Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Веб-застосунок для прихованого обміну повідомленнями

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Софія НЕСТЕРЕНКО
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олег ЖУРИЛО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Нестеренко Софії Миколаївні
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-застосунок для прихованого обміну повідомленнями

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) документація мови програмування C# та платформи Asp.Net Core;
2) документація React;
3) документація PostgreSQL;
4) документація ASP.NET Core SignalR;
5) документація IndexedDB;
6) редактор програмного коду Visual Studio Code;
7) інтегроване середовище розробки Rider.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) аналіз використовуваних технологій;
3) програмна реалізація;
4) інструкція користувача;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технології розробки та інструменталь-	31.05.25-02.06.25	
	них засобів		
3	Розробка алгоритмічного забезпечення	03.06.25-05.06.25	
4	Розробка та відлагодження програмного	06.06.25-09.06.25	
	забезпечення		
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Олег ЖУРИЛО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 88 с., 42 рис., 1 табл., 2 дод., 29 джерел.

ІНТЕРНЕТ, ПРОТОКОЛ ЗВ’ЯЗКУ, СЕРВЕР, МЕСЕНДЖЕР, СТЕГАНОГРАФІЯ, КРИПТОГРАФІЯ, ШИФРУВАННЯ, SIGNALR, WEB API, БАЗА ДАНИХ.

Метою кваліфікаційної роботи є створення вебзастосунку для прихованого обміну повідомленнями.

У ході виконання кваліфікаційної роботи було розроблено вебзастосунок для прихованого обміну повідомленнями. Проєкт спрямований на вирішення актуальної проблеми безпечної передачі конфіденційних даних в умовах зростаючого контролю. Були досліджені та імплементовані сучасні методи стеганографії, зокрема LSB (Least Significant Bit), а також механізми шифрування для забезпечення конфіденційності інформації. Розроблений застосунок дозволяє користувачам вбудовувати текстові повідомлення у зображення, роблячи їх непомітними для стороннього спостерігача. Особлива увага приділялася зручності інтерфейсу та надійності алгоритмів, що дозволило створити ефективний інструмент для захищеного спілкування. Результати роботи демонструють високу ефективність обраних підходів та потенціал для подальшого розвитку у сфері інформаційної безпеки.

ABSTRACT

Bachelor’s thesis: 88 pages, 42 figures, 1 tables, 2 appendices, 29 sources.

INTERNET, COMMUNICATION PROTOCOL, SERVER, MESSENGER, STEGANOGRAPHY, CRYPTOGRAPHY, ENCRYPTION, SIGNALR, WEB API, DATABASE.

The primary goal of this thesis is to develop a web application for hidden message exchange.

In order to complete the qualification work, a web application for covert message exchange was developed. The project aimed to solve the urgent problem of confidential data transmission in the face of growing control. Modern steganography methods, particularly LSB (Least Significant Bit), as well as encryption mechanisms, were researched and implemented to ensure information confidentiality. The developed application allows users to embed text messages into images, making them imperceptible to an external observer. Certain attention was given to the user-friendliness of the interface and the reliability of the algorithms, which enabled the creation of an effective tool for secure communication. The results of the work demonstrate the high effectiveness of the chosen approaches and the potential for further development in the field of information security.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СКБД – система керування базами даних

AES – стандарт розширеного шифрування (англ., Advanced Encryption Standard)

E2EE – наскрізне шифрування (англ., End-to-End Encryption)

GDPR – загальний регламент захисту даних (англ., General Data Protection Regulation)

IV – вектор ініціалізації (англ., Initialization Vector)

JWT – веб-токен JSON (англ., JSON Web Token)

LSB – найменш значущий біт (англ., Least Significant Bit)

PEM – формат для зберігання криптографічних ключів (англ., Privacy-Enhanced Mail)

PSNR – пікове співвідношення сигналу до шуму (англ., Peak Signal-to-Noise Ratio)

RS-аналіз – регулярний-сингулярний аналіз (англ., Regular-Singular Analysis)

RSA – алгоритм асиметричного шифрування (англ., Rivest-Shamir-Adleman)

