Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис

«___»_____2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної роботи
освітньо-професійного ступеня «Фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»

на тему: «Шифрований кросплатформний месенджер з відкритим кодом»

Студент групи КН-41	Мар’ян ПЕЛЕХ	_____ (підпис)
Керівник роботи	Степан ІВАСЬЄВ	_____ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Наталя КУЛЬЧИНСЬКА	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«__»_____2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студенту Пелеху Мар’яну Павловичу

(прізвище, ім’я та по-батькові студента)

1. Тема роботи_____Шифрований кросплатформний месенджер з відкритим кодом_____

   затверджена наказом по коледжу від «25» листопада 2024 р., № 253а-н

2. Термін здачі студентом завершеної роботи «___»_____2025 р.
3. Вихідні дані до роботи_____Результати аналізу наявних рішень у сфері текстової цифрової комунікації, результати порівняння алгоритмів_____ шифрування, аналіз технологій розробки кросплатформних застосунків
4. Перелік питань, які повинні бути розроблені:

   а) основна частина Аналіз предметної області та постановка_____ завдань. Проєктування системи. Реалізація та тестування системи.____

   б) техніко-економічне обгрунтування___Аналіз ринку збуту продукту. Розрахункова частина. Обґрунтування доцільності створення системи.__

5. Перелік графічного матеріалу___Схема архітектурної взаємодії, Діаграма класів серверної частини, Діаграма послідовності аутентифікації користувача, Схема надсилання повідомлень у системі, Діаграма варіантів використання _____
6. Консультанти роботи:_____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
3 техніко-економічного обґрунтування	Меленчук Л. І. (вчена ступінь, звання П.І.Б _____ консультант)		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми, аналіз вимог до кваліфікаційної роботи	20.11.2024	24.11.2024
2.	Огляд конкурентних рішень, написання аналітичного розділу	25.11.2024	04.12.2024
3.	Розробка функціональних та нефункціональних вимог до месенджера	05.12.2024	07.12.2024
4.	Дослідження технологій реалізації та протоколів шифрування	08.12.2024	12.12.2024
5.	Встановлення та налаштування середовища розробки	13.12.2024	14.12.2024
6.	Проєктування системи, написання розділу	15.12.2024	26.12.2024
7.	Реалізація системи месенджера	27.12.2024	02.05.2025
8.	Тестування месенджера, написання розділу реалізації	03.05.2025	09.05.2025
9.	Написання техніко-економічного обґрунтування	10.05.2025	11.05.2025
10.	Оформлення пояснювальної записки	12.05.2025	13.06.2025
11.	Попередній захист кваліфікаційної роботи	13.06.2025	13.06.2025
12.	Підготовка до захисту кваліфікаційної роботи	14.06.2025	24.06.2025
13.	Захист кваліфікаційної роботи	25.06.2025	25.06.2025

7. Дата видачі завдання «__»_____ 2024 р. Керівник _____/

Завдання прийняв до виконання _____/ /

Реферат

Кваліфікаційна робота. Шифрований кросплатформний месенджер з відкритим кодом. Мар’ян Пелех. Галицький фаховий коледж імені В’ячеслава Чорновола. Відділення комп’ютерних технологій. 82с., 34 рисунків, 5 додатків.

Об’єкт дослідження – застосування алгоритмів шифрування у засобах цифрової текстової комунікації.

Метою роботи є реалізація кросплатформного месенджера на базі клієнт-серверної архітектури з підтримкою наскрізного шифрування через поєднання симетричного та асиметричного алгоритмів, використовуючи такі засоби, як мова програмування C++ для реалізації основного функціоналу, фреймворк Qt та декларативна мова QML для графічного інтерфейсу.

Месенджер повинен забезпечувати високий рівень конфіденційності даних користувачів, надавати зручний графічний інтерфейс та забезпечувати стабільну роботу на різних платформах (Windows, Linux). Крім того, необхідно спроєктувати та реалізувати систему обміну повідомленнями з використанням гібридного шифрування для максимального захисту користувацьких даних.

Результатом розробки став готовий до використання повноцінний кросплатформний месенджер із захистом даних через наскрізне гібридне шифрування, зручним користувацьким графічним інтерфейсом та відкритим вихідним кодом.

Ключові слова: ENCRYPTED MESSENGER, CROSS-PLATFORM, OPEN SOURCE, END-TO-END ENCRYPTION, MariaDB, RestfulAPI, QT, QML, C++, Windows, Linux.

Abstract

Qualification work. Encrypted cross-platform open source messenger. Marian Pelekh. Halytsky Applied College named after Vyacheslav Chornovil. Department of Computer Technology. 82 p., 34 figures, 5 appendices.

The object of research is the use of encryption algorithms in digital text communication.

The aim of the work is to implement a cross-platform messenger based on a client-server architecture with support of end-to-end encryption through a combination of symmetric and asymmetric algorithms, using such tools as the C++ programming language for implementing the main functionality, the Qt framework and the QML declarative language for the graphical interface

The messenger should ensure a high level of confidentiality of user data, provide a user-friendly graphical interface, and ensure stable operation on different platforms (Windows, Linux). In addition, it is necessary to design and implement a communication system using hybrid encryption to ensure maximum protection of user data.

The result of the development is a ready-to-use cross-platform messenger with data protection through end-to-end hybrid encryption, a user-friendly graphical user interface, and open source code.

Keywords: ENCRYPTED MESSENGER, CROSS-PLATFORM, OPEN SOURCE, END-TO-END ENCRYPTION, MariaDB, RestfulAPI, QT, QML, C++, Windows, Linux.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних.

ЄСВ – Єдиний Соціальний Внесок.

ІТ – інформаційні технології.

ПК – персональний комп’ютер.

СКБД – Система керування базами даних.

2FA – Two-Factor Authentication (двофакторна автентифікація).

E2EE (E2E) – End-to-end Encryption (наскрізне шифрування).

ER – Entity-Relationship (Сутність-Зв’язок).

GCM – Galois/Counter Mode.

GDPR – General Data Protection Regulation (Загальний регламент захисту даних).

GUI – Graphical User Interface (графічний користувацький інтерфейс).

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту).

IDE – Integrated Development Environment (інтегроване середовище розробки).

MAU – Montly active users (активні користувачі за місяць).

P2P – Peer-to-peer (однорангова архітектура).

QA – Quality Assistance (контроль якості).

QML – Qt Meta Language/Qt Modeling Language.

RAII – Resource Acquisition Is Initialization.

RSA – Rivest-Shamir-Adleman.

SMM – Social Media Marketing (Маркетинг у соціальних мережах).

UI/UX – User Interface/User Expirience (користувацький інтерфейс/користувацький досвід).

URL – Uniform Resource Locator (уніфікований локатор ресурсів).

UUID – Universally unique identifier (універсальний унікальний ідентифікатор).

ВСТУП

Месенджери стали вагомою частиною життя суспільства, забезпечуючи швидкий обмін повідомленнями, файлами та голосовою комунікацією. Вони використовуються не лише для особистого спілкування, але й для ведення бізнесу, навчання, організації спільної роботи тощо. Однак, всупереч їхній популярності, багато платформ мають суттєві недоліки, зокрема у впровадженні належного рівня захищеності даних, забезпеченні конфіденційності та зручності користування.

Популярні месенджери, такі як Telegram, Viber чи Discord, зберігають незашифровані метадані користувачів, використовують непрозорі алгоритми шифрування або не використовують шифрування взагалі. Це створює ризики для конфіденційності та безпеки даних, особливо в умовах сучасних кіберзагроз. Крім того, відсутність конкурентних українських рішень формує необхідність розробки власного месенджера, незалежного від інших держав.

У цьому контексті розробка українського шифрованого месенджера стає важливим кроком для впровадження безпеки, зручності та незалежності цифрової комунікації в Україні. Відкритий код дозволяє здійснювати незалежний аудит безпеки, користувачам пропонувати власні покращення месенджера та зберігати прозорість роботи системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Огляд предметної області

Месенджер – це програмний продукт, призначений для обміну інформацією в режимі реального часу. За останні десятиліття застосунки для обміну повідомленнями стрімко вдосконалювались, трансформуючись від простих текстових сервісів до складних платформ, що інтегрують голосові та відеодзвінки, обмін файлами, групову комунікацію та інструменти для спільної роботи.

Більшість сучасних месенджерів базується на клієнт-серверній архітектурі, де сервер виконує роль посередника: він зберігає дані, спрямовує повідомлення від відправника до отримувача та забезпечує синхронізацію між пристроями. Менш поширені децентралізовані системи пропонують альтернативу, де дані зберігаються на пристроях співрозмовників, що підвищує анонімність й безпеку, але при легко може спричинити втрату даних.

Користувачі є центральними учасниками системи, які здійснюють комунікацію, створюють групи та канали. Для реєстрації зазвичай потрібні мінімальні дані: електронна пошта або номер телефону, пароль і ім’я/псевдонім. Деякі платформи додають додаткові кроки, наприклад двофакторну автентифікацію. Після – або, інколи, під час – реєстрації користувачі налаштовують профілі, додаючи аватарку, відображуване ім’я, параметри приватності, наприклад, обмежуючи видимість своєї активності. Окрему категорію суб’єктів становлять боти – автоматизовані сервіси, які виконують задачі від імені користувачів, такі як надання інформації, організація опитувань або інтеграція зі зовнішніми системами.

Повідомлення є основним носієм інформації. Вони можуть бути текстовими, мультимедійними або файловими, супроводжуючись метаданими: часом відправлення, ідентифікатором відправника,

ідентифікатором чату, статусом отримання чи прочитання. Більшість месенджерів дозволяють редагувати або видаляти відправлені повідомлення.

Зазвичай месенджери мають декілька видів чатів, здебільшого це приватні чати – комунікація між двома користувачами; групи – чати для спілкування обмеженої кількості учасників; та канали – платформи для одностороннього розповсюдження інформації від адміністратора до підписників. Такий поділ забезпечує користувачам гнучкість у виборі формату комунікації. Кожен чат має унікальні атрибути: назву, опис, список учасників з їхніми ролями, правила доступу (наприклад, публічність або необхідність запрошення).

Автентифікація та авторизація є фундаментом безпеки месенджерів. Автентифікація передбачає перевірку особистості користувача через паролі, SMS-коди або біометрію, тоді як авторизація визначає рівні доступу до функцій. Обмін даними здебільшого відбувається через протокол WebSocket для забезпечення комунікації в режимі реального часу. Більшість сучасних месенджерів використовують наскрізне шифрування, яке гарантує, що повідомлення читають лише відправник і одержувач, навіть якщо повідомлення отримала стороння особа як посередник.

Важливим аспектом є політика зберігання: деякі платформи автоматично видаляють метадані через певний час, інші шифрують метадані або резервні копії, щоб уникнути витоку даних у разі зламу.

Різні месенджери орієнтуються на різні цільові аудиторії. Деякі платформи розроблені для повсякденного спілкування, акцентуючись на простоті та конфіденційності. Інші спеціалізуються на професійній комунікації, інтегруючи календарі, хмарні сховища або інструменти для управління проектами. Окрему категорію становлять геймерські месенджери, які оптимізовані для голосового зв’язку з мінімальною затримкою.

Багато месенджерів розроблені з можливістю стабільної та зручної роботи в різних середовищах, що забезпечує користувачам свободу вибору

пристроїв та операційних систем. Кросплатформність передбачає сумісність із iOS, Android, веб-браузерами, десктопними версіями (Windows, macOS, Linux) та спеціалізованими пристроями (наприклад, smart-годинниками). У випадку кросплатформної роботи важливою є миттєва синхронізація даних між пристроями.

Розробка даного месенджера має вирішити низку критичних проблем, які існують у популярних платформах. По-перше, це недостатній рівень конфіденційності, адже велика кількість месенджерів зберігає незашифровані метадані користувачів або використовує слабкі методи шифрування. По-друге, вирішується проблема відсутності українського продукту з функціоналом, що не поступається популярним аналогам у зручності та інноваційності, а також не ставить під загрозу цифрову комунікацію в Україні, адже підпорядковується законам України.

1.2 Аналіз наявних рішень

Сучасні месенджери є невіддільною частиною цифрової комунікації, пропонуючи широкий спектр можливостей для особистого та бізнес використання. Серед найпопулярніших сервісів – Telegram, Viber та Facebook Messenger, які приваблюють користувачів своєю зручністю, багатофункціональністю та підтримкою багатьох форматів спілкування.

Попри свою популярність, ці платформи не мають належного рівня безпеки та конфіденційності. Зокрема, деякі з них зазнають критики через вразливості в системах шифрування або політиці збору даних користувачів.

Для детального аналізу функціональних та нефункціональних можливостей було обрано месенджери Telegram, Viber та Discord. Це дозволить сформулювати чіткі вимоги до проєкту та врахувати як їхні переваги, так і недоліки.

Telegram – один з найпопулярніших месенджерів у сучасному інфопросторі, який станом на кінець 2024 року нараховував майже мільярд

активних користувачів. Він виступає не тільки як застосунок для обміну приватними повідомленнями, а і як повноцінна соціальна мережа [1].

Даний месенджер має широкий користувацький функціонал: можливість відправляти текстові повідомлення, медіаконтент, файли, голосові та відео повідомлення, здійснювати виклики. Платформа дозволяє утворювати приватні чати (двоє користувачів), групи (багато користувачів) та канали (групи, де можуть надсилати повідомлення лише адміністратори для широкого кола осіб).

Застосунок має комфортний дизайн, зрозумілі для користувачів іконки та достатній вибір тем для налаштування колірної палітри. Також Telegram пропонує можливість для написання власних тем та зміни фону чатів.

Візуально вікно програми можна поділити на три частини: панель з чатами у лівій частині, інформація про обраний чат у правій частині та саме вікно чату у центрі (рис. 1.1).

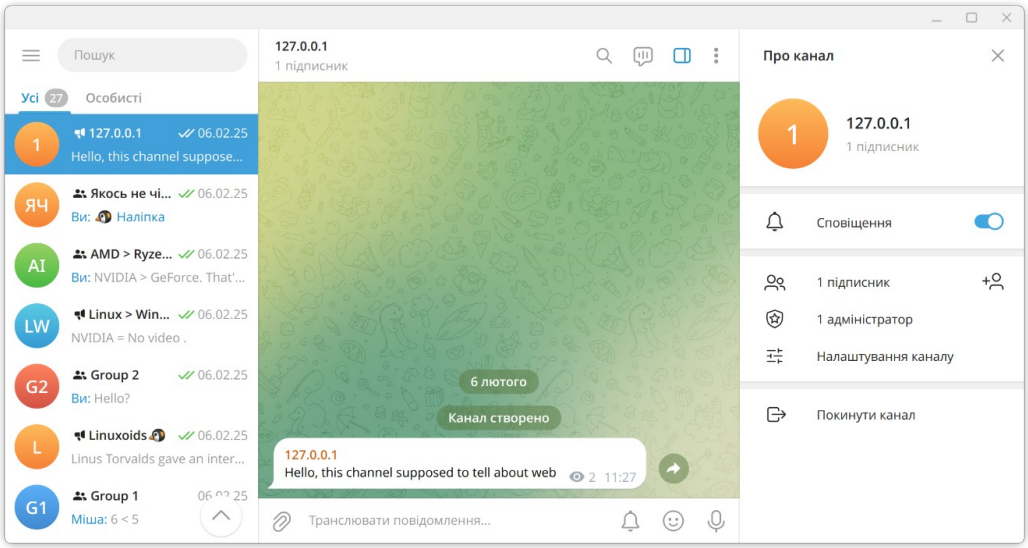

Рисунок 1.1 – Головне вікно месенджера Telegram

Чати у переліку включають зображення чату, назву та останнє повідомлення в чаті. Додатково відображається дата останнього повідомлення, а також марка прочитання й, у разі непрочитаних повідомлень, відповідний індикатор з їхньою кількістю.

Права частина вікна теж, умовно, поділяється на три частини: заголовок, область повідомлень та поле вводу. Заголовок включає назву чату, кількість підписників або мережевий статус співрозмовника, та кнопки: пошуку повідомлень, додавання опитувань, відображення правої панелі та кнопку «Більше».

Центральна частина включає історію повідомлень. У приватних чатах повідомлення включають лише текстовий або медіаконтент, марку прочитання і годину відсилання. У групових чатах повідомлення додатково включають ім’я та зображення відправника. При надсиланні повідомлення додаються унизу списку.

Нижня частина вікна призначена для написання повідомлень та включення додаткового контенту, наприклад, користувач може надсилати медіаконтент, здійснювати запис голосових та відеоповідомлень, а також надсилати наліпки та емоджі.

Навігація у застосунку є простою та інтуїтивною. Telegram можна назвати односторінковим застосунком: перелік чатів, інформація про чати, область повідомлень знаходяться поруч в одному вікні, а меню програми та налаштування відкриваються як модальні вікна (рис. 1.2).

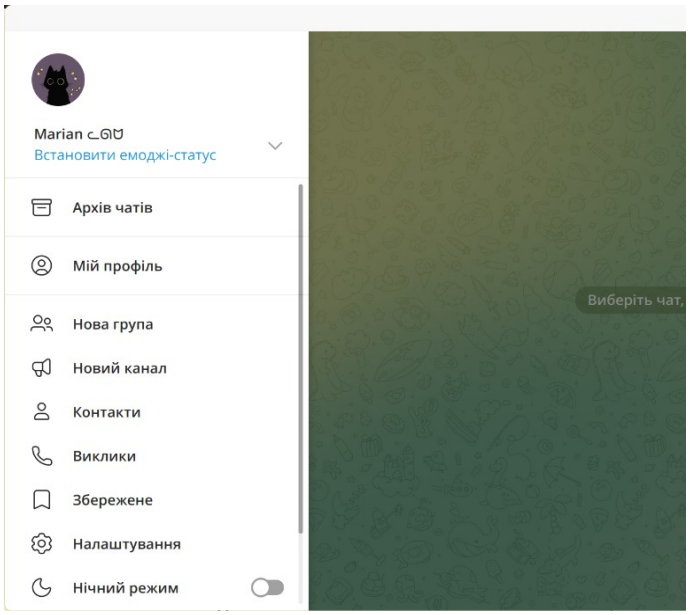

Рисунок 1.2 – Програмне меню Telegram

Однак, попри свої переваги, Telegram не завжди забезпечує належний рівень конфіденційності. Хоча платформа використовує власний протокол шифрування MTProto, звичайні чати не мають наскрізного шифрування, що означає можливість збереження історії повідомлень на серверах компанії. Лише так звані «секретні чати» використовують наскрізне шифрування, але вони мають обмежений функціонал та не підтримуються у веб-версії. Крім того, Telegram зберігає метадані користувачів, що може становити загрозу для їхньої приватності. Окрім того, відомо, що засновником Telegram є російський програміст та бізнесмен Павло Дуров, через що виникає небезпека передавання даних користувачів спеціальним службам країни-агресора. Це викликає певні сумніви щодо рівня безпеки месенджера, особливо у контексті сучасних вимог до конфіденційності цифрової комунікації.

Viber – популярний месенджер, розроблений ізраїльською компанією Viber Media (у 2014 році придбаний японською компанією Rakuten), який дозволяє користувачам обмінюватися повідомленнями та здійснювати дзвінки через інтернет [2].

Viber став одним з найпопулярніших засобів цифрової комунікації у Східній Європі завдяки звичному інтерфейсу, кросплатформності, можливістю не лише обмінюватись текстовими повідомленнями, а й здійснювати дзвінки не витрачаючи коштів на мобільний зв’язок, надсилати стікери, створювати спільноти тощо. Ключовим фактором популярності стало впровадження вищезгаданого функціоналу, не маючи конкуренції як такої з боку інших популярних месенджерів.

Даний застосунок має стандартний інтерфейс, поділений на три вертикальні області: панель чатів, область повідомлень та панель інформації про чат. Така структура є звичною для більшості користувачів, тож більшість месенджерів використовують саме її. На рисунку 1.3 зображено головну сторінку Viber включно з вищезгаданими панелями. Viber, як і Telegram,

фактично є односторінковим застосунком, що дозволило значно спростити навігацію. Щодо тем, Viber пропонує лише два варіанти: світла або темна тема з можливістю встановлення відповідно до системної. Можливості налаштовувати власну тему застосунок не надає, лише встановлювати власний фон чатів.

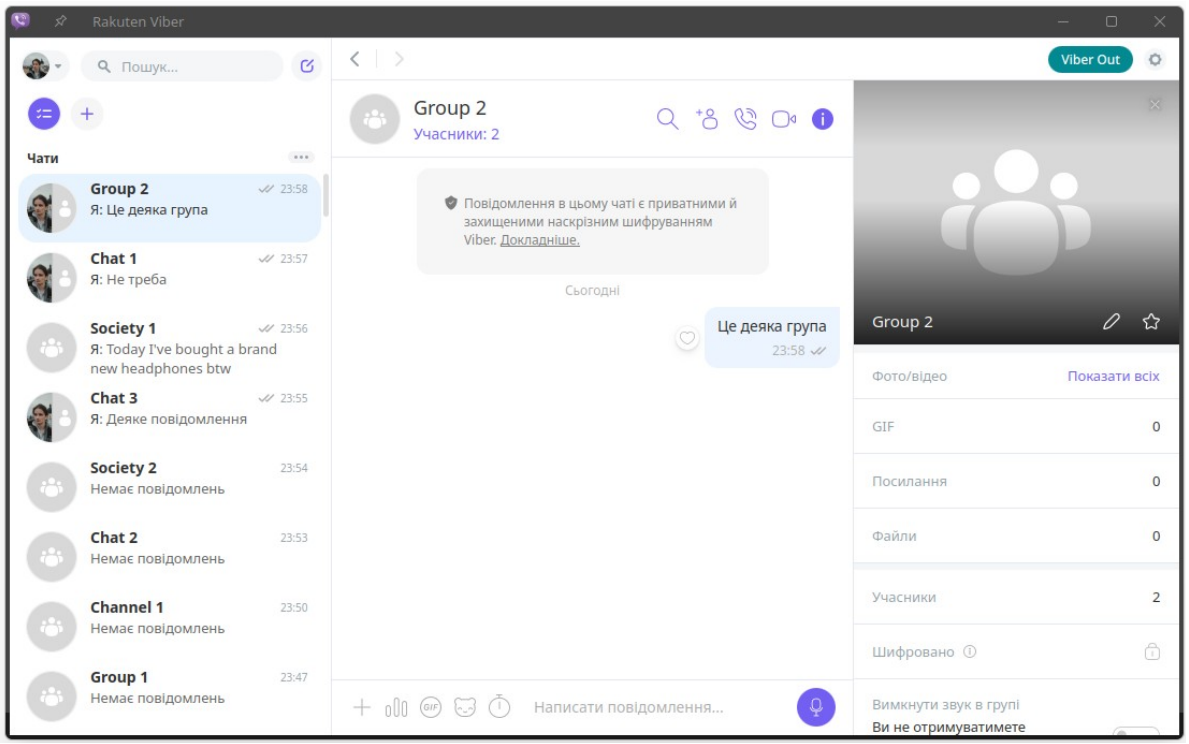

Рисунок 1.3 – Головне вікно Viber

Viber надає користувачам досить широкий функціонал. Окрім надсилання текстових повідомлень, месенджер дозволяє створювати опитування, надсилати медіаконтент, голосові повідомлення, наліпки та GIF-анімації. Щодо зв’язку у режимі реального часу Viber дозволяє здійснювати голосові та відеовиклики як у приватних, так і у групових чатах, а також здійснювати виклики Viber Out, тобто за тарифами мобільного оператора користувача.

