];
                string text = name[(space + 1)..];

                lock (locker)
                {
                    if            (!chats.ContainsKey(chat)            ||
!chats[chat].Contains(login))
                    {  writer.WriteLine("ERROR:Немає  доступу  до  чату");
return; }

                    // Зберігаємо в історію
                    if                  (!chatHistory.ContainsKey(chat))
chatHistory[chat] = new List<string>();
                    string full = $"{login}: {text}";
                    chatHistory[chat].Add(full);
                    File.AppendAllText(Path.Combine(historyFolder,
$"{chat}.txt"), full + Environment.NewLine);

                    // Розсилаємо всім учасникам
                    foreach (var user in chats[chat])
                        if  (loginClient.TryGetValue(user,  out  TcpClient
c))
                            SendRaw(c, $"CHAT_MSG:{chat}:{full}");
                }
                break;
```

```
            case "/get_messages":   // /get_messages chatName
                if (string.IsNullOrWhiteSpace(name))
                { writer.WriteLine("ERROR:Формат /get_messages <chat>");
return; }

                chat = name;
                lock (locker)
                {
                    if            (!chats.ContainsKey(chat)            ||
!chats[chat].Contains(login))
                    { writer.WriteLine("ERROR:Немає доступу до чату");
return; }

                    StringBuilder sb = new($"MESSAGES:{chat}\n");
                    if (chatHistory.TryGetValue(chat, out var list))
                        foreach (var m in list) sb.AppendLine(m);

                    writer.WriteLine(sb.ToString().TrimEnd());
                    LoadHistories();
                }
                break;

            default:
                writer.WriteLine("ERROR:Невідома команда");
                break;
        }
    }

    /*———————————————————————————          БРОДКАСТ       У       ЧАТ
———————————————————————————*/

    static void SendToActiveChat(string sender, string text)
    {
        string chat;
        lock (locker)
            if  (!userChat.TryGetValue(sender,  out  chat))  return;  //
немає активного чату

        byte[] data = Encoding.UTF8.GetBytes($"{sender}: {text}\n");

        lock (locker)
        {
            foreach (string user in chats[chat])
                if (loginClient.TryGetValue(user, out TcpClient c))
                    try { c.GetStream().Write(data, 0, data.Length); }
catch { }
        }
    }

    /*———————————————————————————                ХЕЛПЕРИ         ФАЙЛІВ
———————————————————————————*/

    static void LoadUsers()
    {
        if (!File.Exists(userFile)) return;
        foreach (var line in File.ReadAllLines(userFile))
```

```
        {
            var p = line.Split(':');
            if (p.Length == 2) userHashes[p[0]] = p[1];
        }
    }

    static void LoadChats()
    {
        if (!File.Exists(chatsFile)) return;
        foreach (var line in File.ReadAllLines(chatsFile))
        {
            var p = line.Split(':', 2);
            if (p.Length == 2)
                chats[p[0]]               =               p[1].Split(',',
StringSplitOptions.RemoveEmptyEntries).ToList();
        }
    }

    static void SaveChats()
    {
        var  lines  =  chats.Select(kv  =>  $"{kv.Key}:{string.Join(',',
kv.Value)}");
        File.WriteAllLines(chatsFile, lines);
    }

    static void LoadHistories()
    {
        if (!Directory.Exists(historyFolder)) return;
        foreach (var file in Directory.GetFiles(historyFolder, "*.txt"))
        {
            string chat = Path.GetFileNameWithoutExtension(file);
            chatHistory[chat] = File.ReadAllLines(file).ToList();
        }
    }
    static string Hash(string pass)
    {
        using SHA256 sha = SHA256.Create();
        return
Convert.ToBase64String(sha.ComputeHash(Encoding.UTF8.GetBytes(pass)));
    }
    static void SendRaw(TcpClient client, string line)
    {
        try
        {
            var sw = new StreamWriter(client.GetStream(), Encoding.UTF8)
{ AutoFlush = true };
            sw.WriteLine(line);
        }
        catch { /* клієнт міг відключитись */ }
    }