Аналогічно до Telegram, Viber підтримує різні типи чатів: приватні та групові чати, канали та спільноти. Функціонал видів чатів не відрізняється

від їх функціоналу в Telegram, окрім спільнот – через їх відсутність у вищезгаданому месенджері. Спільноти у Viber – це групові чати на велику кількість осіб з правом кожного учасника спільноти надсилати повідомлення з функцією коментування іншими користувачами.

Viber також є односторінковим застосунком з певними модальними вікнами, як от вікно налаштувань, що зображене на рисунку 1.4. Застосунок пропонує досить обмежений перелік параметрів, однак присутні налаштування, які задовольняють потреби більшості користувачів.

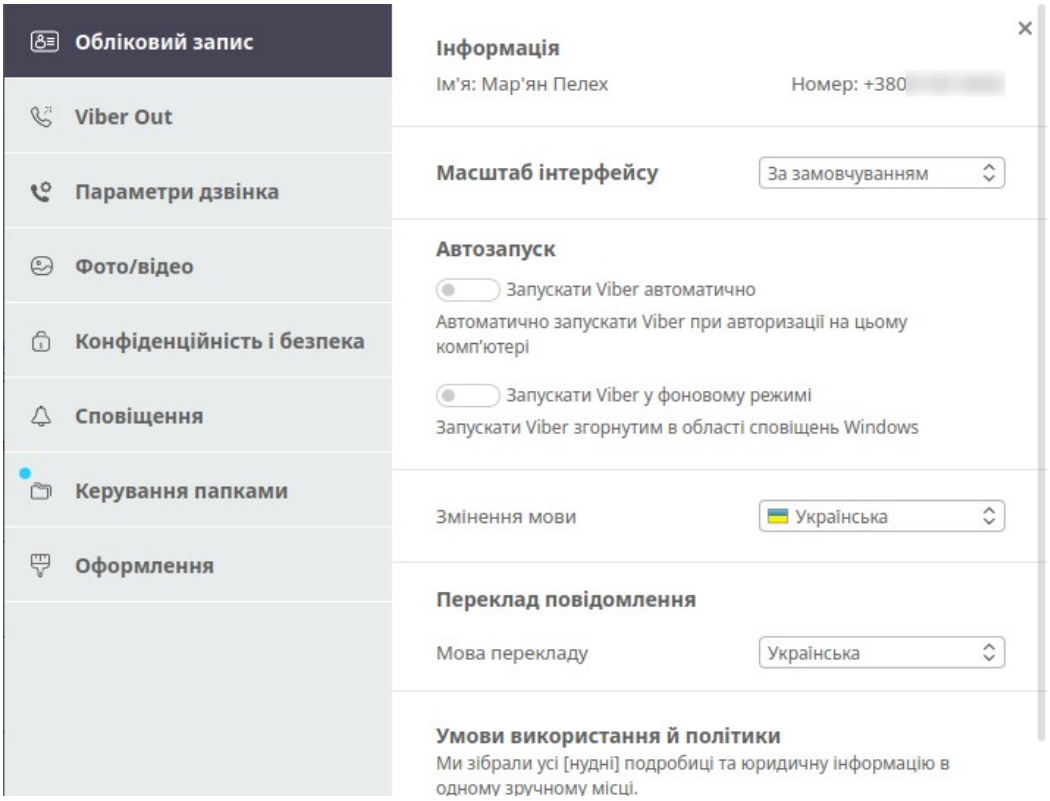

Рисунок 1.4 – Модальне вікно налаштувань Viber

Viber має великий ряд недоліків у своїх UI/UX рішеннях: хоча месенджер написаний за допомогою фреймворку Qt, який має стандартне, оптимізоване вікно вибору файлів, вибір файлів на операційній системі Linux здійснюється за допомогою вікна, написаного вручну, яке відображається некоректно (рис. 1.5).

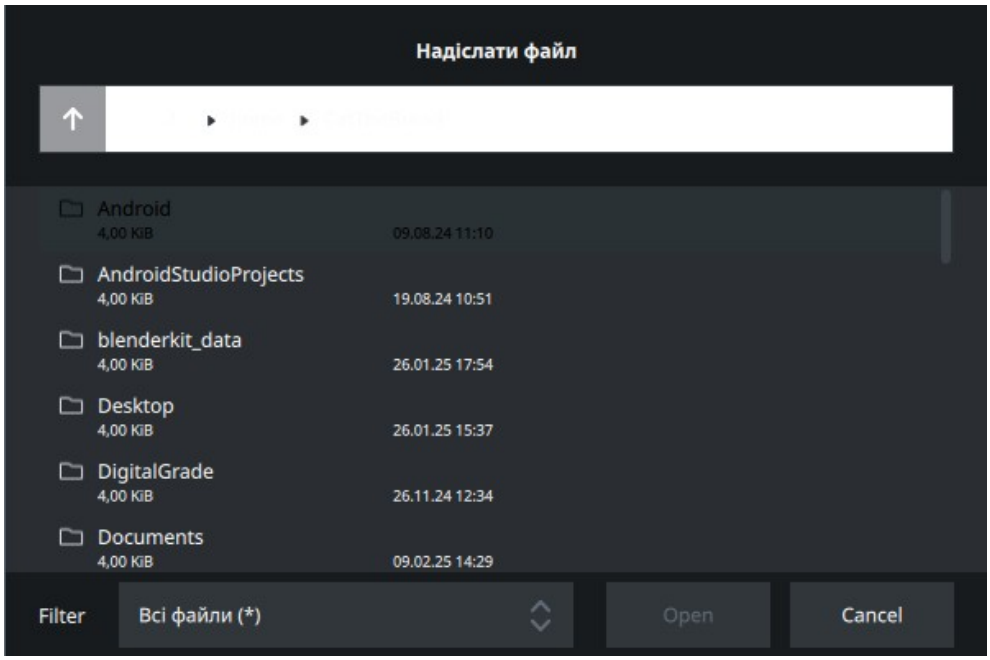

Рисунок 1.5 – Вікно вибору файлів Viber

Хоча Viber використовує наскрізне шифрування для захисту повідомлень користувачів, інші аспекти політики конфіденційності месенджера викликають серйозні занепокоєння. Додаток збирає значний обсяг даних, включаючи особисту інформацію користувачів, метадані їхніх розмов, а також контактні дані осіб, які навіть не зареєстровані в сервісі. Така масштабна практика збору інформації створює ризики неправомірного використання персональних даних та потенційного несанкціонованого доступу третіх сторін до конфіденційної інформації користувачів.

Discord – це масштабна кросплатформенна система обміну повідомленнями, яка з моменту свого заснування у 2015 році здійснила справжню революцію у сфері онлайн-комунікацій. Спочатку орієнтована виключно на геймерську спільноту, платформа швидко еволюціонувала в універсальний інструмент спілкування для найрізноманітніших груп користувачів [3].

Discord пропонує гнучку систему тематичних текстових каналів, організованих у межах серверів (рис. 1.6). Це забезпечує структуровану

комунікацію та ефективний розподіл потоків повідомлень за тематикою чатів. Адміністратори каналів можуть обмежувати доступ до окремих чатів через систему ролей, встановлювати повільний режим для запобігання безперервного потоку повідомлень від користувачів, створювати гілки – тимчасові чати.

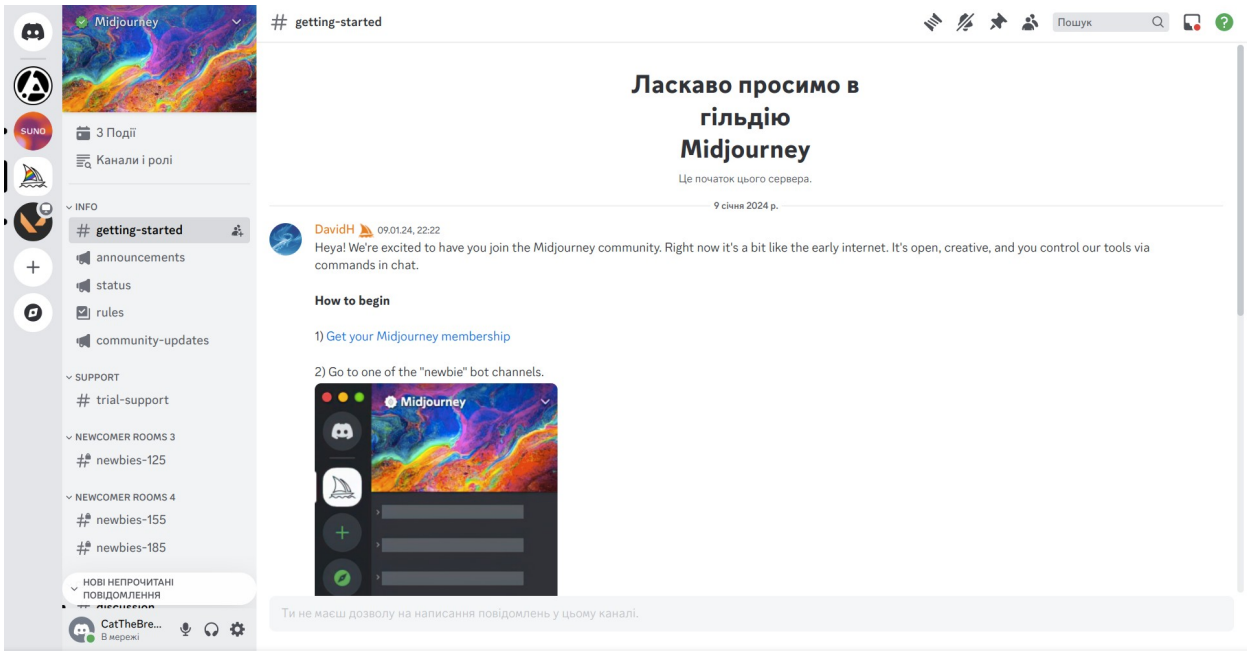

Рисунок 1.6 – Сторінка сервера у Discord

Discord надає розширені можливості форматування текстових повідомлень за допомогою Markdown-синтаксису. Користувачі можуть виділяти текст курсивом, жирним шрифтом, підкреслювати його, створювати багаторівневі списки в межах одного повідомлення. Важливим аспектом форматування є можливість вбудовування блоків коду з підсвічуванням синтаксису різних мов програмування, що робить Discord зручнішим для технічних спільнот. Окрім функцій форматування, вагомим плюсом є система реакцій, яка дозволяє швидко висловити своє ставлення до вмісту повідомлення.

Проблемою може стати не інтуїтивно зрозумілий користувацький інтерфейс платформи, який досить відрізняється від більшої частини

месенджерів. Ліворуч знаходиться дві панелі: крайня – панель каналів, внутрішня – перелік чатів обраного каналу. Зверху знаходиться область заголовка, що включає ім’я користувача або каналу, кнопки здійснення голосового та відеовиклику, поле пошуку в чаті, кнопка відкриття діалогового вікна вхідних повідомлень та кнопка з посиланням на довідку.

Праворуч знаходиться розділ інформації про співрозмовника, якщо це приватний чат, або перелік учасників, якщо це канал.

Поле вводу повідомлення знаходиться внизу сторінки та додатково включає кнопку вкладення файлів, надсилання подарунків, GIF-зображень, наліпок та емоджі. Доступна можливість починати мініігри у чаті.

Область чату займає увесь вільний простір у центрі екрана. Повідомлення розтягуються на всю ширину області, вирівнюючись за лівим краєм. При наведенні на об’єкт повідомлення з’являється меню швидких реакцій.

Discord пропонує широкий вибір тем для налаштування вигляду застосунку, однак лише для користувачів з платною підпискою на платформу.

Увесь інтерфейс Discord є фактично вебсайтом загорнутим у десктопний застосунок за допомогою фреймворку ElectronJS [4]. Це спричиняє серйозні недоліки в ефективності й стабільності системи: застосунок довго завантажується, часто працює з затримками. Це також призводить до значного споживання оперативної пам’яті та ресурсів центрального процесора, що може негативно впливати на продуктивність, особливо на слабких пристроях.

У додатку А наведено порівняльний аналіз безпеки та оптимізації описаних месенджерів.

Проведений аналіз месенджерів дозволяє виокремити функціонал, який пропонують аналоги, а також встановити перелік аспектів, які роблять описані месенджери популярними. Було визначено й недоліки месенджерів, що дозволяє врахувати їх у розробці даного проєкту.

1.3 Аналіз вимог до програмного засобу та постановка завдання

Зважаючи на стрімкий розвиток IT та постійне зростання кількості месенджерів і соціальних мереж, питання безпеки постає критично важливим. Багато популярних сервісів мають сумнівний рівень захисту даних або взагалі нехтують конфіденційністю. Це створює ризики витоку інформації, що ставить під загрозу безпеку персональних даних користувачів.

Крім того, наразі не існує повноцінного українського месенджера, який би поєднував зручність використання, широкий функціонал, аналогічний популярним сервісам, та достатній рівень безпеки. Створення такої системи дозволить задовольнити потреби користувачів у надійному і захищеному спілкуванні, забезпечуючи збереження особистих даних.

Для успішної реалізації месенджера важливо чітко визначити, які функції, характеристики та рівень безпеки будуть гарантувати його зручність та надійність для користувачів. На основі отриманої інформації про месенджери-аналоги було визначено перелік вимог до власного ПЗ.

У першу чергу месенджер повинен бути кросплатформним, тобто працювати як мінімум на двох операційних системах, наприклад Windows та Linux. Користувацький інтерфейс застосунку має бути зручним та інтуїтивно зрозумілим для користувачів.

Визначено наступні нефункціональні вимоги до програмного продукту:

– месенджер повинен забезпечувати швидке відсилання повідомлень при стабільному з'єднанні;

– додаток має бути оптимізований для швидкої та стабільної роботи на пристроях різної потужності;

– для захисту повідомлень має застосовуватися E2EE, а для персональних даних використовуватись RSA шифрування;

– додаток повинен ефективно споживати системні ресурси пристрою, у фоновому режимі використання повинне зводитися до мінімального;

– часто використовувані дані (повідомлення, аватарки користувачів, медіаконтент) повинен кешуватися для швидшого завантаження сторінок застосунку;

– користувачі повинні мати можливість створювати та зберігати власні теми застосунку;

– повинна забезпечуватись можливість використання шаблонів тем з відкритих джерел;

– месенджер повинен підтримувати локалізацію принаймні на українській та англійській мовах для зручності користувачів;

– застосунок має мати зручний та комфортний GUI, привабливий для користувачів.

Окрім нефункціональних вимог, були виокремлені функціональні вимоги до месенджера:

– користувачі повинні мати можливість реєструватися за номером телефону або електронною поштою;

– застосунок повинен підтримувати відсилання текстових повідомлень, медіафайлів, використання emoji та наліпок, а також надавати можливість редагування та видалення повідомлень;

– застосунок повинен підтримувати систему реакцій на повідомлення за допомогою emoji;

– повинна бути можливість надсилати відкладені повідомлення за вказаною датою і годиною;

– месенджер повинен дозволяти відсилання фото, відео, аудіо та документів з обмеженням розміру файлів до 1000 МБ;

– користувачі повинні мати можливість створювати групи до 10000 учасників та керувати правами учасників групи;

– передбачається створення каналів без обмежень у кількості учасників для односторонньої комунікації, а також можливість коментування постів у каналах;

– користувачі повинні мати можливість приховувати свій статус в мережі та блокувати небажані контакти;

– додаток повинен забезпечувати швидкий пошук повідомлень та контактів;

– месенджер повинен дозволяти швидко пересилати повідомлення до інших месенджерів.

Також месенджер повинен забезпечувати синхронізацію історії повідомлень між різними пристроями, щоб користувачі могли продовжувати спілкування без втрати даних.

Для забезпечення високого рівня безпеки передбачається використання наскрізного шифрування для всіх повідомлень та RSA для персональних даних користувачів.

Додатково, для підвищення безпеки облікових записів, можливе використання двофакторної автентифікації (2FA). Це унеможливить доступ до облікового запису навіть у разі компрометації пароля, забезпечуючи високий рівень захисту приватної інформації.

Провівши аналіз наявних рішень, визначено ключові недоліки популярних месенджерів, зокрема недостатній рівень безпеки, ризики маніпуляцій з конфіденційними даними та неоптимальне споживання системних ресурсів.

На основі проведеного дослідження сформульовано перелік функціональних та нефункціональних вимог до кінцевого продукту, що має забезпечити високий рівень конфіденційності, кросплатформність, інтуїтивний користувацький інтерфейс та достатній функціонал для щоденного використання. Реалізація цих вимог дозволить створити конкурентоспроможний месенджер з перспективами розвитку як на локальному, так і на глобальному ринку.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування структури системи

Проєктування є одним з найважливіших етапів у розробці застосунків, адже воно дозволяє встановити чіткі обмеження і визначити способи реалізації запланованого функціоналу.

Від продуманості проєкту залежить не лише ефективність системи, але і її стійкість до кіберзагроз, масштабованість та зручність використання для кінцевого користувача.

Проєктування структури системи має критичне значення позаяк воно визначає взаємодію між системними компонентами, способи досягнення належного рівня безпеки та розмежування рівнів доступу.

На основі аналізу вимог та тенденцій у розробці комунікаційних систем було обрано клієнт-серверну архітектуру, у зв’язку з надійністю даної архітектури в плані зберігання даних і простотою масштабування. Однорангову архітектуру (P2P) було відкинуто через відсутність централізованого контролю над безпекою даних та ризик втрати інформації при збоях пристроїв користувачів [5].

Серверна частина оброблятиме запити, що надходять з клієнтської частини програми, взаємодіятиме з базою даних, відправлятиме сигнали з певною інформацією до підключених пристроїв користувачів.

Клієнтський застосунок не буде мати жодного прямого доступу до бази даних, уся комунікація з БД здійснюватиметься через серверну частину. Застосунок, який буде безпосередньо доступним користувачу, передбачає лише відображення інтерфейсу для взаємодії з системою і логіку відправлення/отримання запитів.

На рисунку 2.1 зображено загальну схему взаємодії компонентів обраної архітектури.

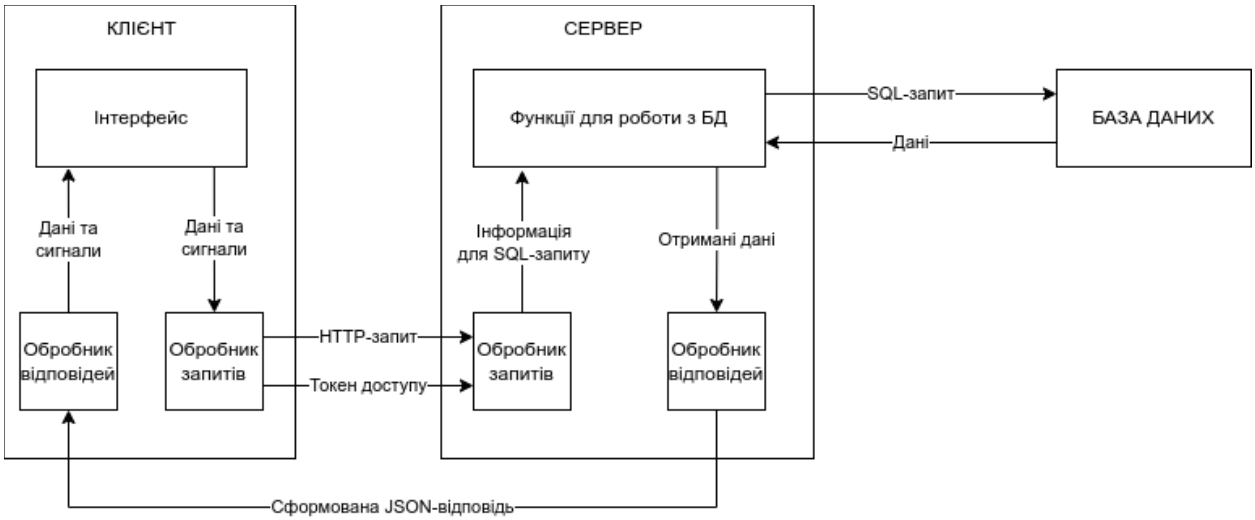

Рисунок 2.1 – Схема архітектурної взаємодії

Для впровадження базової безпеки планується використовувати JSON Web Tokens (JWTs) – стандарт (RFC 7519) безпечного передавання інформації як JSON-об’єкта між компонентами системи [6][7].

Наведені токени створюються і підписуються секретним ключем при аутентифікації користувача і використовуються при кожному запиті до сервера. Планується використовувати два токени: access-токен та refresh-токен.

Access-токен використовується для перевірки особи користувача і передається при кожному запиті у заголовку HTTP «Authorization». Токен доступу має короткий термін дії для запобігання його використання у разі викрадення.

Refresh-токен передається на сервер у разі закінчення терміну дії access-токена для створення нового токена доступу. Refresh-токен також має термін дії. Після його закінчення користувач повинен увійти знову.

На рисунку 2.2 зображено діаграму класів серверної компоненти системи месенджера. Діаграма відображає основні компоненти сервера та їх взаємозв'язки.

Центральним елементом буде клас RestAPI, який забезпечуватиме HTTP-інтерфейс для взаємодії з клієнтами. Він міститиме методи для аутентифікації користувачів, авторизації, обміну повідомленнями тощо.

Клас Database відповідатиме за роботу, пов’язану з отриманням та збереженням інформації у базі даних.

Встановлювати з’єднання через вебсокети та обробляти відповідні запити буде клас WebSocketHandler.

Допоміжний клас Crypt забезпечуватиме здійснення криптографічних операцій, а клас Utils міститиме методи для валідації даних, конвертації типів даних тощо.

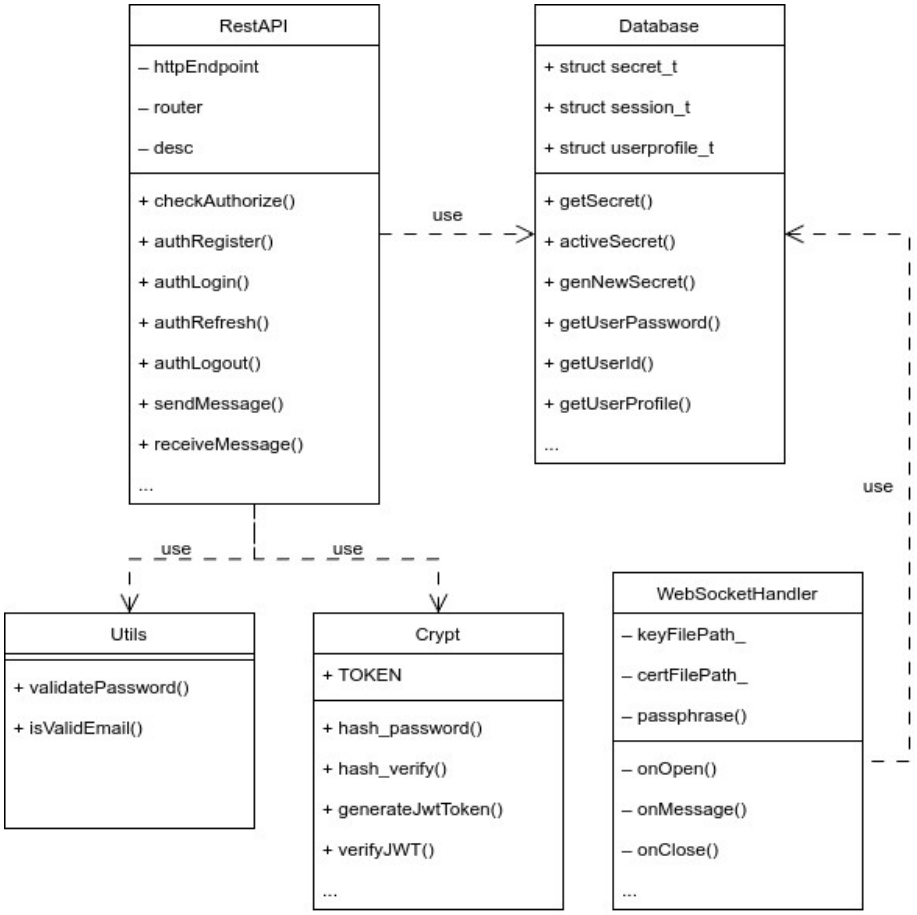

Рисунок 2.2 – Діаграма класів серверної частини

Клієнтська частина системи міститиме перелік незалежних один від одного класів для створення і відправлення запитів до сервера, а також

обробку його відповідей. Кожен клас буде відповідати лише за одну область: аутентифікацію, повідомлення, чати тощо.

Зокрема, клас AuthManager відповідатиме за процес аутентифікації, включаючи роботу зі збереженням JWT-токенів. Загальний процес аутентифікації користувача зображено на рисунку 2.3.

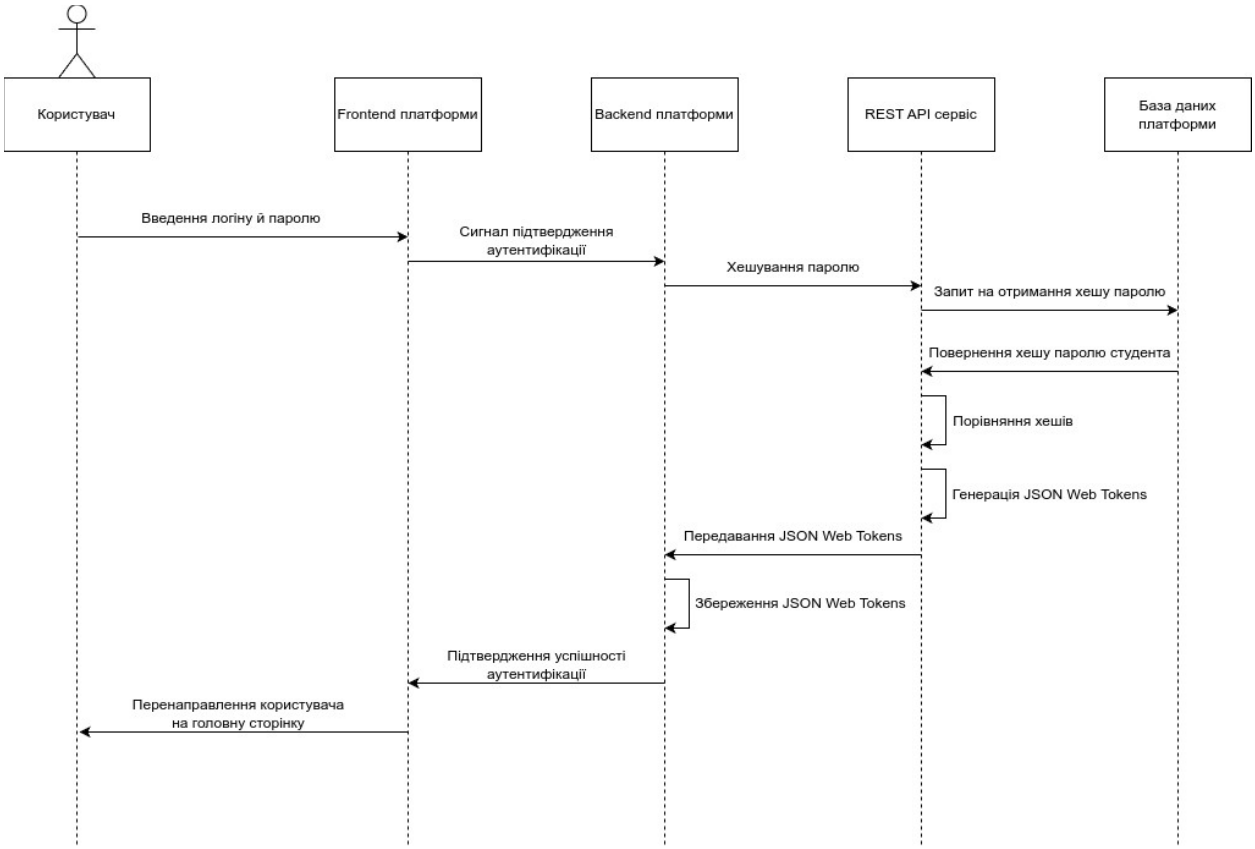

Рисунок 2.3 – Діаграма послідовності аутентифікації користувача

На рисунку 2.4 зображено схему надсилання повідомлень у системі. Процес ініціюватиметься, коли користувач формуватиме текстове повідомлення та здійснюватиме відправку повідомлення. Клієнтський застосунок первинно перевірятиме термін дії токена автентифікації: у разі виявлення його протермінування викликатиметься функція оновлення токена доступу за допомогою надсилання токена оновлення. Якщо токен буде дійсним, система застосовуватиме асиметричне шифрування з використанням публічного ключа одержувача, після чого зашифровані дані разом із токеном передаватимуться на сервер, де здійснюватиметься остаточна валідація

токена доступу. При успішній перевірці сервер використовуватиме SQL-запит для збереження повідомлення в базі даних та ініціюватиме асинхронне відправлення push-сповіщення.

На фінальному етапі клієнтський інтерфейс одержувача отримуватиме зашифрований контент, розшифровуватиме його за допомогою приватного ключа та відображатиме отримане повідомлення в графічному інтерфейсі застосунку.

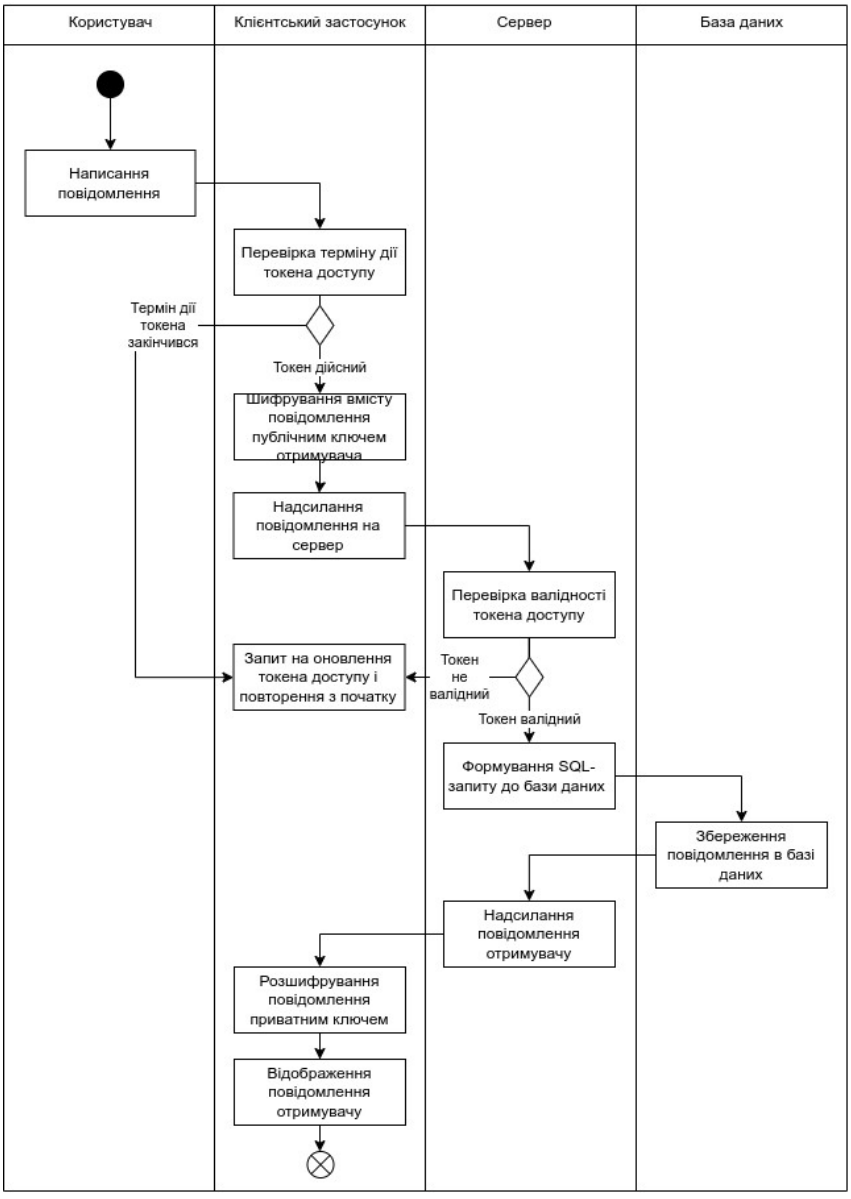

Рисунок 2.4 – Схема надсилання повідомлень у системі

Для отримання повідомлень клієнтом-одержувачем буде використовуватися технологія вебсокетів. Вебсокети дозволяють утримувати неперервний двосторонній канал зв’язку, що надає можливість не лише отримувати запити з клієнта на сервер, а й відправляти дані напряму з сервера до клієнта в режимі реального часу без відповідного запиту клієнтом [8].

Для забезпечення ефективної реалізації поставлених функціональних та нефункціональних вимог критично важливим етапом проєктування системи є виділення основних варіантів використання для розподілення можливостей користувачів.

На рисунку 2.5 показано діаграму варіантів використання системи месенджера користувачами.

Рисунок 2.5 – Діаграма варіантів використання

Система буде орієнтованою виключно на кінцевих користувачів. Інформація, що циркулює в системі буде зашифрованою і розшифровуватися лише на пристроях користувачів, що унеможливлює читання будь-якої інформації сторонніми користувачами, у тому числі адміністраторами системи, тому, наприклад, модерація контенту фізично є неможливою.

2.2 Проєктування бази даних

Проєктування бази даних є вкрай важливим аспектом у розробці шифрованого месенджера, адже добре структурована БД гарантує швидшу взаємодію з сервером, мінімізацію використання дискового простору, а також її масштабованість.

На основі поставлених вимог і структури проєкту визначено перелік ключових таблиць бази даних. Ключовими будуть таблиці для керування користувачами, їхніми сесіями, чатами та повідомленнями. Таблиця users фіксуватиме дані облікових записів, хеші паролів та унікальні ідентифікатори. Активні сесії користувачів реєструватимуться в таблиці sessions.

Збереження інформації про чати буде реалізовано через таблицю chats, яка визначатиме типи комунікації (приватні чати, групи, канали) і зберігатиме загальну інформацію про чати. Безпосередньо вміст повідомлень, включаючи текст та медіа, зберігатиметься в таблиці messages, для забезпечення зв’язку повідомлень з чатами буде використовуватися таблиця chat_messages.

Архітектура бази даних спроєктована так, що всі чутливі дані, включаючи зміст повідомлень, імена користувачів та метадані чатів, зберігаються у зашифрованому вигляді. Це означає, що навіть в разі отримання несанкціонованого доступу до бази даних зловмисники не зможуть отримати зміст спілкування користувачів.

Перелік визначених сутностей наведено у таблиці 2.1. В таблиці наведено назви сутностей, назви таблиць сутностей, типи первинних ключів та функціональне призначення кожної сутності.

Таблиця 2.1 – Сутності бази даних

Назва сутності	Назва таблиці	Первинний ключ	Примітки
Користувач	users	id (UUID)	Зберігає дані облікових записів
Сесія	sessions	id (UUID)	Відстежує активні сесії користувачів
Чат	chats	id (UUID)	Містить інформацію про чати
Повідомлення	messages	id (UUID)	Зберігає вміст повідомлень
Чат-Повідомлення	chat_messages	id (UUID)	Встановлює зв’язок між повідомленнями та чатами
Секрет	secrets	id (INT)	Зберігає ключі для шифрування
Учасник чату	user_chat	chat_id + user_id (UUID)	Визначає учасників чатів та їхні ролі

Для кожної сутності, зазначеної у таблиці 2.1, визначено таблиці атрибутів. Для сутності «Користувач» визначено перелік атрибутів, наведений у таблиці 2.2. Атрибути підібрані з урахуванням безпеки та надійності ідентифікації, наприклад, унікальність полів «email» та «username» запобігає створенню дублікатів облікових записів, зберігання пароля у вигляді хешу є однією з основних мір безпеки. Поля «name» та «surname» є опціональними, що дозволяє користувачам зберігати анонімність, або ж, для зручності їхньої ідентифікації у чатах, користувачі можуть надати відповідні дані.

Таблиця 2.2 – Таблиця атрибутів сутності «Користувач»

Атрибут	Назва поля	Тип даних	Примітки
ID	id	UUID	Генерується автоматично (uuid())
Електронна пошта	email	VARCHAR(64)	NOT NULL, унікальний
Псевдонім користувача	username	VARCHAR(16)	NOT NULL, унікальний
Пароль	password	VARCHAR(256)	NOT NULL
Ім’я	name	VARCHAR(32)	NULL

Продовження таблиці 2.2

Атрибут	Назва поля	Тип даних	Примітки
Прізвище	surname	VARCHAR(32)	NULL
Аватар	image	VARCHAR(256)	DEFAULT: «assets/noimage.png»
Статус	status	VARCHAR(16)	DEFAULT: «Offline»
Дата народження	birthdate	DATE	NULL
Дата реєстрації	created_at	TIMESTAMP	DEFAULT: current_timestamp()

У таблиці 2.3 наведено перелік атрибутів для сутності «Сесія». Сутність використовує такі атрибути, як ідентифікатор сесії, ідентифікатор користувача, останній час оновлення токена доступу, час останньої авторизації та назву операційної системи, з якої здійснено вхід – для простішої ідентифікації сесії користувачем.

Таблиця 2.3 – Таблиця атрибутів сутності «Сесія»

Атрибут	Назва поля	Тип даних	Примітки
ID сесії	id	UUID	Генерується автоматично (uuid()).
ID користувача	user_id	UUID	Зовнішній ключ до users.id.
Час оновлення	updated_at	TIMESTAMP	DEFAULT: current_timestamp().
Остання авторизація	last_auth	DATETIME	NOT NULL.
Операційна система	os	VARCHAR(32)	NOT NULL.

Аналогічна таблиця атрибутів була створена для сутності «Чат» (табл. 2.4). У даній сутності атрибути також визначалися з урахуванням необхідності у надійності й безпеці, наприклад, тип чату обмежений значеннями «chat», «channel» та «group», що визначає чіткі правила чатів і виключає будь-які інші значення.

Таблиця 2.4 – Таблиця атрибутів сутності «Чат»

Атрибут	Назва поля	Тип даних	Примітки
ID чату	id	UUID	Генерується автоматично (uuid())
Назва	title	VARCHAR(64)	NULL
Тип чату	type	ENUM	Допустимі значення: «chat», «channel», «group»; DEFAULT: «chat»
Опис	description	TEXT	NULL
Аватар чату	group_img	VARCHAR(255)	DEFAULT: «assets/noimage.png»
Дата створення	created_at	TIMESTAMP	DEFAULT: current_timestamp()

Таблиця 2.5 наводить перелік атрибутів сутності «Повідомлення». Сутність включає такі атрибути, як ідентифікатор повідомлення, ідентифікатор відправника, власне текст повідомлення і поле медіа. Атрибут «sender_id» має зв’язок з полем «id» таблиці users для запобігання створення повідомлень з відсутнім ідентифікатором користувача.

Таблиця 2.5 – Таблиця атрибутів сутності «Повідомлення»

Атрибут	Назва поля	Тип даних	Примітки
ID повідомлення	id	UUID	Генерується автоматично (uuid())
Ідентифікатор відправника	sender_id	UUID	Зовнішній ключ до users.id (NOT NULL)
Текст	text	TEXT	NULL
Медіа	media	LONGTEXT	JSON з даними про медіафайли

Атрибути сутності «Чат-Повідомлення» наведено у таблиці 2.6. Таблиця пов’язує повідомлення з відповідним чатом, додаючи час надсилання й маркер чи повідомлення є пересланим з іншого чату.

Таблиця 2.6 – Таблиця атрибутів сутності «Чат-Повідомлення»

Атрибут	Назва поля	Тип даних	Примітки
ID зв’язку	id	UUID	Генерується автоматично (uuid()).
ID повідомлення	message_id	UUID	Зовнішній ключ до messages.id (NOT NULL).
ID чату	chat_id	UUID	Зовнішній ключ до chats.id (NOT NULL).
Переслане	forwarded	TINYINT(1)	DEFAULT: 0 (не переслане).
Час додавання	created_at	TIMESTAMP	DEFAULT: current_timestamp().

У таблиці 2.7 перелічено атрибути сутності «Учасник чату». Сутність включає ідентифікатори чату, користувача, а також булеву змінну чи є користувач адміністратором чату, присвоюючи йому відповідні права.

Таблиця 2.7 – Таблиця атрибутів сутності «Учасник чату»

Атрибут	Назва поля	Тип даних	Примітки
ID користувача	user_id	UUID	Зовнішній ключ до users.id (NOT NULL).
ID чату	chat_id	UUID	Зовнішній ключ до chats.id (NOT NULL).
Адміністратор	admin	TINYINT(1)	DEFAULT: 0

Сутність «Секрет» містить у собі ідентифікатор секретного ключа, власне ключ, булеву змінну чи активний цей ключ і термін дії ключа. Перелік атрибутів з відповідними примітками та типами даних наведено у таблиці 2.8.

Таблиця 2.8 – Таблиця атрибутів сутності «Секрет»

Атрибут	Назва поля	Тип даних	Примітки
ID секрету	id	INT UNSIGNED	AUTO_INCREMENT.
Секрет	secret	VARCHAR(1024)	NOT NULL.
Активність	is_active	TINYINT(1)	NOT NULL (1 — активний, 0 — неактивний).
Термін дії	expires_at	TIMESTAMP	NOT NULL.

Для унеможливлення передбачення ідентифікаторів в ході атак перебором значень, у більшості таблиць для атрибута «id» обрано тип даних UUID – 36-значний рядок випадкових символів, що генерується автоматично.

Обмеження NOT NULL та UNIQUE для критичних полів (email, username тощо) захищатиме від некоректних або дубльованих даних, зберігаючи цілісність БД.

Значення за замовчуванням (DEFAULT) зменшують ризик порожніх полів і спрощують розробку, автоматично заповнюючи необов'язкові атрибути, що важливо для масштабованості та стабільності системи.

Усі таблиці використовуватимуть систему зовнішніх ключів, яка гарантуватиме цілісність даних і запобігатиме виникненню конфліктів. Така архітектура дозволятиме ефективно керувати БД та забезпечуватиме швидкий пошук інформації.

Зв'язки між таблицями бази даних спроєктовано за допомогою розробки Entity-Relationship діаграми. На рисунку 2.6 зображено розроблену ER-діаграму бази даних.