}
```

Program

```
namespace ChatClient
{
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.SetHighDpiMode(HighDpiMode.SystemAware);
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            var loginForm = new LoginForm();
            if (loginForm.ShowDialog() == DialogResult.OK)
            {
                string login = loginForm.Login;
                string password = loginForm.Password;
                bool isRegister = loginForm.IsRegister;

                Application.Run(new ChatForm(login, password,
isRegister));
            }
        }
    }
}
```

LoginForm

```
using Microsoft.VisualBasic.Logging;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ChatClient
{
    public partial class LoginForm : Form
    {
        public string Login { get; private set; }
        public string Password { get; private set; }
        public bool IsRegister { get; private set; } = false;
        public LoginForm()
        {
            InitializeComponent();
        }

        private void buttonLogin_Click(object sender, EventArgs e)
        {
```

```
            if (!ValidateInputs()) return;

            Login = textBoxLogin.Text.Trim();

            Password = textBoxPassword.Text.Trim();
            IsRegister = false;

            DialogResult = DialogResult.OK;
            Close();
        }

        private void buttonRegister_Click(object sender, EventArgs e)
        {
            if (!ValidateInputs()) return;

            Login = textBoxLogin.Text.Trim();
            Password = textBoxPassword.Text.Trim();
            IsRegister = true;