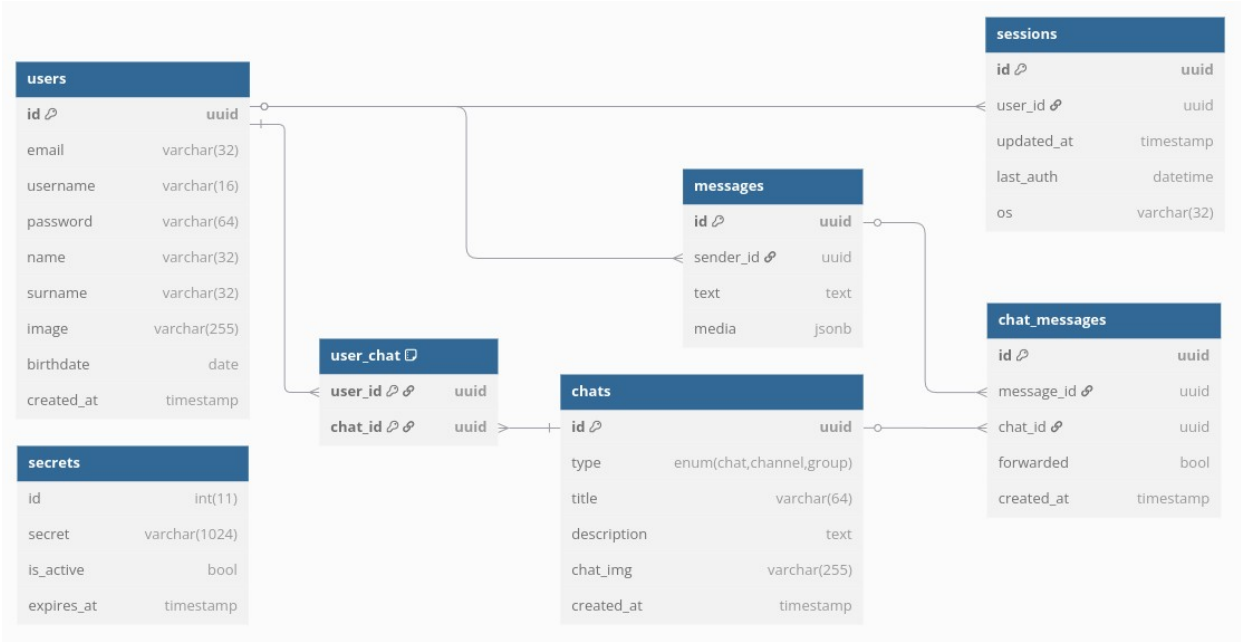

Рисунок 2.6 – ER-діаграма бази даних

У результаті проєктування, модель бази даних задовольняє вимоги розробки шифрованого кросплатформного месенджера з можливістю її ефективного масштабування в майбутньому, додаючи нові таблиці та розширюючи функціонал самого застосунку.

2.3 Проєктування інтерфейсу

Зручний, інтуїтивно-зрозумілий користувацький інтерфейс привертає увагу користувачів до застосунку у першу чергу. Інтерфейс має не лише виглядати естетично, але й не бути перевантаженим зайвими елементами, знижуючи поріг входження нового користувача.

При проєктуванні інтерфейсу акцентовано на мінімалістичність та зручність: іконки мають бути інтуїтивними, а розташування елементів повинне повторювати звичну для користувачів структуру популярних месенджерів. Важливим елементом інтерфейсу є доступність – за замовчуванням у систему повинна бути встановлена світла та темна тема, а також планується можливість для користувачів створювати власні тематичні оформлення. Це важливо, адже користувачі можуть мати індивідуальні вподобання стосовно колірної теми застосунку.

Основною темою буде обрана темна, що виправдано високою популярністю нічного режиму серед користувачів месенджерів та соціальних мереж. Темна тема буде включати темно-сірі відтінки для фону інтерфейсу, що сприятиме зменшенню навантаження на очі в умовах низького освітлення.

Для основного тексту планується використання білого кольору, що створить необхідний контраст із темним фоном. Такий підхід дозволить зберегти високий рівень читабельності та мінімізувати можливість візуального напруження під час тривалого використання системи.

Акцентним елементом дизайну буде колір #b4245b (малиновий), який планується застосовувати для виділення важливих елементів інтерфейсу та інтерактивних кнопок. Крім того, для виділення власних повідомлень

користувачів передбачається використання лінійного градієнту, що додасть інтерфейсу сучасного вигляду та покращить візуальне сприйняття повідомлень.

Важливою частиною інтерфейсу буде адаптивна композиція елементів, задля забезпечення зручного відображення сторінок месенджера на різних типах пристроїв.

За допомогою програмного забезпечення Figma спроєктовано макети сторінок застосунку. Макет вікна месенджера зображений на рисунку 2.7. У верхній частині вікна знаходиться область заголовка. Ліворуч розташована панель з переліком чатів, праворуч – робоча область. Внизу робочої області розташована область написання повідомлень.

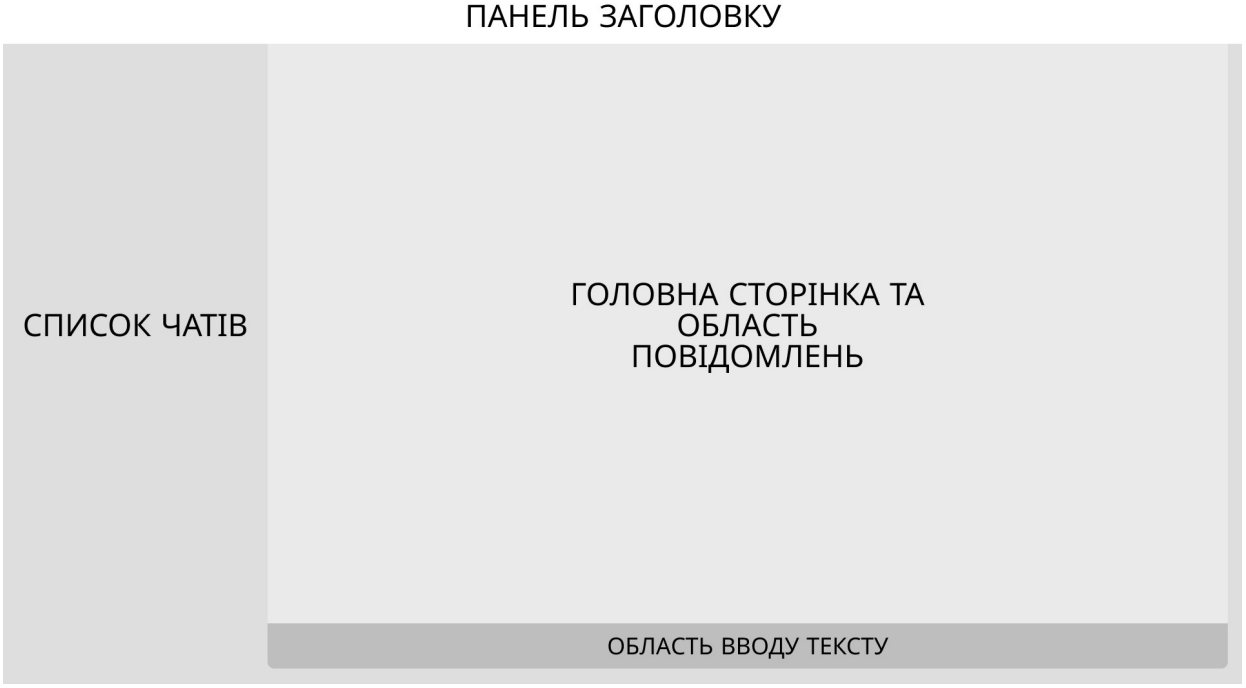

Рисунок 2.7 – Макет вікна месенджера

На рисунку 2.8 зображено макет сторінки аутентифікації. Ліворуч розташовується панель з формою входу, праворуч – область для розміщення певного зображення.

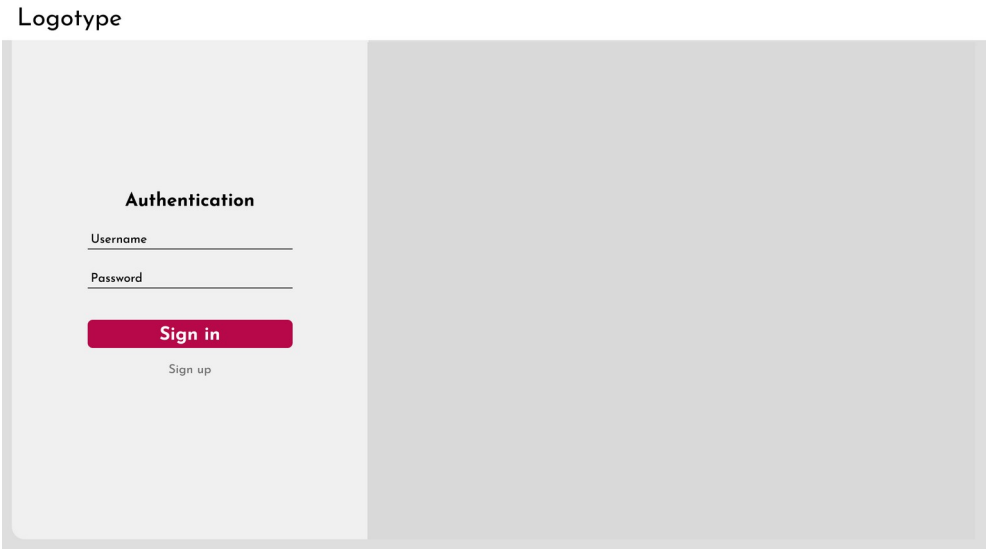

Рисунок 2.8 – Макет сторінки аутентифікації

Рисунок 2.9 зображає макет головної сторінки месенджера, який відповідає попередньо розробленому макету. На головній сторінці месенджера у верхній частині ліворуч розміщується логотип і праворуч – відображення імені користувача з його зображенням (аватаром). Бічна панель включає перелік чатів, де кожен чат відображається у вигляді аватара співрозмовника або зображення групи, імені чату, до якого додається фрагмент останнього повідомлення та час його надходження. Активні чати виділяються іншим кольором фону та відповідним підкресленням.

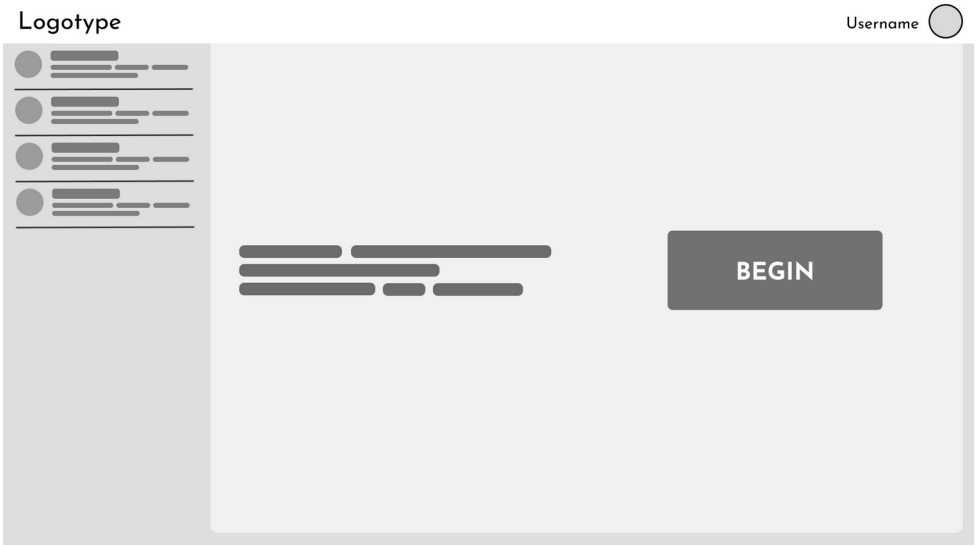

Рисунок 2.9 – Макет головної сторінки месенджера

Сторінка чату зображена на рисунку 2.10. Угорі сторінки розміщується панель із назвою чату або ім'ям співрозмовника. Історія повідомлень організована так, що старіші повідомлення знаходяться вгорі, а новіші – внизу, при цьому повідомлення інших користувачів вирівнюються ліворуч і мають світлий фон, тоді як власні повідомлення – праворуч із темнішим або градієнтним оформленням. До кожного повідомлення додається час його відправлення і позначки про статус доставлення. Нижня частина сторінки містить область вводу тексту, де ліворуч розташовується іконка для додавання вкладень, посередині знаходиться розширюване текстове поле, а праворуч – кнопка для відправлення повідомлення.

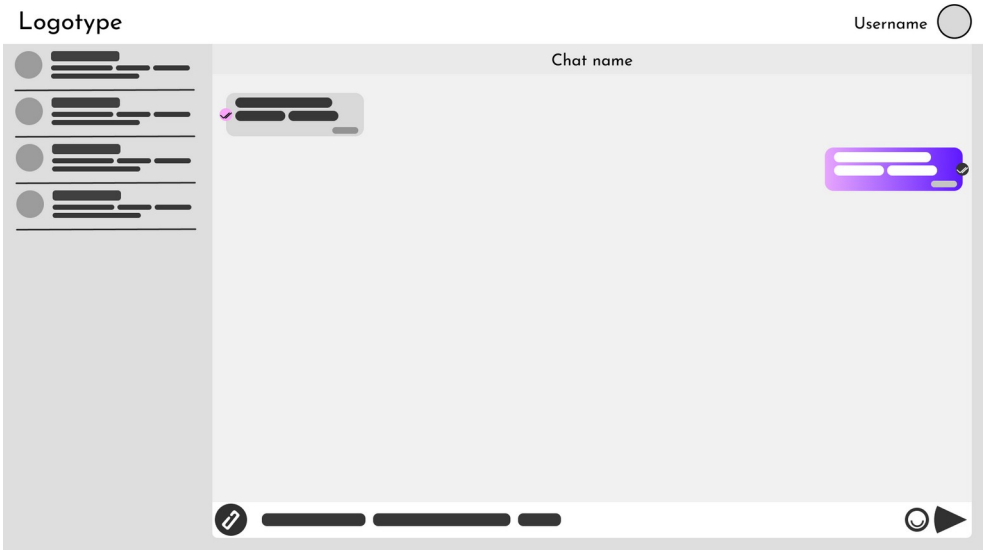

Рисунок 2.10 – Макет сторінки чату

Інтерфейс месенджера поєднує інтуїтивність, доступність та стиль, забезпечуючи користувачам комфортну взаємодію з застосунком. Акцент на мінімалізм робить систему інтуїтивною для нових користувачів, а темна тема не лише відповідає сучасним трендам, але й знижує навантаження на зір.

Проєктування месенджера охопило всі важливі аспекти – спроєктована архітектура системи, база даних та дизайн інтерфейсу створюють основу для подальшої розробки месенджера, забезпечуючи надійність та зручність користування застосунком.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Опис засобів реалізації

Від засобів реалізації залежить продуктивність і масштабованість системи, швидкість її розробки та можливість застосунку працювати на різних платформах. Для реалізації кросплатформного шифрованого месенджера було обрано перелік мов програмування та технологій, здатних забезпечити необхідний функціонал без погіршення продуктивності та безпеки.

Обрані засоби реалізації:

C++ – основна мова розробки, обрана через високу швидкість роботи як строго типізованої мови, велику кількість бібліотек та фреймворків, можливість безпосередньо керувати пам'яттю та забезпечувати високу продуктивність системи.

Qt6 – кросплатформний фреймворк для розробки застосунків на Windows, Linux, macOS, Android, iOS та інших платформах. Надає широкий перелік бібліотек для роботи з мережею, файловими системами, графічним інтерфейсом тощо. Вирізняється високою продуктивністю та стабільністю.

QML (Qt Modeling Language) – декларативна мова розмітки, що надається фреймворком Qt. Перевагами є підтримка апаратного графічного прискорення, простота інтеграції з C++, можливість створення сучасного адаптивного користувацького інтерфейсу з анімаціями та динамічними елементами без докладання надлишкових зусиль.

OpenSSL – криптографічна бібліотека, призначена для реалізації алгоритмів шифрування, хешування та інших криптографічних операцій. Забезпечує надійний захист повідомлень та даних користувачів відповідно до сучасних стандартів безпеки.

RESTful API – підхід до архітектури побудови вебсервісів, що відзначається простотою використання, масштабованістю та поширеністю у

системах з високим навантаженням. Забезпечує ефективну взаємодію між клієнтом та сервером через стандартні HTTP-методи.

Pistache – високопродуктивний HTTP-фреймворк для C++, призначений для створення RESTful вебсервісів. Характеризується низькою затримкою, підтримкою асинхронного програмування та ефективним використанням системних ресурсів.

uWebSockets – бібліотека для роботи з вебсокет-з'єднаннями, призначена для підтримки постійного з'єднання між клієнтом і сервером. Забезпечує постійний канал обміну повідомленнями між серверною програмою та клієнтським застосунком з мінімальною затримкою, що дозволяє отримувати повідомлення на пристрої-клієнті в режимі реального часу.

AES-256-GCM – симетричний блоковий алгоритм шифрування, з ключем довжиною 256 біт у режимі GCM, що забезпечує як шифрування, так і автентифікацію даних завдяки тегу автентифікації.

RSA – асиметричний алгоритм шифрування, використовує пару публічний-приватний ключ. Ключ довжиною 2048 або 4096 біт. Дозволяє досягти високого рівня безпеки, однак недоліком є низький ліміт вхідного тексту, розміром 245 байтів [12].

MariaDB – високопродуктивна СКБД з відкритим кодом, є розгалуження MySQL. Відзначається високою швидкодією, масштабованістю, покращеною безпекою та надійністю. Забезпечує ефективне зберігання, організацію та швидкий пошук структурованих даних завдяки оптимізованим механізмам індексування.

PHPMyAdmin – вебінструмент для адміністрування баз даних MySQL та MariaDB. Надає інтуїтивний графічний інтерфейс для керування структурою БД, виконання SQL-запитів, управління користувачами та правами доступу.

CMake – система автоматизації збірки програмного забезпечення, що забезпечує гнучкість конфігурації проєкту, управління залежностями та підтримку різних компіляторів і операційних систем.

Neovim/Neovide – текстовий редактор з розширеними можливостями для розробки, що забезпечує високу швидкість редагування коду завдяки можливості налаштування під індивідуальні потреби розробника.

Visual Studio Code – IDE з широкою підтримкою плагінів, систем контролю версій та інструментами для зручнішої та продуктивнішої розробки.

Обрані технології забезпечують оптимальне поєднання продуктивності, безпеки та зручності розробки, що є необхідним для створення кросплатформного шифрованого месенджера.

3.2 Реалізація системи

3.2.1 Реалізація бази даних

На основі створеної реляційної структури здійснено реалізацію бази даних «crescent_db» у середовищі MariaDB версії 11.6.2. Створення бази даних відбувалося на основі попередньо розробленої концептуальної моделі, яка забезпечує всі необхідні операції для роботи месенджера.

База даних розгортається в контейнері Docker, що зроблено для забезпечення ізольованості, портативності та простоти розгортання. Такий підхід дозволяє легко відтворити робоче середовище на будь-якій системі без необхідності ручного налаштування серверних компонентів [9].

Конфігурація Docker Compose включає два основні сервіси: MariaDB як основну систему управління базою даних та phpMyAdmin для графічного інтерфейсу адміністратора.

Основний сервіс бази даних налаштовано з використанням офіційного образу MariaDB. Конфігурація передбачає автоматичне створення бази даних з ім'ям «crescent_db» та користувача. Сервіс налаштовано на автоматичний

перезапуск у випадку збою. Для збереження даних використовується постійний том, який монтується до директорії «/var/lib/mysql», що гарантує збереження інформації навіть після перезапуску контейнера (лістинг 1).

Лістинг програмного коду 1 – Docker Compose конфігурація для запуску бази даних CrescentDB

```
services:
  mariadb:
    image: mariadb:latest
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: [root_password]
      MYSQL_DATABASE: crescent_db
      MYSQL_USER: [username]
      MYSQL_PASSWORD: [password]
      #TZ: Europe/Kyiv
    ports:
      - "3306:3306"
    volumes:
      - ./mariadb:/var/lib/mysql
      - ./mariadb.cnf:/etc/mysql/my.cnf
    networks:
      - internal
```

Для забезпечення графічного інтерфейсу адміністрування бази даних налаштовано сервіс phpMyAdmin. Цей сервіс автоматично підключається до MariaDB через внутрішню мережу та надає вебінтерфейс на порту 8080. Код конфігурації сервісу наведений у лістингу програмного коду 2.

Лістинг програмного коду 2 – Конфігурація сервісу phpMyAdmin.

```
  phpmyadmin:
     image: phpmyadmin/phpmyadmin
     restart: always
     ports:
       - "8080:80"
     environment:
       PMA_HOST: mariadb
       PMA_USER: [user]
       PMA_PASSWORD: [password]
     networks:
         - internal
networks:
    internal:
        driver: bridge
```

Процес розгортання системи значно спрощується завдяки використанню контейнеризації. Команда «docker-compose up -d», виконана в терміналі Linux, автоматично завантажує необхідні образи, налаштовує мережу та запускає всі сервіси. Такий підхід забезпечує ідентичність робочого середовища на різних машинах та спрощує процес розробки.

Власне реалізація бази даних розпочалася з налаштування часового поясу на UTC для уніфікації роботи з часовими мітками незалежно від географічного розташування сервера. Під час створення таблиць приділено увагу оптимізації структури для забезпечення високої продуктивності системи при численних запитах сервера. Використання ідентифікаторів типу UUID замість автоінкрементних цілих чисел підвищує безпеку системи, оскільки такі ідентифікатори стають непередбачуваними для зловмисників.

Створення основних таблиць бази даних здійснювалося через серію SQL-запитів, які враховують спроєктовані раніше аспекти бази даних. Для створення таблиці користувачів використано запит, наведений у лістингу 3.

Лістинг програмного коду 3 – SQL запит створення таблиці користувачів

```
CREATE TABLE `users` (
 `id` uuid NOT NULL DEFAULT uuid(),
 `email` varchar(64) NOT NULL,
 `username` varchar(16) NOT NULL,
 `name` varchar(32) NOT NULL,
 `surname` varchar(32) DEFAULT NULL,
 `password` varchar(256) NOT NULL,
 `image` varchar(256) NOT NULL DEFAULT 'assets/noimage.png',
 `birthdate` date DEFAULT NULL,
 `created_at` timestamp NOT NULL DEFAULT current_timestamp()
)
```

Структура таблиці чатів розроблена для підтримки різних типів комунікації з можливістю налаштування приватності та додаткових параметрів. Створення цієї таблиці здійснено з використанням запиту з лістингу програмного коду 4.

Лістинг програмного коду 4 – SQL запит створення таблиці чатів

```
CREATE TABLE `chats` (
`id` uuid NOT NULL DEFAULT uuid(),
`type` enum('chat','channel','group','') NOT NULL DEFAULT
'chat',
`public` tinyint(4) NOT NULL DEFAULT 0,
`title` varchar(64) DEFAULT NULL,
`description` text DEFAULT NULL,
`chat_img` varchar(255) NOT NULL DEFAULT
'assets/noimage.png',
`created_at` timestamp NOT NULL DEFAULT current_timestamp())
```

Таблиця повідомлень зберігає текстовий контент та мультимедійні дані у форматі JSON, забезпечуючи гнучкість у роботі з різними типами медіаконтенту. Відповідний SQL-запит наведено у лістингу 5.

Лістинг програмного коду 5 – SQL запит створення таблиці повідомлень

```
CREATE TABLE `messages` (
  `id` uuid NOT NULL DEFAULT uuid(),
  `sender_id` uuid NOT NULL,
  `text` text NOT NULL,
  `media` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
DEFAULT NULL)
```

Зв'язкова таблиця «chat_messages» містить інформацію про належність повідомлень до чатів, включаючи інформацію про пересилання та часові мітки. Структура цієї таблиці, створеної запитом з лістингу програмного коду 6, забезпечує ефективне управління розподілом повідомлень між чатами без дублювання контенту.

Лістинг програмного коду 6 – SQL запит створення зв'язкової таблиці повідомлень чату

```
CREATE TABLE `chat_messages` (
 `id` uuid NOT NULL DEFAULT uuid(),
 `message_id` uuid NOT NULL,
 `chat_id` uuid NOT NULL,
 `forwarded` tinyint(1) NOT NULL DEFAULT 0,
 `forwarded_by` uuid DEFAULT NULL,
 `created_at` timestamp NOT NULL DEFAULT current_timestamp()
)
```

Система сесій користувачів реалізована з урахуванням сучасних вимог безпеки та можливості відстеження активності. Кожна сесія містить інформацію про операційну систему користувача та часові мітки останньої авторизації, що дозволяє користувачам контролювати доступ до облікових записів.

Система безпеки включає таблицю секретів для зберігання криптографічних ключів та токенів з можливістю контролю терміну дії та активності. Це дозволяє реалізувати ротацію ключів безпеки та забезпечити додатковий рівень захисту системи.

Після створення таблиць було налаштовано систему індексів та зовнішніх ключів для забезпечення цілісності даних та збільшення продуктивності БД. Первинні ключі автоматично індексуються автоматично, додаткові індекси створено для полів, які часто використовуються у запитах JOIN та WHERE. Це забезпечує швидкий доступ до даних навіть при значному зростанні кількості записів.

Усі зв’язки між таблицями встановлено з каскадним видаленням та оновленням даних. Це гарантує, що при видаленні користувача автоматично видаляються всі пов'язані з ним дані, включаючи повідомлення, сесії та участь у чатах. Така архітектура запобігає появі непов’язаних записів та підтримує консистентність бази даних.

Реалізована база даних повністю наслідує спроєктовану раніше архітектуру, запроваджуючи належний рівень продуктивності та безпеки та відповідаючи вимогам третього рівня нормалізації БД.

3.2.2 Реалізація серверної частини

Сервер у системі шифрованого кросплатформного месенджера виконує дії, пов’язані з базою даних, з авторизацією користувачів, а також посідає головну роль у передачі повідомлень між пристроями-клієнтами.

Сервер умовно поділений на модулі: криптографічний модуль; модуль зв’язку з базою даних; модуль утиліт та мережевий модуль. Кожен з них включає свій набір класів, що відповідають за певні дії.

Криптографічний модуль забезпечує безпеку даних у системі. Модуль включає класи SecureString та Crypt.

Клас SecureString призначений для безпечного зберігання конфіденційної інформації у пам'яті. Він блокує ділянки оперативної пам'яті від можливості їх збереження на диск, а при знищенні об'єкта автоматично очищує дані, впроваджуючи безпечне керування пам’яттю.

Простір імен Crypt містить функції для хешування паролів за допомогою алгоритму Yescrypt, створення та верифікації JWT токенів для авторизації користувачів, а також оперування ними.

Модуль зв'язку з базою даних відповідає за ефективне управління підключеннями до MariaDB, використовуючи класи ConnectionPool та ConnectionGuard, а також простір імен Database. Клас ConnectionPool реалізує пул підключень, який дозволяє уникнути витрат ресурсів сервера на створення нових з'єднань під час кожного запиту до бази даних. Пул підтримує максимальну кількість одночасних підключень та автоматично керує їх розподілом між різними потоками. Метод для створення нових підключень у пулі наведено у лістингу програмного коду 7.

Лістинг програмного коду 7 – Створення підключення у пулі підключень до бази даних

```
     std::shared_ptr<sql::Connection>
ConnectionPool::createConnection() {
     try {
     sql::SQLString url = sql::SQLString("jdbc:mariadb://") +
     sql::SQLString(m_host.data(), m_host.size()) +
     sql::SQLString("/") +
     sql::SQLString(m_database.data(), m_database.size());
     sql::Properties properties(
     {{"user", sql::SQLString(m_user.data(), m_user.size())},
     {"password", sql::SQLString(m_password.data(),
m_password.size())}, {"connectTimeout", "3"}, {"tcpKeepAlive",
"true"}, {"autoReconnect", "true"}});
```

```
    auto driver = sql::mariadb::get_driver_instance();
    if (!driver) throw std::runtime_error("Missing driver
instance");
    std::shared_ptr<sql::Connection> conn(driver->connect(url,
properties));
    conn->createStatement()->execute("SELECT 1");
    return conn;
    } catch (sql::SQLException &e) {
    Log_Warning("SQL Exception: " << e.what());
    throw std::runtime_error("Failed to create database
connection"); }}
```

Клас ConnectionGuard реалізує RAII-патерн для автоматичного отримання підключення з пулу та його повернення після завершення роботи, що гарантує коректне управління ресурсами навіть у випадку виникнення винятків [10]. Простір імен Database містить функції для роботи з різними сутностями системи та базою даних, включаючи управління секретами для підпису токенів, створення та перевірку користувацьких сесій, управління даними користувачів, чатами та повідомленнями.

Модуль утиліт забезпечує допоміжні функції для роботи системи за допомогою просторів імен Validator, Utils та Logger. Простір імен Validator містить функції для перевірки валідності користувацьких даних: перевірку паролів на відповідність вимогам безпеки та валідацію електронних адрес. Простір імен Utils надає функції для конвертації між різними форматами часу. Простір імен Logger реалізує систему логування з різними рівнями повідомлень, включаючи інформаційні повідомлення, попередження, помилки та інформацію налагодження, з автоматичним додаванням часу та інформації про місце виклику.

Основним модулем, навколо якого побудований сервер, є мережевий. Даний модуль забезпечує комунікацію між клієнтськими застосунками та базою даних за допомогою класів RestAPI та WebSocketHandler.

Клас RestAPI створює HTTP сервер з використанням фреймворку Pistache, який обробляє REST API запити для автентифікації користувачів,

управління профілями, чатами, відправлення та редагування повідомлень. При кожному запиті, окрім автентифікації та реєстрації, сервер очікує JWT токен доступу в заголовковій частині HTTP-пакета, після чого проводить авторизацію користувача і лише в разі успіху виконує відповідні операції.

Для визначення, яку саме операцію запросив клієнт, сервер використовує кінцеві вузли – URL за якими клієнтська програма може відправляти HTTP запити. Частина функції встановлення серверних кінцевих вузлів наведена у лістингу 8. Решта вузлів встановлюються за аналогічним принципом.

Лістинг програмного коду 8 – Встановлення кінцевих вузлів на сервері

```
desc.schemes(Rest::Scheme::Https)
      .basePath("/v1")
      .produces(MIME(Application, Json))
      .consumes(MIME(Application, Json));
desc.route(desc.get("/ready")).bind(&RestAPI::handleReady);
auto v1 = desc.path("/v1");
auto authPath = v1.path("/auth");
authPath.route(desc.post("/register"))
      .bind(&RestAPI::auth_register, this)
      .consumes(MIME(Application, Json));
authPath.route(desc.post("/login"))
      .bind(&RestAPI::auth_login, this)
      .consumes(MIME(Application, Json));
authPath.route(desc.post("/refresh"))
      .bind(&RestAPI::auth_refresh, this)
      .consumes(MIME(Application, Json));
```

До кінцевих вузлів прив’язуються методи класу RestAPI. У самих методах передаються однакові аргументи: об’єкт запиту та об’єкт для формування відповіді.

Кожна функція реалізована з використанням блоків try-catch для уникнення непередбачених ситуацій. У разі винятків сервер повертає на клієнтську програму HTTP-код і текст помилки. У додатку Б наведено код реалізації RestAPI метода auth_login() призначеного для автентифікації користувача.

Інший клас мережевого модуля – WebSocketHandler – відповідає за реалізацію WebSocket сервера, який забезпечує комунікацію в режимі реального часу між клієнтами для миттєвого доставлення повідомлень та сповіщень про статус активності користувачів у чатах без додаткових запитів від клієнтів.

Вебсокети використовують пул з'єднань і утримують їх доти, доки активна сесія користувача. Таким чином користувач може отримувати повідомлення навіть тоді, коли клієнтський застосунок працює у фоновому режимі. З'єднання між сервером та клієнтом створюються при автентифікації та реєстрації користувача, як показано у лістингу програмного коду 9, і діють протягом усього часу існування сесії.

Лістинг програмного коду 9 – Створення Websocket-з'єднань у пулі підключень

```
ws->setUserData(new UserData{user_id.value()});
{
    std::unique_lock lock(connectionsMutex_);
    userConnections_[user_id.value()].insert(ws);
    wsToUser_[ws] = user_id.value();
}
```

Описані компоненти формують серверну архітектуру, що забезпечує всі необхідні операції для функціонування шифрованого кросплатформного месенджера. Розбиття сервера на модулі дозволяє в майбутньому легко масштабувати його, додавати нові функції та змінювати старі. Така архітектура також спростила й розробку системи, спрощуючи орієнтацію у файлах програми при збільшенні їхнього обсягу.

3.2.3 Реалізація функціональної частини клієнта

Для забезпечення масштабованості системи та покращення орієнтації у структурі, бекенд месенджера побудовано за принципом сервісної архітектури.

Класи поділено на два модулі: core та models. Модуль core відповідає за основну роботу з сервером, обробку даних та шифрування. Models відповідає за зберігання даних і представлення їх для QML-інтерфейсу.

Core вміщає класи AuthManager, UserManager, ChatManager, MessageManager, CryptManager та RestAPIManager, Theme та структури User і Message для надання інформації про користувача і про повідомлення відповідно.

Усі класи модуля core використовують синглтон-патерн [11]. Синглтон – шаблон проєктування, який забезпечує наявність лише одного екземпляру класу, і надає можливість звертатися до цього екземпляра з будь-якої точки програми. Це дозволяє уникнути додаткових витрат оперативної пам’яті пристрою, які відбувалися б при створенні нових екземплярів класів.

Усі класи core та models використовують систему слотів та сигналів Qt для можливості реалізації асинхронності у системі. Сигнали це спеціальні методи класів, призначені для оголошення подій. Слоти – це функції або методи, які можуть бути підключені до сигналів, при отриманні сигналу всі підключені слоти виконуються автоматично.

RestAPIManager – основний клас, призначений для обміну даними з сервером через методи get() і post(). Метод get() використовується для здійснення запитів, які вимагають HTTP-метода GET. Реалізація методу наведена у лістингу 10.

Лістинг програмного коду 10 – Реалізація методу RestAPIManager::get()

```
void RestAPIManager::get(const QString &endpoint,
RestAPIManager::ResponseCallback callback) {
  if (!AuthManager::instance()->isAccessTokenValid()) {
    qWarning() << "Cannot make request: not authenticated";
    AuthManager::instance()->checkExistingTokens();
    return;}
  QNetworkRequest request = createRequest(endpoint);
  QNetworkReply *reply = m_manager->get(request);
  connect(reply, &QNetworkReply::finished, this, [this,
reply, callback]() { handleReply(reply, callback); });
  connect(reply, &QNetworkReply::sslErrors, this,
          &RestAPIManager::handleSslErrors);}
```

Даний метод здійснює відправлення запитів за переданою йому адресою кінцевого вузла і повертає результат за допомогою виклику функції handleReply(), пов’язаної з сигналом finished(). Аналогічно реалізований метод post(), проте у ньому як аргумент приймається ще тіло запиту.

Описані методи використовуються у класах AuthManager, ChatManager, UserManager, CryptManager та MessageManager. Обидва методи використовують допоміжну функцію createRequest(), наведену у лістингу 11, для формування HTTP-заголовків та handleReply() для обробки відповідей сервера. Функція handleReply() аналізує статус відповіді, обробляє можливі помилки та викликає передану функцію зворотного виклику з результатом операції.

Лістинг програмного коду 11 – Функція створення запиту

```
QNetworkRequest RestAPIManager::createRequest(const QString
&endpoint) {
       QUrl url(m_baseUrl + endpoint);
       QNetworkRequest request(url);
       for (auto it = m_headers.constBegin(); it !=
m_headers.constEnd(); ++it) {
         request.setRawHeader(it.key(), it.value());
       }
     request.setSslConfiguration(QSslConfiguration::defaultConfig
uration());
       return request;
}
```

Інші класи використовують RestAPIManager для створення усіх запитів до сервера. Це рішення значно спрощує процес розробки, оскільки розробникам не потрібно вручну налаштовувати кожен мережевий запит, а достатньо викликати відповідний метод з необхідними параметрами. Наприклад, у лістингу програмного коду 12 наведено функцію реєстрації користувача класу AuthManager через виклик метода post() за кінцевим вузлом «/v1/auth/register», з тілом запиту у форматі QVariant та функцією зворотного виклику handleSignUpResponse().

Лістинг програмного коду 12 – Функція реєстрації користувача

```
    void AuthManager::signUpUser(const QString &email, const
QString &username, const QString &name, const QString &password)
{
       RestAPIManager *api = RestAPIManager::instance();
       QJsonObject data{{"email", email},
                        {"username", username},
                        {"name", name},
                        {"password", password}};
    api->post("/v1/auth/register", QJsonDocument(data).toJson(),
                [this](int httpCode, const QVariant &response) {
                  handleSignUpResponse(httpCode, response);
    });
}
```

При автентифікації та реєстрації користувачу повертаються JWT-токени: токен доступу та токен оновлення. Функції зворотного виклику, при отриманні HTTP-коду 200 від сервера, викликають метод saveTokens(), наведений у додатку В. Метод використовує QKeychain для збереження токенів у захищеному середовищі системи. Бібліотека є кросплатформною і підтримується на Linux, Windows та MacOS.

При кожному запиті до сервера (окрім запитів автентифікації та авторизації) викликається метод checkExistingTokens() для перевірки наявності токенів та їхньої валідності. Код методу подано у додатку Г.

Після збереження токенів клас AuthManager відсилає сигнал про те, що користувач автентифікований. При отриманні сигналу відправляється запит на отримання даних автентифікованого користувача через метод UserManager::get_me(), код якого наведено у лістингу 13.

Лістинг програмного коду 13 – Метод отримання даних користувача

```
    void UserManager::get_me() {
    if (AuthManager::instance()->getAccessToken().isEmpty()) {
      qWarning() << "UserManager::get_me(): Access token is
empty";
       return;
    }
    RestAPIManager *api = RestAPIManager::instance();
    api->setAuthorizationHeader(AuthManager::instance()-
>getAccessToken());
```

```
    api->get("/v1/users/me", [this](int code, const QVariant
&response) {
    if (code == 200 && response.canConvert<QJsonDocument>()) {
    QJsonObject json = response.value<QJsonDocument>().object();
    m_currentUser.user_id = json["user_id"].toString();
    m_currentUser.username = json["username"].toString();
    QUrl avatarUrl(json["avatar_url"].toString());
    QNetworkReply *reply =
m_networkManager.get(QNetworkRequest(avatarUrl));
    connect(reply, &QNetworkReply::finished, [this, reply]()
{ this->handleAvatarDownload(reply); });
    } else {
      emit errorOccurred("Failed to load user profile");
      AuthManager::instance()->signOutUser();
    }
  });
}
```

Клас UserManager керує даними користувачів та їхніми профілями. Метод записує отримані дані в об'єкт структури User, що містить поля для відображуваних даних користувача: ідентифікатор, ім'я користувача, відображуване ім'я та аватар.

Для роботи з чатами реалізовано клас ChatManager, що відповідає за створення та видалення чатів. Клас напряму пов’язаний з моделлю ChatModel, яка відповідає за збереження та надання інформації про чати.

Клас CryptManager призначений для шифрування та дешифрування даних у застосунку з використанням гібридного підходу, що поєднує симетричний алгоритм AES-256-GCM та асиметричний алгоритм RSA.

Коли створюється екземпляр класу, CryptManager перевіряє наявність збережених ключів RSA у QKeychain через виклик методу loadPrivateKey(). Якщо приватний ключ відсутній, викликається метод generateKeyPair() для створення нової пари ключів. Реалізація функції генерації пари ключів RSA міститься у додатку Ґ. Згенерований приватний ключ зберігається у системному сховищі QKeychain, а публічний ключ відправляється на сервер для доступу іншим користувачам.

При створенні нового чату метод requestPublicKey() відправляє запит на сервер для отримання публічного ключа співрозмовника з сервера. Отримані ключі зберігаються у масиві m_userPublicKeys для швидкого доступу під час шифрування без повторних звернень до сервера.

Для шифрування даних застосовується метод encryptMessage(), що приймає вхідний текст повідомлення та ідентифікатор його отримувача.

Спочатку перевіряється наявність публічного ключа отримувача і генерується ключ AES, яким шифрується саме повідомлення з використанням режиму GCM, що є одним з найбільш ефективних режимів симетричних видів шифрування.

Отриманий AES-ключ шифрується публічним ключем отримувача, після чого зашифрований ключ, вектор ініціалізації, тег автентифікації та шифротекст об’єднуються у єдиний пакет для передачі в мережі. У лістингу програмного коду 14 наведено реалізацію функції encryptMessage().

Лістинг програмного коду 14 – Функція шифрування повідомлень

```
    QByteArray CryptManager::encryptMessage(const QString&
plainText, const QString& receiverId) {
        if (!m_userPublicKeys.contains(receiverId))
            return emit encryptionError("No public key
available"), QByteArray();
        QByteArray aesKey = generateAESKey();
        if (aesKey.isEmpty())
            return emit encryptionError("Failed to generate AES
key"), QByteArray();
        EncryptedPackage package =
encryptAES(plainText.toUtf8(), aesKey);
        if (package.cipherText.isEmpty())
            return emit encryptionError("AES encryption
failed"), QByteArray();
        if (RSA* pubKey =
createRSAFromPem(m_userPublicKeys.value(receiverId).toLatin1(),
true)) {
            package.encryptedKey = encryptRSA(pubKey, aesKey);
            RSA_free(pubKey);
        }
        return serializePackage(package);
    }
```

Для дешифрування реалізовано метод decryptMessage(), який використовує локально збережений приватний ключ для розшифровки отриманих даних. Метод працює у зворотному порядку відносно функції шифрування.

Для виконання базових операцій з повідомленнями використовується клас MessageManager, який також є синглтоном. Основні методи класу включають sendMessage() для відправлення нових повідомлень на сервер з тимчасовим ідентифікатором, deleteMessage() для видалення повідомлень за їх ідентифікатором та getMessages() для запиту історії повідомлень конкретного чату. При успішному відправленні повідомлення генерується сигнал messageSent() з остаточним ідентифікатором повідомлення від сервера.

Підхід з наявністю тимчасових ідентифікаторів використовується для створення черги повідомлень, щоб повідомлення додавалось до інтерфейсу користувача одразу, а його доставлення підтверджувалося вже по відповіді від сервера.

MessageManager працює як проміжний шар між QML-інтерфейсом та сервером, делегуючи детальну обробку, зберігання та представлення повідомлень екземплярам моделі повідомлень, які створюються для кожного чату окремо.

Окрім менеджерів модуль core включає клас Theme, призначений для надання тем. Теми реалізовані через JSON-файли та зберігаються поруч з виконуваним файлом, тож користувачі можуть змінювати встановлені, а також писати власні теми застосунку. Клас Theme отримує кольори з формату JSON і записує їх у невпорядкований асоціативний масив у форматі QColor за допомогою коду, поданого у лістингу 15.

Лістинг програмного коду 15 – Збереження JSON кольорів у асоціативний масив

```
void Theme::loadFromFile(const QString &path)
{
    QFile file(path);
    if (!file.open(QIODevice::ReadOnly)) {
        qWarning("Could not open theme file.");
        return;
    }
QJsonDocument doc = QJsonDocument::fromJson(file.readAll());
    file.close();
    if (!doc.isObject()) {
        qWarning("Incorrect format of JSON theme.");
        return;
    }
    QJsonObject json = doc.object();
    for (auto it = json.begin(); it != json.end(); ++it) {
        if (it.value().isString()) {
           colors[it.key()] = QColor(it.value().toString());
        }
    }
    emit themeChanged();
}
```

Модуль models включає моделі, призначені для відображення даних в інтерфейсі месенджера. Моделі отримують дані від сервера, формують з них масиви типу QVariant з використанням QSharedPointer для безпечного управління пам’яттю і надають сформовані об’єкти для використання у QML.

Розроблені моделі включають ChatModel – для надання інформації про чат; ChatListModel – для формування списку чатів користувача; MessageModel – для зберігання повідомлень конкретного чату.

Модель ChatListModel призначена для управління списком чатів користувача на панелі чатів. Модель містить два основні QVariant контейнери: m_chats для зберігання повного списку чатів користувача та m_searched_chats для тимчасового зберігання результатів пошуку. Модель забезпечує методи для завантаження чатів з серверної частини системи, пошуку чатів за назвою та очищення результатів пошуку.

ChatModel являє собою клас для представлення окремого чату в системі. Клас містить основні властивості чату: ідентифікатор, тип, назву, опис, URL зображення та дату створення. Клас імплементує getter-методи для доступу до всіх властивостей. ChatModel служить основою об'єктів чатів, які створюються, коли користувач обирає чат з переліку ChatListModel.

MessageModel є найбільш комплексною з моделей і призначена для управління повідомленнями конкретного чату. Для кожного чату створюється окремий екземпляр MessageModel. Модель використовує MessageManager для виконання запитів на отримання, відправлення та видалення повідомлень. Також модель включає два стани повідомлень: тимчасові повідомлення, що очікують відправлення, та підтверджені повідомлення від сервера.