            DialogResult = DialogResult.OK;
            Close();
        }

        private bool ValidateInputs()
        {
            if (string.IsNullOrWhiteSpace(textBoxLogin.Text) ||
                string.IsNullOrWhiteSpace(textBoxPassword.Text))
            {
                MessageBox.Show("Будь ласка, введіть логін і пароль!");
                return false;
            }
            return true;
        }
    }
}
```

ChatForm

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Windows.Forms;

namespace ChatClient
{
    public partial class ChatForm : Form
    {
        TcpClient client;
        StreamReader reader;
        StreamWriter writer;
        NetworkStream stream;
        Thread receiveThread;
        string login, password;
```

```
        bool isRegister;
        private string currentChat = null;

        public ChatForm(string login, string password, bool isRegister)
        {
            InitializeComponent();
            this.login = login;

            this.password = password;
            this.isRegister = isRegister;

            ConnectToServer();
            listChats.SelectedIndexChanged +=
listChats_SelectedIndexChanged;

        }

        void ConnectToServer()
        {
            try
            {
                client = new TcpClient("127.0.0.1", 5000);
                stream = client.GetStream();

                // Ініціалізація reader та writer
                reader = new StreamReader(stream, Encoding.UTF8);
                writer = new StreamWriter(stream, Encoding.UTF8) {
AutoFlush = true };

                // Надсилаємо тип запиту: login або register
                string command = isRegister ? "REGISTER" : "LOGIN";
                string auth = $"{command}:{login}:{password}\n";
                byte[] authData = Encoding.UTF8.GetBytes(auth);
                stream.Write(authData, 0, authData.Length);

                receiveThread = new Thread(ReceiveMessages) {
IsBackground = true };
                receiveThread.Start();
                writer.WriteLine("/list_chats");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка підключення: " + ex.Message);
                Close();
            }
        }

        void ReceiveMessages()
        {
            while (true)
            {
                try
                {
                    string line = reader.ReadLine();
                    if (line == null) break;          // сервер закрився
                    /* ---------- CHATS ---------- */
```

```
                    if (line.StartsWith("CHATS:"))
                    {
                        string[] chats = line[6..].Split(',',
StringSplitOptions.RemoveEmptyEntries);
                        Invoke(() =>
                        {
                            listChats.Items.Clear();
                            listChats.Items.AddRange(chats);
                        });
                        continue;
                    }

                    /* ---------- CHAT_MSG ---------- */
                    if (line.StartsWith("CHAT_MSG:"))
                    {
                        // CHAT_MSG:chat:текст
                        var p = line.Split(':', 3);
                        string chat = p[1];
                        string body = p[2];

                        if (chat == currentChat)
                            Invoke(() => richTextBox1.AppendText(body +
"\n"));
                        continue;
                    }

                    /* ---------- MESSAGES (багаторядкові) ---------- */
                    if (line.StartsWith("MESSAGES:"))
                    {
                        string chat = line.Split(':')[1];
                        List<string> msgs = new();

                        // читаємо до порожнього рядка
                        while (true)
                        {
                            string m = reader.ReadLine();
                            if (string.IsNullOrWhiteSpace(m)) break;
                            msgs.Add(m);
                        }

                        if (chat == currentChat)
                        {
                            Invoke(() =>
                            {
                                richTextBox1.Clear();
                                foreach (var m in msgs)
richTextBox1.AppendText(m + "\n");
                            });
                        }
                        continue;
                    }

                    /* ---------- INFO / ERROR ---------- */
                    if (line.StartsWith("OK:") ||
line.StartsWith("INFO:") || line.StartsWith("ERROR:"))
                    {
```

```
                        Invoke((MethodInvoker)(() =>
                        {
                            MessageBox.Show(line);
                        }));
                        continue;
                    }
                }
                catch { break; }
            }
        }



        private void buttonSend_Click_1(object sender, EventArgs e)
        {
            if (string.IsNullOrWhiteSpace(textBox1.Text)) return;

            if (listChats.SelectedItem == null)
            {
                MessageBox.Show("Оберіть чат для надсилання
повідомлення.");
                return;
            }

            string selectedChat = listChats.SelectedItem.ToString();
            string msg = textBox1.Text;
            string full = $"/msg {selectedChat} [{login}]: {msg}";

            try
            {
                writer.WriteLine(full);
                richTextBox1.AppendText("Я: " + msg + "\n");
                textBox1.Clear();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка при відправці повідомлення: " +
ex.Message);
            }
        }

        private void btnCreateChat_Click(object sender, EventArgs e)
        {
            // Просимо користувача ввести назву нового чату
            string chatName =
Microsoft.VisualBasic.Interaction.InputBox(
                "Введіть назву нового чату:",
                "Створити чат");

            // Якщо рядок порожній або відміна — нічого не робимо
            if (string.IsNullOrWhiteSpace(chatName)) return;

            // Надсилаємо команду серверу
            writer.WriteLine($"/create_chat {chatName}");

            // Одразу запитуємо актуальний список чатів
```

```
            writer.WriteLine("/list_chats");
        }

        private void btnAddUser_Click(object sender, EventArgs e)
        {
            if (listChats.SelectedItem == null)
            {
                MessageBox.Show("Оберіть чат.");
                return;
            }

            string selectedChat = listChats.SelectedItem.ToString();

            string username =
Microsoft.VisualBasic.Interaction.InputBox("Введіть ім'я користувача для
додавання:", "Додати учасника");
            if (!string.IsNullOrWhiteSpace(username))
            {
                writer.WriteLine($"/add_user {selectedChat}
{username}");
            }
        }
        private void listChats_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (listChats.SelectedItem != null)
            {
                currentChat = listChats.SelectedItem.ToString();
                richTextBox1.Clear();

                writer.WriteLine($"/get_messages {currentChat}"); //
запит історії повідомлень чату
            }
        }

        protected override void OnFormClosing(FormClosingEventArgs e)
        {
            try
            {
                writer?.WriteLine("/exit");
                receiveThread?.Abort();
                stream?.Close();
                client?.Close();
            }
            catch { }

            base.OnFormClosing(e);
        }
    }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: **Розробка локального корпоративного месенджера для комунікації учнів Калуського ліцею ім. Д. Бахматюка**

Обсяг пояснювальної записки 72 аркушів:

15 таблиць;

13 рисунків;

2 додатка.

Дата завершення роботи: *09 червня 2025р*.

Підпис студента - _____*Олійник Р.П..*