Модель підтримує асинхронну обробку повідомлень через систему сигналів і слотів, що дозволяє відстежувати зміни статусу доставлення повідомлень та автоматично оновлювати інтерфейс користувача.

Особливістю реалізації є використання кешування повідомлень для різних чатів у хеші m_chatMessages, що оптимізує продуктивність при перемиканні між активними розмовами. Модель також включає механізм відстежування унікальних ідентифікаторів через QSet для запобігання дублювання при отриманні масиву нових повідомлень з сервера.

Реалізована функціональна частина клієнтського застосунку забезпечує надійну модульну архітектуру з чітким розділенням відповідальностей між компонентами. Використання шаблону синглтон для ключових класів гарантувало оптимальне використання системних ресурсів, а асинхронна обробка запитів через систему сигналів і слотів забезпечила плавний користувацький досвід. Впровадження гібридного шифрування та кешування даних дозволило підвищити як безпеку, так і продуктивність системи.

3.2.4 Реалізація користувацького інтерфейсу

Для реалізації інтерфейсу використовувалася декларативна мова QML, надана фреймворком Qt. QML підтримує і написання функцій, однак для забезпечення максимальної продуктивності, весь функціонал перенесено у бекенд-частину на C++. На стороні інтерфейсу використовуються лише мінімальні функції, що не вимагають здійснення складних операцій.

Слідуючи спроєктованому макету в розділі проєктування інтерфейсу, реалізовано сторінку автентифікації та реєстрації, зображену на рисунку 3.1.

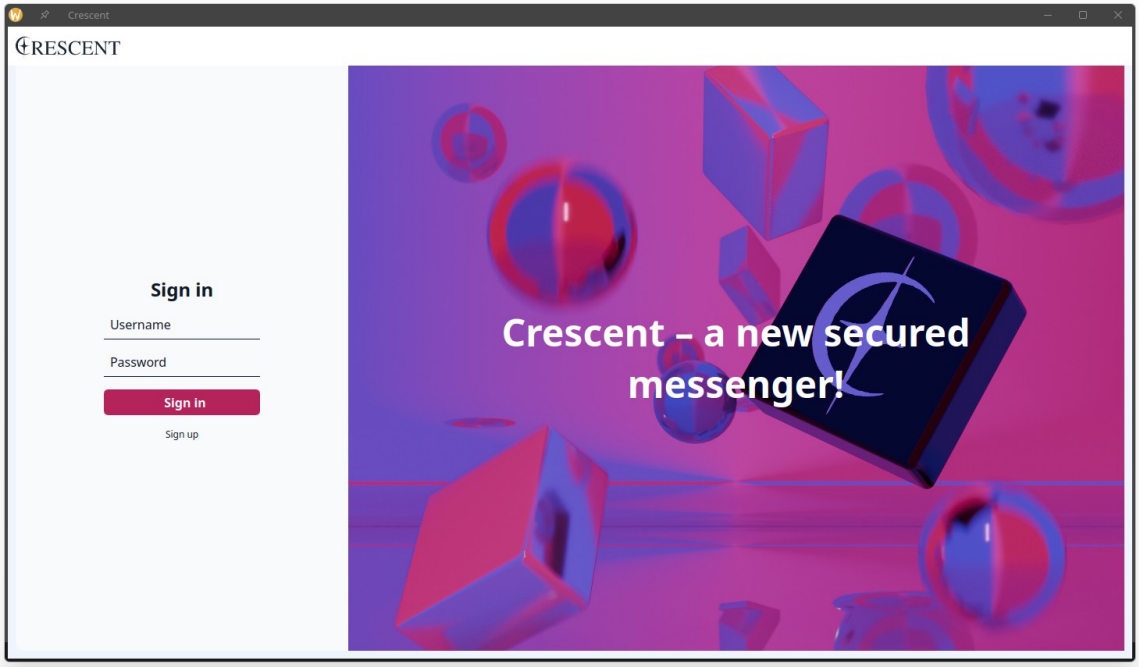

Рисунок 3.1 – Реалізована сторінка авторизації

У лівій частині розміщений макет колонки, що вміщає дві форми: форму автентифікації та форму реєстрації.

Форми змінюються за допомогою змінної isSignUp у батьківському елементі сторінки. При натисканні кнопки переходу до іншої форми значення isSignUp міняється на протилежне, завдяки якому змінюється прозорість та активність форм. У формах використовується власний стиль полів вводу, тож на різних пристроях забезпечено їх однаковий стиль.

При натисканні кнопки входу викликається метод signInUser() з даними форми. Відповідно, при реєстрації, викликається signUpUser().

Після входу користувач потрапляє на головну сторінку месенджера (рис. 3.2).

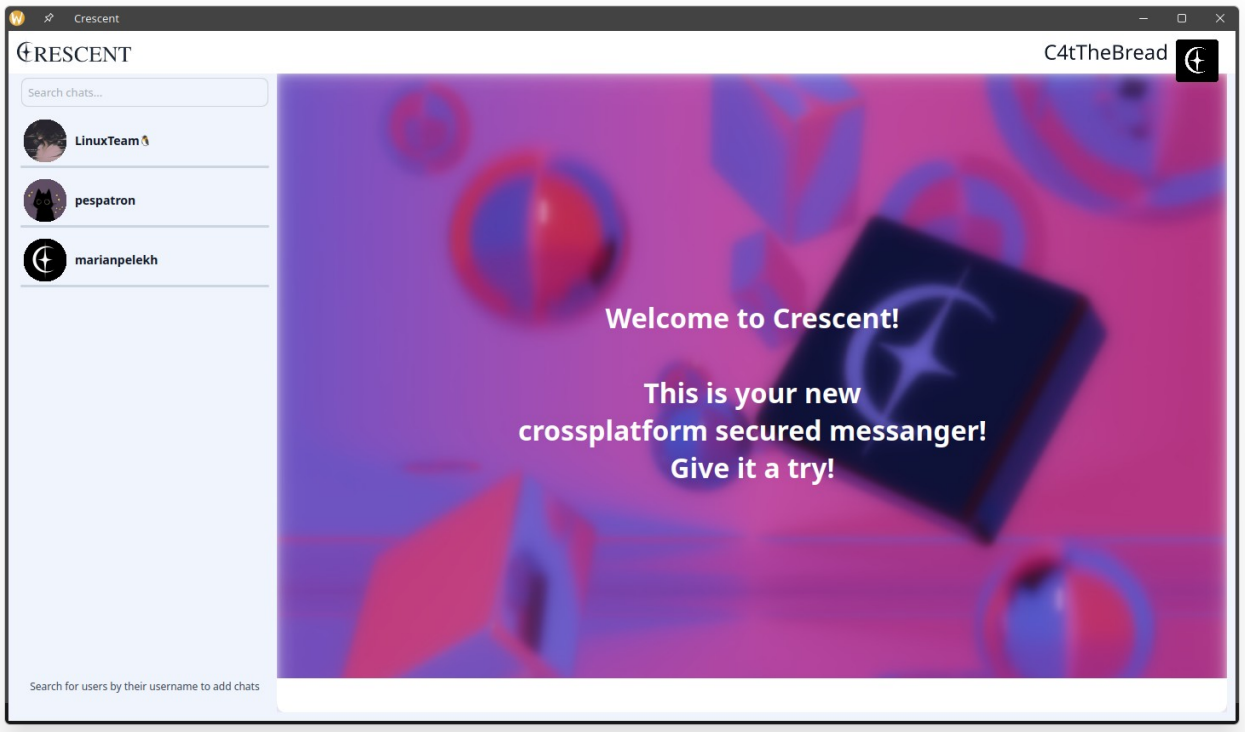

Рисунок 3.2 – Головна сторінка месенджера

Ліворуч розташований список чатів користувача та поле для пошуку нових чатів. Список надається моделлю ChatListModel, що використовує структуру Chat для формування масиву.

Відображення реалізовано через компонент ListView. Компонент автоматично обробляє прокрутку списку, позиціювання елементів і відстежує зміни в моделі через з'єднання Connections, що дозволяє оновлювати інтерфейс у режимі реального часу, при додаванні або видаленні чатів.

Для пошуку чатів використовується TextField з викликом функції chatListModel.searchChats() із затримкою 400 мілісекунд. При пошуку список чатів користувача замінюється чатами, які відповідають пошуковому запиту, як зображено на рисунку 3.3.

Рисунок 3.3 – Пошук чатів

При натисканні на елемент викликається сигнал chatSelected, який передає ідентифікатор та назву обраної розмови для подальшого переходу на сторінку чату.

За завантаження чатів відповідає модуль ChatPage. Сторінка одразу звертається до моделі повідомлень, яка автоматично підвантажує історію відповідного чату.

Модель MessageModel використовує систему ролей для ефективного відображення даних повідомлень у QML інтерфейсі, забезпечуючи оптимальну продуктивність при роботі з великими обсягами текстових даних.

Інтерфейс чату складається з трьох основних компонентів: верхньої панелі з інформацією про розмову, центральної області для відображення повідомлень та нижньої панелі для введення тексту. Загальний вигляд інтерфейсу представлено на рисунку 3.4.

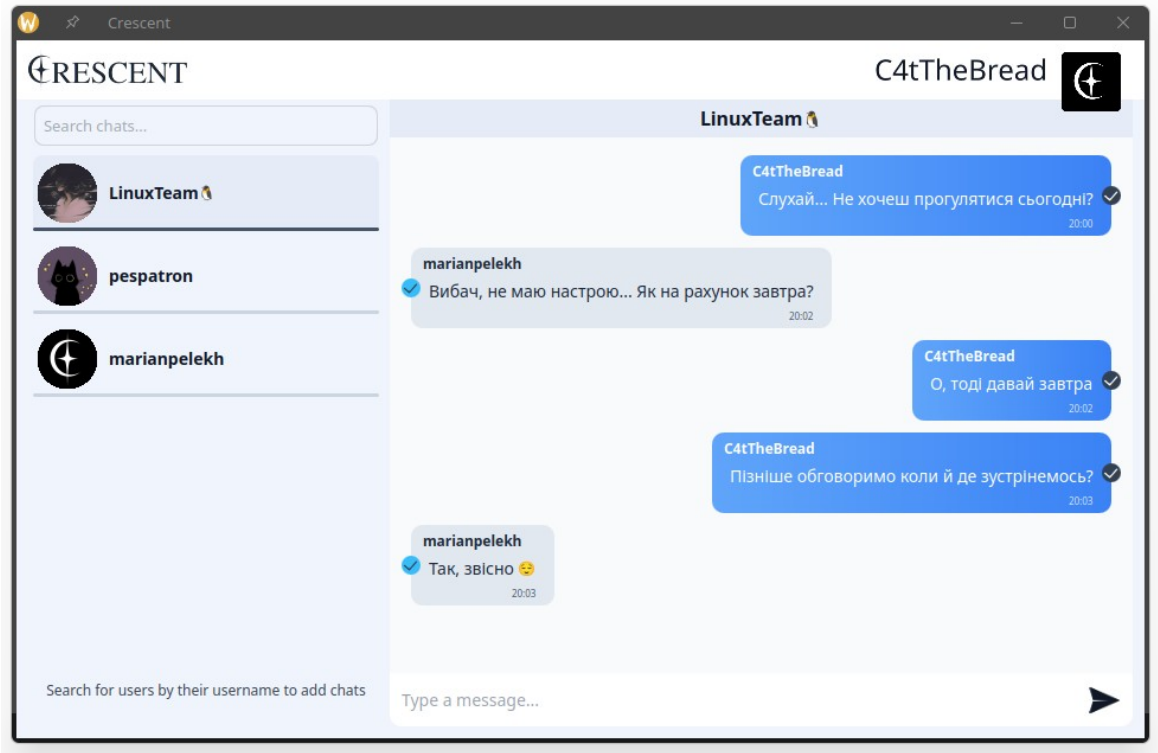

Рисунок 3.4 – Сторінка чату

Вгорі сторінки модуль створює рядок з назвою чату або іменем користувача. Для написання повідомлень ChatPage створює унизу сторінки текстове поле та кнопку відправлення (рис. 3.5).

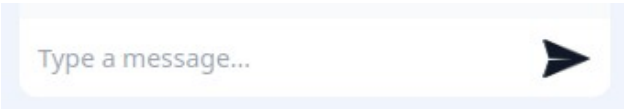

Рисунок 3.5 – Поле введення повідомлення

Поле підтримує розширення у висоту залежно від розміру введеного тексту та вертикальну прокрутку при переповненні.

При відправленні повідомлення його об’єкт додається у MessageModel з тимчасовим ідентифікатором поки сервер не поверне постійний. Доки у повідомлення тимчасовий ідентифікатор, воно вважається не надісланим і відображає відповідну іконку.

За відображення повідомлень відповідає розміщений в центрі сторінки компонент ListView з делегатом типу Message, як наведено у лістингу програмного коду 16.

Лістинг програмного коду 16 – Делегат повідомлень у чаті

```
delegate: Message {
sender: model.sender
status: model.status
created_at: model.created_at ? model.created_at :
Qt.formatTime(new Date(), "H:mm")
text: model.text
}
```

У делегаті використовується власний тип Message. Тип є рядковим динамічним макетом, усередині якого міститься власне контейнер повідомлення. За допомогою булевої змінної isSender визначається де розташувати повідомлення: ліворуч чи праворуч.

Для покращення користувацького досвіду реалізовано додаткові інтерфейсні елементи. Для керування повідомленнями передбачено контекстне меню, яке з'являється при натисканні на повідомлення правою клавішею миші. Меню дозволяє копіювати повідомлення, а також видаляти власні повідомлення з чату для себе та інших співрозмовників (рис. 3.6).

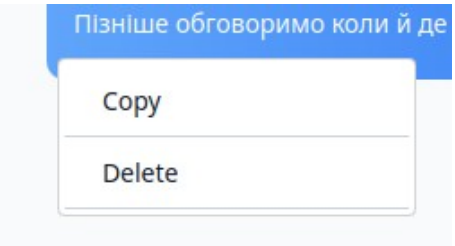

Рисунок 3.6 – Контекстне меню повідомлень

У правому верхньому куті головної сторінки розміщено меню користувача з опціями вибору теми та виходу. Система надає п'ять тематичних оформлень за замовчуванням: Dark, Light, Ice, Paper та Acid, які застосовуються до всього інтерфейсу та зберігаються в налаштуваннях користувача (рис. 3.7).

Рисунок 3.7 – Контекстне меню програми

Усі іконки в системі були створені власноруч з використанням програмного засобу Inkscape. Створені іконки у форматі SVG були конвертовані у шрифт формату TTF, що забезпечило можливість зміни їх кольору як текстових символів.

Реалізований інтерфейс забезпечує зручний користувацький досвід впроваджуючи мінімалістичний дизайн та інтуїтивне розташування елементів. Адаптивність компонентів QML гарантує коректне відображення на різних співвідношеннях сторін екрана, а оптимізація взаємодії між інтерфейсом і функціональною частиною забезпечує плавну роботу навіть при великому навантаженні.

3.3 Тестування системи

Тестування роботи системи важливе на всіх етапах розробки. Протягом реалізації системи проводилось модульне, інтеграційне та системне тестування системи шифрованого кросплатформного месенджера.

Враховуючи, що для виконання будь-яких операцій користувач повинен авторизуватися, спочатку було проведено тестування автентифікації та реєстрації. Очікуваним результатом успішної автентифікації є повернення токенів доступу та оновлення, додавання нової сесії, а також перенаправлення користувача на головну сторінку. Очікуваний результат реєстрації є аналогічним, однак додатково повинен з’явитись запис у таблиці users.

Заповнення форми входу наявними даними користувача призвело до перенаправлення на домашню сторінку. Логування в режимі налагодження дозволило побачити, що токени у форматі JSON були отримані успішно (рис. 3.8). При введенні неправильного логіну чи пароля вхід не відбувається.

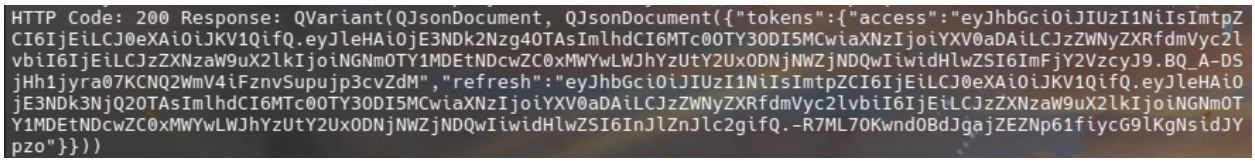

Рисунок 3.8 – Логування отримання токенів доступу та оновлення

Аналогічно було протестовано функціональність реєстрації нових користувачів. Заповнення форми реєстрації даними та послідовне надсилання форми дало очікуваний результат – отримані токени, користувача перенаправлено на головну сторінку, у базі даних створено записи.

Для перевірки винятків, випробувано роботу автентифікації/реєстрації при введенні некоректних даних.

При введенні електронної адреси неправильного формату користувача одразу попереджають про це, як зображено на рисунку 3.9.

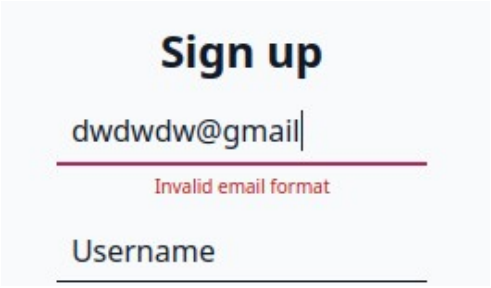

Рисунок 3.9 – Попередження про некоректний формат електронної адреси

Поля встановлення псевдоніма та відображуваного імені користувача мають обмеження на довжину тексту розміром 16 символів.

Поля введення пароля мають обмеження розміром 32 символи. Якщо пароль і його підтвердження відрізняються, користувач отримує спливне повідомлення про помилку (рис. 3.10).

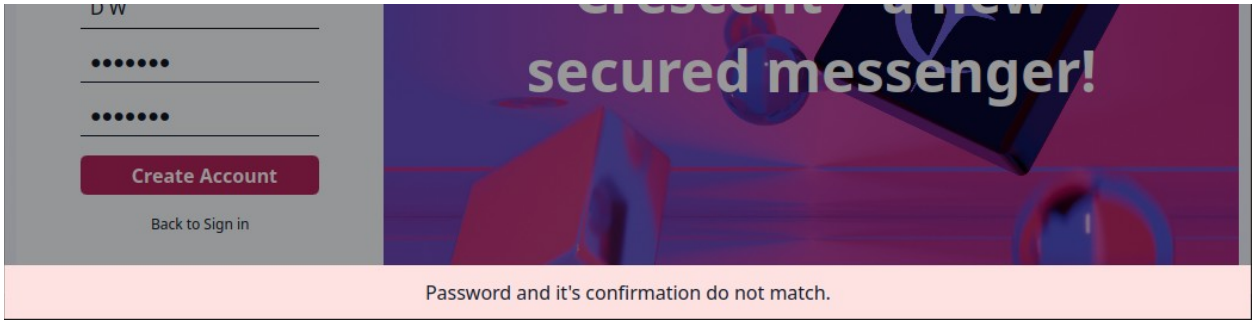

Рисунок 3.10 – Повідомлення про розбіжність паролів

Наступним об’єктом тестування стала панель чатів. Одразу після отримання токенів завантажилася модель списку чатів. Жоден чат не був упущений. При кількості чатів більшій, ніж вміщається в межах вікна, список стає прокручуваним (рис. 3.11).

Рисунок 3.11 – Переповнений список чатів

Пошук чатів та користувачів здійснюється успішно, незалежно від символьного регістру, що зображено на рисунку 3.12.

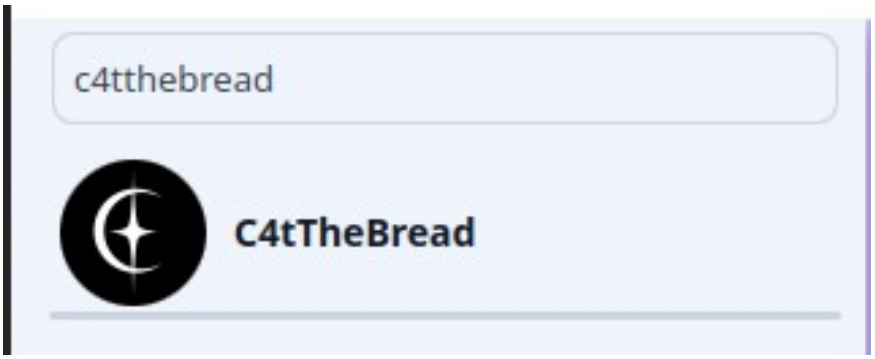

Рисунок 3.12 – Пошук чатів/користувачів

При відкритті чату зі знайденим користувачем, відкривається порожня сторінка чату. Назва чату відображається коректно.

Для створення чату з обраним користувачем необхідно надіслати повідомлення. Після успішного надсилання повідомлення у базі даних створюється відповідний запис у таблиці chats та два записи у таблиці зв’язку user_chat. Для видалення розмов використовується контекстне меню на панелі чатів. Тестування довело, що їхнє видалення здійснюється без будь-яких проблем.

Відправлення повідомлень здійснюється без дефектів. При переповненні поля вводу, область розширюється, даючи користувачеві можливість зручніше переглядати написаний текст (рис. 3.13).

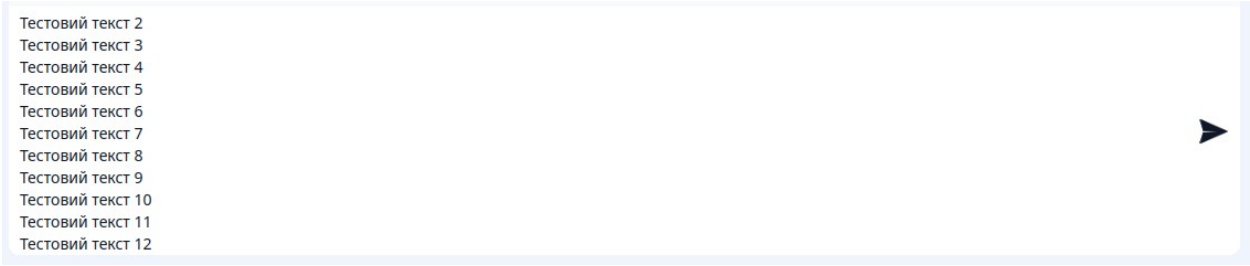

Рисунок 3.13 – Тестування розширення поля вводу повідомлення

Для проведення тестування одержання повідомлень було відкрито поруч два вікна з сесіями у різних облікових записах (рис. 3.14). Тестування підтвердило швидкодію системи як у локальній мережі, так і в глобальній – повідомлення приходили з мінімальною затримкою.

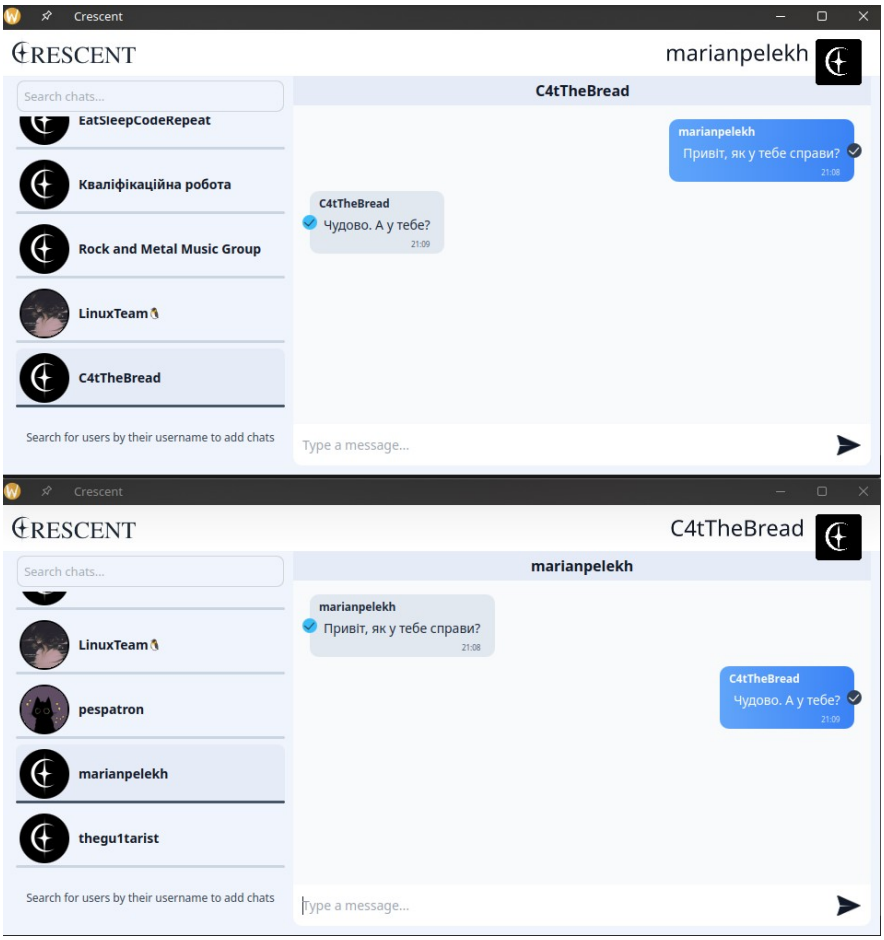

Рисунок 3.14 – Тестування затримки надходження повідомлень

Довжина тексту повідомлення обмежена 4096 символами, ні введення тексту вручну, ні вставлення тексту за допомогою сполучення клавіш Ctrl + V не дозволяє перетнути обмеження.

Контекстне меню повідомлень успішно відкривається правою клавішею миші. Меню дозволяє копіювати та видаляти повідомлення. При видаленні повідомлення видаляються з інтерфейсу всіх учасників чату та бази даних.

Контекстне меню месенджера також працює успішно, виконуючи заплановані дії. При виборі теми здійснюється заплановане перезавантаження

графічного інтерфейсу месенджера, після чого обрана тема вже є застосованою до компонентів застосунку (рис. 3.15).

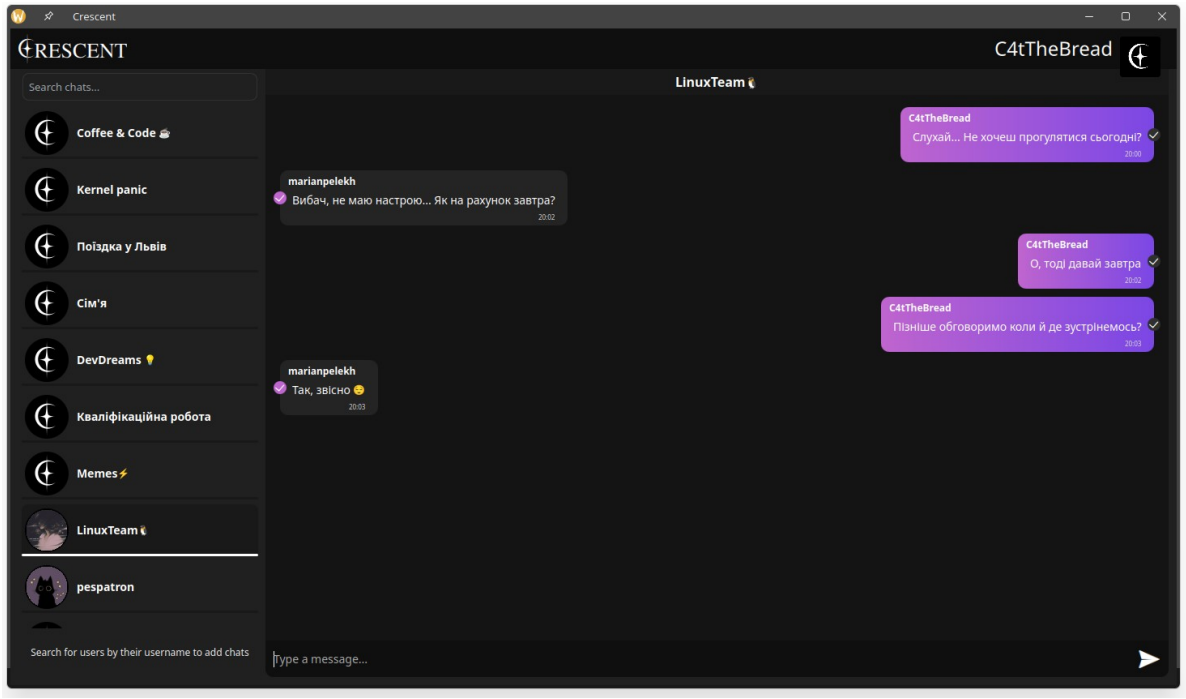

Рисунок 3.15 – Застосована темна тема до застосунку

Було протестовано мережеву взаємодію між клієнтським застосунком та сервером за допомогою програмного забезпечення Wireshark. На рисунку 3.16 представлено перехоплений мережевий пакет. Даний приклад підтверджує наявність шифрування мережевого трафіку за допомогою протоколу TLSv1.3, що використовується у HTTPS.

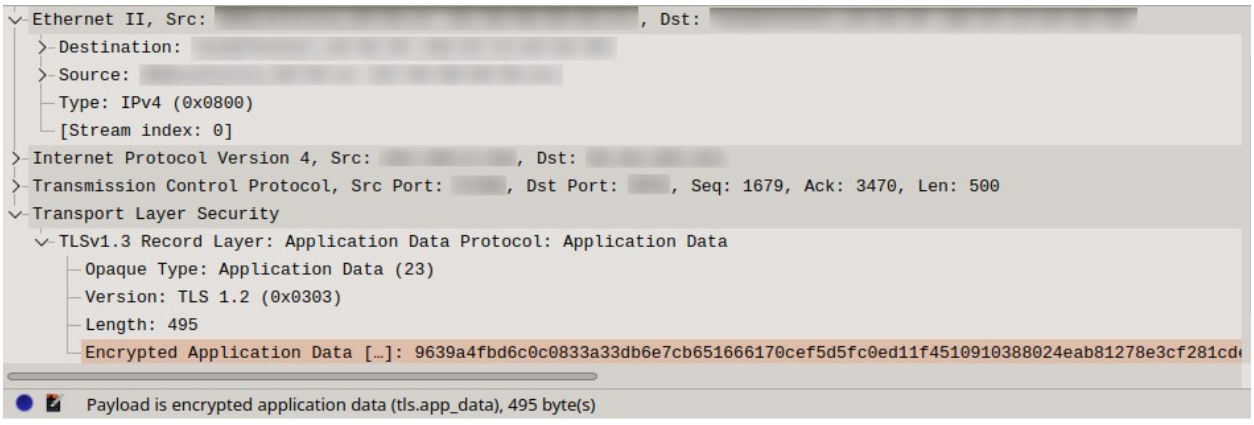

Рисунок 3.16 – Перехоплений пакет за допомогою Wireshark

Роботу E2E шифрування з використанням алгоритмів AES та RSA перевірено за допомогою графічного інтерфейсу керування СУБД PhpMyAdmin. На рисунку 3.17 зображено формат зберігання даних, зашифрованих за допомогою AES-256-GCM.

id ▲ 1	sender_id	text	media
9d7373b3-46e5-11f0-bac5-ce183c5fc440	595efd6f-f628-11ef-9e90-5ac937da9cf7	ur7K0wrAUnPLMmRjYAGqR/EklhkNN815yNpEmjcK5E+8hN7PW1...	
e10e160f-46e5-11f0-bac5-ce183c5fc440	62c243ef-f740-11ef-9c49-664f51ce00a6	yDKgkon2UGHTnXl0CJqERbuXE3Vb0okWPR4dWIGuerE+RQ8eIl...	
e79e7ef9-46e5-11f0-bac5-ce183c5fc440	595efd6f-f628-11ef-9e90-5ac937da9cf7	4dH75w3f3J2KwZdn3p9ctuqZUT8Yjp8ocCITz+PkjZdVzqSa1b...	
02f978af-46e6-11f0-bac5-ce183c5fc440	595efd6f-f628-11ef-9e90-5ac937da9cf7	4+aZQ8sbsJ9qnezF6R7d+EYNfnuZw7K7QdlaXmP5i6be1gahCw...	
107e432e-46e6-11f0-bac5-ce183c5fc440	62c243ef-f740-11ef-9c49-664f51ce00a6	NEdC2s4GM0u6TMx5pHp6h9bRv/coMVsDw7t23Dxc7W+MRMc3sg...	
c05759b1-46f7-11f0-bac5-ce183c5fc440	595efd6f-f628-11ef-9e90-5ac937da9cf7	Cq/LuB7ENfWjlQW4ofQVMEnPm7pbaaeG4bkgMZIbyjq9wSkVl0...	
fd62b6a5-46f7-11f0-bac5-ce183c5fc440	62c243ef-f740-11ef-9c49-664f51ce00a6	l+pOSVdPSLYWoE/m90BRTF9bIsvbszfF2lSvancYqhSmn9hjs+...	

Рисунок 3.17 – Таблиця зашифрованих повідомлень

Завершальним етапом тестування стала перевірка роботи застосунку на різних операційних системах. Окрім тестування на Arch Linux з графічною оболонкою KDE Plasma 6, протестовано кросплатформність застосунку на операційній системі Windows 11 24H2 (рис. 3.18). Окрім незначних дефектів графічного інтерфейсу, змін у роботі не знайдено.

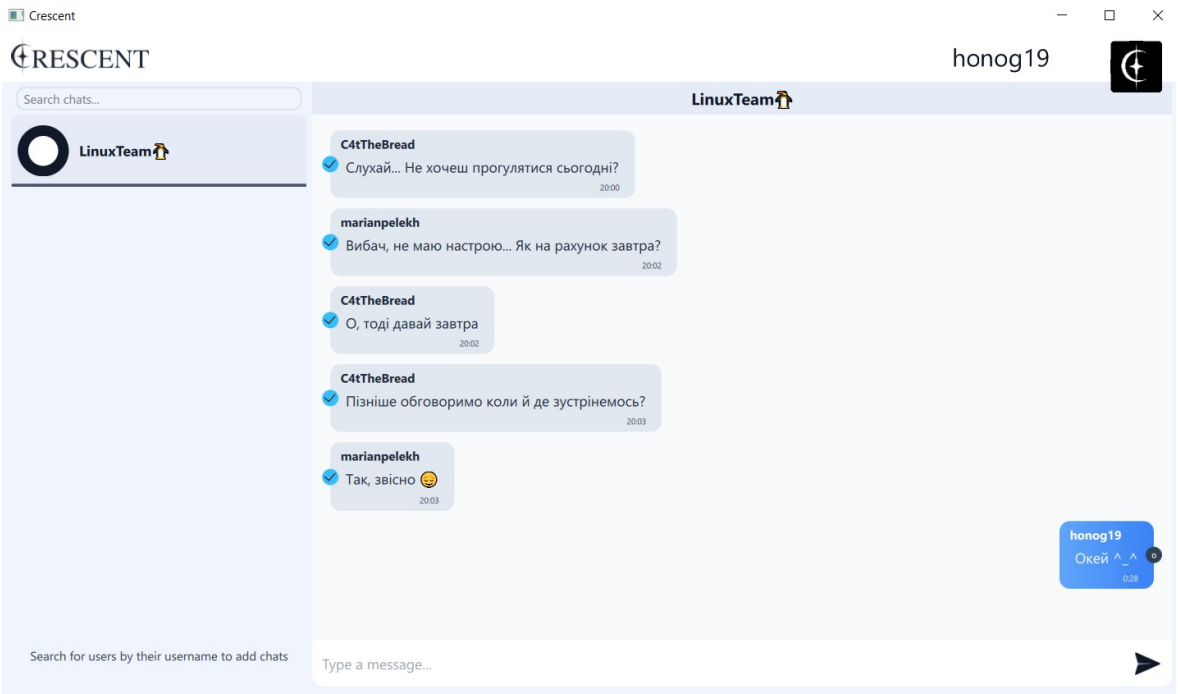

Рисунок 3.18 – Інтерфейс застосунку на Windows 11

Основними недоліками інтерфейсу на ОС Windows є відображення аватарів чатів та контекстних меню з некоректними відступами.

Проведене модульне та інтеграційне тестування підтвердило стабільну роботу всіх ключових функцій системи шифрованого кросплатформного месенджера. Успішно протестовано автентифікацію та реєстрацію, обмін повідомленнями, оперування чатами, безпеку передачі даних через TLS-шифрування та кросплатформність застосунку.

У результаті реалізації системи кросплатформного шифрованого месенджера та проведення комплексного тестування її функціональності отримано працездатний програмний продукт. Обрані засоби реалізації забезпечили оптимальне поєднання продуктивності, безпеки та підтримки різними платформами.

Реалізація бази даних у контейнеризованому середовищі продемонструвала зручність та ефективність такого підходу розгортання системи. Структура бази даних забезпечила необхідний рівень безпеки та продуктивності при роботі з даними.

Серверна частина успішно реалізує API для основних операцій та вебсокет-з'єднання для комунікації в режимі реального часу.

Функціональна частина клієнтського застосунку реалізована на основі сервісної архітектури, яка забезпечує чітке розділення обов'язків між компонентами.

Користувацький графічний інтерфейс використовує мінімалістичний дизайн з підтримкою користувацьких тематичних оформлень. Адаптивність компонентів забезпечує коректне відображення на різних співвідношеннях сторін екрана.

Реалізована система шифрованого кросплатформного месенджера повністю відповідає поставленим вимогам щодо безпеки, продуктивності та кросплатформності, забезпечуючи надійну основу для подальшого розвитку та масштабування функціональності месенджера.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку збуту продукту

Світовий ринок месенджерів залишається висококонкурентним, проте чітко сегментованим за географічними та функціональними ознаками. Глобальні рішення, такі як оглянуті раніше Telegram, Viber та Discord, домінують у Європі, Північній Америці та частині Азії, тоді як локальні рішення, зокрема WeChat і LINE, контролюють значні частки національних ринків Китаю та Японії відповідно.

Telegram, запущений у 2013 році, станом на березень 2025 року нараховує 1 млрд місячно активних користувачів (MAU), з яких 450 млн використовують платформу щодня. Середній час, який користувач проводить у додатку, становить понад 4 години на місяць, а щоденний приріст аудиторії сягає 2.5 млн нових реєстрацій. Не зважаючи на відсутність публічної інформації про точний річний дохід, експерти оцінюють його в межах 2–2.5 млрд доларів США завдяки монетизації через платні підписки та внутрішню рекламу.

Viber, заснований у 2010 році, демонструє менший ріст, ніж Telegram. У 2025 році його місячна аудиторія становить 250 млн користувачів, причому середній сеанс використання обмежується 8 хвилинами. Проте платформа залишається прибутковою: у 2024 році вона принесла материнській компанії Rakuten близько 3.2 млрд доларів США, частково завдяки інтеграції з екосистемою Rakuten в плані електронної комерції.

Discord, що з’явився на ринку у 2015 році, у 2025 році охоплює 200 млн MAU, а загальна кількість зареєстрованих акаунтів перевищує 560 млн. Його річний дохід у 2024 році склав 575 млн доларів США, з яких 85% припадає на платні підписки та продажі ігрових бібліотек. Платформа зберігає лідерство серед геймерів і технічних спільнот, але стикається з конкуренцією з боку Telegram у сегменті групових чатів.

У Китаї беззаперечним лідером є WeChat, запущений у 2011 році, який у 2025 році нараховує 1.385 млрд MAU, причому 78% аудиторії зосереджені в межах країни. Ця платформа інтегрує не лише месенджер, а й мобільні платежі, соціальні мережі та державні сервіси.

Натомість месенджер LINE, заснований у 2011 році, домінує в Японії, Таїланді, Тайвані та Індонезії, де його аудиторія становить 97 млн MAU. Доходи платформи формуються через рекламу, продажі наліпок та партнерства з брендами, що приносить понад 1 млрд доларів США щорічно.

Реалізований месенджер може використовувати поетапну стратегію виходу на ринок, спочатку зосередившись на локальному сегменті. Досвід таких платформ, як LINE та WeChat, довів, що успіх на національному рівні створює міцну основу для подальшого масштабування.

Після закріплення на локальному рівні наступним кроком може стати вихід на ринки Східної та Центральної Європи, де попит на альтернативи глобальним месенджерам зростає через підвищену увагу до конфіденційності. На завершальному етапі, маючи стабільну базу в Європі, месенджер може вийти на міжнародний рівень.

4.2 Розрахункова частина

4.2.1 Розрахунок витрат на проєктування

Вирахування витрат на розробку – важливий етап роботи з проєктом. Цей етап встановлює основу для подальшого розрахування необхідного доходу для покриття вартості розробки.

Витрати на проєктування включають суму витрат на заробітну плату працівникам, на внески до фондів, на електроенергію, на купівлю/оренду програмного забезпечення та технічних засобів та інші витрати:

$$C_{розр} = C_{зп} + C_{вн} + C_{ел} + C_{пр} + C_{інші}$$

$C_{зп}$ розраховується як добуток кількості відпрацьованих працівником годин та його погодинної оплати праці. Успішна розробка проєкту включає

наявність щонайменше двох розробників, UI/UX-дизайнера, системного адміністратора, менеджера проєкту, QA-інженера та SMM-спеціаліста. Погодинна заробітна плата і загальний добуток за час розробки представлено у таблиці 4.1.

Таблиця 4.1 – Розрахунок зарплати працівників

Працівник	Кількість робочих годин	Погодинна заробітна плата (грн)	Сумарна заробітна плата (грн)
UI/UX-дизайнер	160	218,75	35000
Frontend-розробник	320	281,25	90000
Backend-розробник	320	390,625	125000
Системний адміністратор	480	197,9	94992
QA-інженер	240	187,5	45000
SMM-спеціаліст	120	156,25	18750
Всього			408742

Підсумовуючи сумарну заробітну плату працівників за проєкт, витрати на заробітну плату дорівнюють:

$$С_{зп} = 408742 \text{ грн}$$

Наступним кроком є здійснення розрахунку обов’язкових внесків до державних фондів, що проводиться відповідно до чинного законодавства України. До таких внесків належать податок на доходи фізичних осіб, військовий збір і єдиний соціальний внесок (ЄСВ). Станом на перший квартал 2025 року відсотки цих податків становлять:

- податок на дохід фізичних осіб – 18%;
- військовий збір – 5% від мінімальної заробітної плати (400 грн);
- ЄСВ – 22% від заробітної плати.

Результати обчислень відрахувань на податки за актуальними відсотковими значеннями податків наведено у таблиці 4.2.

Таблиця 4.2 – Обчислення відрахувань на податки

Працівник	Податок на дохід ФОП (грн)	Військовий збір (грн)	ЄСВ (грн)	Сума відрахувань (грн)
UI/UX-дизайнер	4914	400	7700	13014
Frontend-розробник	12636	400	19800	32836
Backend-розробник	17550	400	27500	45450
Системний адміністратор	13338	400	20900	34638
QA-інженер	6318	400	9900	16618
SMM-спеціаліст	2632.5	400	4125	4525
Всього			89925	147081

Сумарна вартість витрат на податки та внески до фондів $С_{вн}$ дорівнює 147081 грн включно з коштами, виділеними із заробітної плати працівників.

Важливою категорією витрат є витрати на електроенергію. Це включає плату енергоспоживання робочих місць працівників, освітлення, кондиціонера тощо.

Для розрахунку витрат було визначено перелік приблизних витрат на електроспоживання під час розробки. До уваги бралися комп’ютери усіх працівників, освітлення приміщення, регулювання температури та забезпечення зв’язку. Додатково вирахувано приблизні інші витрати електроенергії (табл. 4.3).

Таблиця 4.3 – Розрахунок енергоспоживання робочим середовищем

Пристрій	Кількість	Потужність (Вт)	Час роботи/день (год)	Днів/міс	Споживання/міс (кВт·год)
Ноутбук	5	20	8	22	17,6

Продовження таблиці 4.3

Пристрій	Кількість	Потужність (Вт)	Час роботи/день (год)	Днів/міс	Споживання/міс (кВт·год)
Освітлення офісу	1	200	8	22	35,2
Обігрівач/кондиціонер	1	1500	4	22	132
Wi-Fi роутер	1	10	24	30	7,2
Інші пристрої	1	240	3	22	15,84
Всього					207,84

Середні місячні витрати на електроенергію, беручи для обчислень тариф розміром 4.32 грн, дорівнюють:

$$C_{ел/міс} = 207,84 \times 4,32 = 897,87 \text{ грн/міс.}$$

Враховуючи тривалість розробки терміном 5 місяців сумарні витрати на електроенергію під час розробки складають 4489,35 грн, що було обчислено за формулою:

$$C_{ел} = C_{ел/міс} \times 5 = 4489,35 \text{ грн.}$$

Важливим аспектом при розробці системи є вартість програмного забезпечення та технічних засобів. Оскільки проєкт реалізований на фреймворку Qt з використанням відкритого вихідного коду, ліцензійні витрати зведені до мінімуму. Ліцензія GPL (GNU General Public License) дозволяє безплатне розповсюдження застосунків з обов’язковим наданням доступу до вихідного коду, що повністю відповідає ідеї проєкту і виключає необхідність придбання комерційних ліцензій Qt.

Як IDE обрано безплатний інструмент, такий як Neovide, що не потребує витрат на платні підписки. Інші використані бібліотеки та фреймворки також відповідають критеріям відкритого ПЗ, що дозволяє уникнути порушень ліцензійних умов і знизити загальні витрати.

Обчислено витрати на купівлю/оренду технічних засобів, а також оренду робочого приміщення.

Розробка проєкту передбачає, що кожен працівник має власний персональний комп'ютер та комп'ютерну периферію або ноутбук.

Оренда офісного приміщення для розробки варта близько 7000 грн/місяць.

Додатково планується використання хостингу в короткостроковій перспективі. Для старту обрано сервіс HOSTiQ, розташований в Україні. Щомісячна плата за розміщення серверної частини проєкту на даному хостингу дорівнює 329.62 грн/місяць, що у термін перших 12 місяців існування проєкту складе 3955,44 грн. У довгостроковій перспективі планується купівля і використання власного сервера для роботи системи.

У результаті, за час розробки проєкту, витрати на купівлю/оренду програмного забезпечення та технічних засобів становлять:

$$C_{пр} = 7000 \times 5 + 329.62 \times 5 = 36648,1 \text{ грн}$$

Додатково були розраховані інші витрати. Обчислені витрати і їх сума за час розробки наведені у таблиці 4.4.

Таблиця 4.4 – Інші витрати

Тип витрати	Сума (грн)
Навчальні курси	700
Непередбачені витрати	3560
Транспорт	2640
Харчування	5500

У сумі інші витрати під час розробки проєкту складають:

$$C_{інші} = 700 + 3560 + 2640 + 5500 = 12400 \text{ грн}$$

Таким чином, загальні витрати на розробку проєкту можна розрахувати за формулою:

$$C_{розр} = 408742 + 89925 + 4489,35 + 36648,1 + 12400 = 552204,45$$

У результаті проведених обчислень отримано суму розміром 552204,45 гривень, що є сумою усіх витрат на розробку даного месенджера.

Для подальшої підтримки існування проєкту необхідні витрати складають суму оренди офісного приміщення, витрати на електроенергію та витрати на зарплату працівникам, необхідним для обслуговування системи.

Враховуючи менший обсяг роботи після реалізації месенджера, кількість робочих годин працівників скоротиться, тож витрати на зарплату працівникам стануть близько 60-70 тисяч грн в місяць, залежно від кількості необхідної роботи. Витрати на електроенергію зменшуються пропорційно, а оренду приміщення залишаються сталими.

Підсумовуючи значення, для підтримки проєкту після виведення його в експлуатацію необхідно близько 80 тисяч гривень в місяць.

4.2.2 Економічний ефект

На початковому етапі проєкт не отримує прямий фінансовий прибуток через відмову від платних підписок та реклами. Це дозволяє залучити максимальну аудиторію, зокрема IT-фахівців, які можуть вносити власні покращення до месенджера завдяки відкритому вихідному коду. Однак економічний ефект проявляється через запобігання витратам, пов’язаним із витоками даних, що є критичним для користувачів і бізнесів.

Месенджер знижує ризики витоків даних завдяки комбінації наскрізного шифрування, яке гарантує, що дані користувачів не можуть бути прочитані навіть у разі перехоплення, та відкритого коду, що дозволяє стороннім розробникам перевіряти систему на вразливості, зменшуючи ймовірність виникнення помилок чи витоків даних.

Дослідження показують, що застосування E2EE та відкритого коду знижує кількість успішних атак на 70-80%. Для платформи з 100000 зареєстрованих користувачів це означає уникнення 1-2 масштабних витоків даних на рік, які характерні для аналогів без подібного рівня захисту.

Середня світова вартість одного витоку даних для компанії становить 4,45 млн доларів США (або 186,82 млн грн), враховуючи витрати на розслідування, штрафи, втрату довіри клієнтів та відновлення систем. Якщо середня вартість втрати одного запису даних користувача складає 6890 грн, то для месенджера з аудиторією 100000 користувачів потенційні збитки від інциденту можна оцінити як:

$$100000 \times 6890 = 689 \text{ млн грн.}$$

При зменшенні ймовірності витоків даних на 80%, річна економія складає:

$$689 \text{ млн} \times 0{,}8 = 551{,}2 \text{ млн грн.}$$

Додатково враховуються штрафи за порушення GDPR (до 874,66 млн грн за інцидент), яких також вдається уникнути.

Підсумовуючи збитки, з аудиторією 100 тисяч користувачів месенджер дозволяє заощадити:

$$689 \times 0{,}8 + 874{,}66 = 1{,}426 \text{ млрд грн}$$

Після досягнення стабільної аудиторії планується впровадити платну підписку (₴250-350/місяць) з додатковими функціями: збільшений обсяг файлів, відсутність реклами, ексклюзивні наліпки. При оплаті підписки 5% аудиторії річний дохід може досягати:

$$100000 \times 0.05 \times 300 \times 12 = 18 \text{ млн грн.}$$

Додаткова рекламна інтеграція може додати ще від 8 до 20 млн гривень щороку, залежно від кількості партнерів, що в сумі з платною підпискою приносить дохід 26-38 млн грн в рік.

Таким чином проєкт отримує економічний ефект у вигляді збереження близько 1,426 млрд грн щороку.

4.2.3 Окупність проєкту

Окупність проєкту визначає наскільки швидко витрати на розробку проєкту будуть повернені за допомогою прямих і додаткових доходів.

Враховуючи старт експлуатації з невеликою початковою аудиторією і високими витратами на розробку, проєкт окупиться приблизно за 27 місяців, за умови, що початковою аудиторією буде 2000 користувачів з темпом зростання 5-8% в місяць; щомісячні витрати 80 тис. грн; у перший рік застосунок використовує лише рекламну інтеграцію як джерело доходу, а з початком другого року додається платна підписка за ціною 300 грн/міс.

Станом на 12 місяць експлуатації проєкт зазнає найбільших збитків, розміром 1,116 мільйона гривень, натомість на 27 місяці роботи месенджер принесе 62373,55 грн прибутку компанії при 138250 грн чистого прибутку.

4.3 Обґрунтування доцільності створення системи

Необхідність створення власного месенджера обґрунтовується не стільки технічними недоліками наявних рішень, скільки практичними викликами сучасності. Події останніх років продемонстрували вразливість залежності від іноземних комунікаційних платформ, коли зовнішні рішення можуть обмежити доступ до сервісів або передавати персональні дані користувачів третім сторонам без їхньої згоди.

Більшість популярних месенджерів знаходяться під контролем іноземних компаній. Це створює значні ризики щодо конфіденційності даних користувачів, оскільки такі платформи можуть передавати особисту інформацію, історію чатів та метадані третім сторонам відповідно до законодавства країн їхньої реєстрації. Використання таких платформ означає втрату контролю над власними персональними даними.

Проєкт було ініційовано для забезпечення незалежності у сфері цифрових комунікацій. Створення власного рішення гарантує користувачам стабільний доступ до якісного засобу спілкування незалежно від зовнішніх факторів та корпоративних рішень великих технологічних компаній.

Відкритість вихідного коду відрізняє проєкт від більшості комерційних аналогів, дозволяючи повну прозорість роботи системи. На відміну від

закритих рішень типу Viber, користувачі можуть самостійно переконатися в реальному впровадженні E2E шифрування у комунікації, адаптувати функціонал під власні потреби та не залежати від рішень корпорацій щодо політики використання.

Багато популярних месенджерів не забезпечують повного захисту метаданих або використовують частково закриті алгоритми шифрування. Створена система реалізує абсолютне наскрізне шифрування повідомлень, що забезпечує максимальний рівень приватності користувачів.

Технічна реалізація продемонструвала можливість створення високопродуктивного рішення з використанням сучасних відкритих технологій. Оптимізована архітектура дозволяє досягти високої швидкості роботи при мінімальному споживанні ресурсів, що особливо помітно порівняно з ресурсозатратними платформами типу месенджера Discord.

Локальна розробка означає можливість швидкого реагування на потреби локальної аудиторії, впровадження необхідних функцій та виправлення проблем без необхідності очікування рішень міжнародних корпорацій. Система може розвиватися відповідно до реальних потреб користувачів у максимально короткі терміни.

Створення власного месенджера обґрунтовується необхідністю впровадити незалежне рішення для забезпечення стабільних комунікацій з дотриманням конфіденційності. Це питання особистої свободи користувачів, їхнього контролю над власними даними та гарантованого доступу до засобів спілкування.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи успішно створено систему шифрованого кросплатформного месенджера з відкритим кодом, що повністю відповідає поставленим вимогам щодо безпеки, функціональності та кросплатформності. Розроблена система впроваджує достатній рівень конфіденційності завдяки використанню E2E шифрування на основі гібридного підходу, що поєднує симетричний алгоритм AES-256-GCM та асиметричний алгоритм RSA.

Проведений аналіз предметної області підтвердив актуальність розробки української альтернативи наявним месенджерам, оскільки популярні рішення мають суттєві недоліки у сфері конфіденційності та прозорості. Архітектурне рішення у вигляді клієнт-серверної моделі виявилося оптимальним для забезпечення надійного зберігання повідомлень та синхронізації між пристроями. Використання JWT-токенів для автентифікації забезпечило необхідний рівень безпеки при мінімальному навантаженні на сервер.

Реалізована база даних на основі MariaDB продемонструвала високу продуктивність та надійність завдяки використанню ідентифікаторів типу UUID замість цілих чисел, що підвищило безпеку системи. Серверна частина, розроблена на C++ з використанням фреймворку Pistache для REST API та uWebSockets для з'єднань у режимі реального часу, отримала високу пропускну здатність та продуктивність завдяки багатопотоковості та мікросервісній архітектурі.

Клієнтська частина, побудована на основі Qt/QML з використанням C++ для бізнес-логіки, продемонструвала відмінну адаптивність до різних платформ та продуктивність завдяки прискоренню через апаратне забезпечення. Застосування синглтон-патерну для ключових класів

забезпечило оптимальне використання системних ресурсів, а асинхронна обробка через систему сигналів і слотів гарантувала плавний користувацький досвід. Реалізований користувацький інтерфейс відповідав принципам сучасного дизайну завдяки мінімалістичному підходу та підтримці кількох тематичних оформлень.

Комплексне тестування підтвердило стабільну роботу всіх ключових функцій месенджера на різних операційних системах. Модульне та інтеграційне тестування виявило лише незначні дефекти графічного інтерфейсу на Windows, що не впливали на основну функціональність месенджера. Перевірка мережевого трафіку через Wireshark підтвердила ефективність TLS-шифрування, а аналіз бази даних довів коректність зберігання зашифрованих повідомлень.

Техніко-економічне обґрунтування продемонструвало комерційну життєздатність проєкту з очікуваною окупністю протягом 27 місяців при досягненні аудиторії у 2000 користувачів. Потенційний економічний ефект від запобігання витокам даних оцінюється у 1.4 мільярда гривень заощаджень на рік при 100 тисяч зареєстрованих користувачів.

Створена система шифрованого кросплатформного месенджера здатна забезпечити користувачів надійним інструментом для конфіденційного спілкування з повним контролем над власними даними. Для подальшого вдосконалення об'єкта розробки планується розширити типи підтримуваних форматів повідомлень, додати функціональність голосового та відеозв'язку з наскрізним шифруванням, а також впровадити можливість VPN-тунелювання для створення віртуальних приватних мережевих кімнат з іншими користувачами. Відкритість коду дозволяє проводити незалежні аудити безпеки та адаптовувати продукт під специфічні потреби різних категорій користувачів, що створює міцне підґрунтя для подальшого розвитку та комерціалізації проєкту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Telegram Messenger. *Telegram* : вебсайт. URL: https://telegram.org/ (дата звернення: 24.02.2025).

2. Viber. *Viber* : вебсайт. URL: https://www.viber.com/ua/ (дата звернення: 24.02.2025).

3. Discord - Group Chat That's All Fun & Games. *Discord* : вебсайт. URL: https://discord.com/ (дата звернення: 24.02.2025).

4. Build cross-platform desktop apps with JavaScript, HTML, and CSS. *Electron* : вебсайт. URL: https://www.electronjs.org/ (дата звернення: 26.02.2025).

5. Peer-to-Peer (P2P) Architecture. *GeeksforGeeks* : вебсайт. URL: https://www.geeksforgeeks.org/peer-to-peer-p2p-architecture/ (дата звернення: 01.04.2025).

6. JSON Web Token (JWT). *GeeksforGeeks* : вебсайт. URL: https://www.geeksforgeeks.org/json-web-token-jwt/ (дата звернення: 31.03.2025).

7. RFC 7519 - JSON Web Token (JWT). *Datatracker* : вебсайт. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 31.03.2025).

8. The WebSocket API. *MDN* : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 09.04.2025).

9. Docker Compose. *Docker Docs* : вебсайт. URL: https://docs.docker.com/compose/ (дата звернення: 09.06.2025).

10. RAII. *Cppreference* : вебсайт. URL: https://en.cppreference.com/w/cpp/language/raii.html (дата звернення: 09.06.2025).

11. Singletons in QML. *Qt Documentation* : вебсайт. URL: https://doc.qt.io/qt-6/qml-singleton.html (дата звернення: 10.06.2025).

12. Пелех М. Розробка програмного засобу для порівняння алгоритмів шифрування. МАТЕРІАЛИ СТУДЕНТ. НАУКОВО-ПРАКТ. КОНФ., м. Тернопіль, 15 трав. 2025 р. Тернопіль, 2025.

ДОДАТКИ

Додаток А

Порівняння безпеки та оптимізації месенджерів

	Telegram	Viber	Discord
Шифрування	MTProto (секретні чати – E2EE)	E2EE	TLS, без використання E2EE
Відкритість коду	Клієнт – відкритий код, сервер – закритий	Закритий код	Клієнт частково відкритий, сервер – закритий
Двофакторна авторизація	Так	Підтримка PIN-коду	Так
Приватність даних	Приховування номера, зображення профілю, дати народження тощо; підтримка псевдонімів	Приховування номера, зображення профілю, дати народження тощо	Обмежені налаштування
Синхронізація пристроїв	Звичайні чати – синхронізуються, секретні – ні	Так	Так
Платформи	Android, iOS, Windows, macOS, Linux, Web	Android, iOS, Windows, macOS	Android, iOS, Windows, macOS, Linux, Web
Використання ресурсів апаратного забезпечення	Оптимізований для мобільних пристроїв, низьке використання	Оптимізований для мобільних пристроїв, низьке використання	Схильний до вищого споживання апаратних ресурсів (особливо на ПК через використання Electron)

Додаток Б

Реалізація серверного методу автентифікації користувача

```
     void RestAPI::auth_login(const Pistache::Rest::Request
&request, Pistache::Http::ResponseWriter response) {
       using namespace Pistache::Http;
       response.headers()
           .add<Header::AccessControlAllowOrigin>("*")
           .add<Header::AccessControlExposeHeaders>("Content-
Type, Content-Length")
           .add<Header::AccessControlAllowHeaders>("Content-
Type")
           .add<Pistache::Http::Header::ContentType>(Pistache::Ht
tp::Mime::MediaType(
               Pistache::Http::Mime::Type::Application,
               Pistache::Http::Mime::Subtype::Json));

       nlohmann::json request_data;
       try {
         request_data = nlohmann::json::parse(request.body());
       } catch (const nlohmann::json::exception &e) {
         std::cerr << "JSON parse error: " << e.what() <<
std::endl;
         response.send(Http::Code::Bad_Request, "Invalid JSON");
         return;
       }
       std::string response_data;

       if (request_data["login"].empty() ||
request_data["password"].empty()) {
         response.send(Http::Code::Unauthorized, "Not all
credentials");
         return;
       }

       const auto login =
request_data["login"].get<std::string>();
       const auto password =
request_data["password"].get<std::string>();

       Database::ConnectionGuard conn(*m_connPool);

       try {
         conn->setAutoCommit(false);

         if (!Database::isLoginExists(conn, login)) {
           response.send(Http::Code::Unauthorized,
                         "Username or email does not exists");
           return;
         }
```

```
        auto user_id = Database::getUserId(conn, login);

        const auto password_hash =
Database::getUserPassword(conn, user_id.value());

        if (!Crypt::hash_verify(password,
password_hash.value())) {
          response.send(Http::Code::Unauthorized, "Wrong
password");
          return;
        }

        // insert session
        const auto session_id =
Database::insertUserSession(conn, user_id.value());
        if (!session_id.has_value()) {
          response.send(Http::Code::Internal_Server_Error,
                        "Failed to create session");
          return;
        } else {
          std::cout << "auth login session id: " <<
session_id.value() << std::endl;
        }

        // get secret
        auto secret = Database::activeSecret(conn);
        if (!secret.has_value()) {
          secret = Database::genNewSecret(conn);
          if (!secret.has_value()) {
            response.send(Http::Code::Internal_Server_Error,
                          "Failed to get active secret");
            return;
          }
        }

        // tokens generation
        const std::string access_token =
Crypt::generateJwtToken(
            session_id.value(), secret->secret, secret->id,
Crypt::TOKEN::ACCESS,
            std::chrono::system_clock::now() +
Crypt::ACCESS_TOKEN_LIVE());

        const std::string refresh_token =
Crypt::generateJwtToken(
            session_id.value(), secret->secret, secret->id,
Crypt::TOKEN::REFRESH,
            std::chrono::system_clock::now() +
Crypt::REFRESH_TOKEN_LIVE());
```

```
            nlohmann::json tokens;
            tokens["tokens"]["access"] = access_token;
            tokens["tokens"]["refresh"] = refresh_token;
            response_data = tokens.dump(2);

            conn->commit();

        } catch (const std::exception &e) {
            std::cerr << "Critical error: " << e.what() <<
std::endl;
            response.send(Http::Code::Internal_Server_Error,
                          "Critical error: " +
std::string(e.what()));
            conn->rollback();
        }

        response.send(Http::Code::Ok, response_data);
    }
```

Додаток В

Метод збереження JWT-токенів

```
    void AuthManager::saveTokens(const QString &accessToken,
                                 const QString &refreshToken) {
      m_access_token = accessToken;
      m_refresh_token = refreshToken;

      auto *writeAccess = new
QKeychain::WritePasswordJob(SERVICE_NAME, this);
      writeAccess->setKey("access_token");
      writeAccess->setTextData(accessToken);
      writeAccess->start();

      connect(writeAccess, &QKeychain::Job::finished, this,
              [this](QKeychain::Job *job) {
                if (job->error()) {
                  qCritical() << "Failed to save access token:"
                              << job->errorString();
                }
              });

      auto *writeRefresh = new
QKeychain::WritePasswordJob(SERVICE_NAME, this);
      writeRefresh->setKey("refresh_token");
      writeRefresh->setTextData(refreshToken);
      writeRefresh->start();

      connect(writeRefresh, &QKeychain::Job::finished, this,
              [this](QKeychain::Job *job) {
                if (job->error()) {
                  qCritical() << "Failed to save refresh token:"
                              << job->errorString();
                }
              });
    }
```

Додаток Г

Метод перевірки наявних JWT-токенів

```
void AuthManager::checkExistingTokens() {
  if (m_isCheckingTokens)
    return;
  m_isCheckingTokens = true;
  qDebug() << "Checking existing tokens";

  auto *readAccess = new
QKeychain::ReadPasswordJob(SERVICE_NAME, this);
  readAccess->setKey("access_token");
  connect(readAccess, &QKeychain::Job::finished, this,
[this](QKeychain::Job *job) {
    if (!job->error()) {
      m_access_token =
qobject_cast<QKeychain::ReadPasswordJob *>(job)->textData();
    } else {
      m_access_token.clear();
    }

    auto *readRefresh = new
QKeychain::ReadPasswordJob(SERVICE_NAME, this);
    readRefresh->setKey("refresh_token");
    connect(readRefresh, &QKeychain::Job::finished, this,
[this](QKeychain::Job *job2) {
      if (!job2->error()) {
        m_refresh_token =
qobject_cast<QKeychain::ReadPasswordJob *>(job2)->textData();
      } else {
        m_refresh_token.clear();
      }
      validateTokens();
      m_isCheckingTokens = false;
    });
    readRefresh->start();
  });
  readAccess->start();
}
```

Додаток Ґ

Метод генерація ключів RSA

```
void CryptManager::generateKeyPair() {
    RSA* rsa = RSA_new();
    BIGNUM* bne = BN_new();
    BN_set_word(bne, RSA_F4);

    if (!RSA_generate_key_ex(rsa, 2048, bne, nullptr)) {
        qCritical() << "Failed to generate RSA keys";
        RSA_free(rsa);
        BN_free(bne);
        return;
    }

    BIO* privBio = BIO_new(BIO_s_mem());
    PEM_write_bio_RSAPrivateKey(privBio, rsa, nullptr, nullptr,
0, nullptr, nullptr);
    char* privData;
    long privSize = BIO_get_mem_data(privBio, &privData);
    m_privateKeyPem = QString::fromLatin1(privData, privSize);
    BIO_free(privBio);

    BIO* pubBio = BIO_new(BIO_s_mem());
    PEM_write_bio_RSAPublicKey(pubBio, rsa);
    char* pubData;
    long pubSize = BIO_get_mem_data(pubBio, &pubData);
    m_publicKeyPem = QString::fromLatin1(pubData, pubSize);
    BIO_free(pubBio);

    RSA_free(rsa);
    BN_free(bne);

    savePrivateKeyToKeychain();
    uploadPublicKey();
    emit keysInitialized();
}
```