SHA-256 – алгоритм безпечного хешування на 256 біт (англ., Secure Hash Algorithm 256-bit)

TLS – захист транспортного рівня (англ., Transport Layer Security)

XEPs – протоколи розширення XMPP (англ., XMPP Extensions Protocols)

XMPP – розширюваний протокол обміну повідомленнями та присутності (англ., Extensible Messaging and Presence Protocol)

ВСТУП

З розвитком технологій у сучасному світі більшість людей обирають спілкування за допомогою програм для онлайн комунікації через текстові повідомлення, замість фізичного спілкування. Вже вважається загально прийнятим попереджати текстовим повідомленням в месенджерах або по електронній пошті про майбутній дзвінок. Це дозволяє людям краще керувати своїм часом і відповідати у зручний для них момент. Але з перевагами онлайн-комунікації постає питання безпеки особистого листування та захисту приватної інформації від сторонніх осіб.

Розробники програм для онлайн комунікації повинні забезпечити необхідний рівень захисту відповідно до законодавства багатьох країн, в тому числі законів про захист персональних даних і цифрової конфіденційності, зокрема GDPR (General Data Protection Regulation). Цей закон діє в Європейському Союзі та є найсуворішим у світі нормативним актом щодо захисту та обробки персональних даних, зокрема він контролює роботу месенджерів [1].

На сьогоднішній день найвідомішим способом захисту особистих даних у месенджерах є використання наскрізного шифрування (End-to-End Encryption, E2EE) [2]. Багато популярних засобів для онлайн спілкування використовують подібні протоколи, такі як Signal Protocol, які мають гарантувати безпеку [2]. Але більшість подібних технологій викликають недовіру, тому що за зашифрованим зв’язком часто здійснюється стеження. Стеганографія може бути рішенням цієї проблеми для месенджерів, оскільки вона приховує сам факт передачі повідомлення, і дані матимуть вигляд, наприклад, звичайних зображень, не привертаючи уваги [3].

Таким чином, розробка програмного забезпечення з використанням стеганографії є актуальним рішенням для багатьох проблем, що стосуються конфіденційності та безпеки користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність прихованого обміну повідомленнями

Зростання вимог до конфіденційності у сфері цифрової комунікації зумовлене як технічним прогресом, так і посиленням контролю за мережевим трафіком. У багатьох країнах спостерігаються тенденції до впровадження стеження, цензури та фільтрації контенту.

Криптографія, що ґрунтується на перетворенні даних у форму, зрозумілу лише авторизованим користувачам, широко використовується для забезпечення приватності [4]. Однак, незважаючи на впровадження наскрізного шифрування [2], традиційні месенджери залишаються вразливими до аналізу метаданих, що дозволяє ідентифікувати учасників комунікації та характер взаємодії між ними.

У цьому контексті стеганографія, яка передбачає вбудовування даних у мультимедійні файли, виступає альтернативним методом забезпечення конфіденційності [5]. Такий підхід дозволяє не лише захистити вміст повідомлень, а й приховати сам факт їх передачі, знижуючи ймовірність виявлення з боку систем моніторингу [6].

Водночас ефективне застосування стеганографії потребує врахування існування спеціалізованих методів стегоаналізу, спрямованих на виявлення прихованої інформації. Як і у випадку з криптографією, яка постійно вдосконалюється у відповідь на розвиток криптоаналізу, стеганографічні методи повинні адаптуватися до дедалі складніших алгоритмів аналізу цифрових медіа [6].

Поєднання криптографії зі стеганографією формує ефективну модель безпечної цифрової комунікації, що особливо актуально в умовах зростаючого контролю та вразливостей традиційних засобів обміну даними.

1.2 Аналіз існуючих рішень

Сучасні месенджери відіграють головну роль у приватному онлайн-спілкуванні, а їхня безпека та конфіденційність є важливими аспектами для користувачів. Це зумовлює потребу розробників у впровадженні надійних протоколів шифрування та передачі даних, які також мають відповідати законодавчим вимогам [1]. Однак, незважаючи на потенціал стеганографії для прихованої комунікації, більшість сучасних месенджерів традиційно покладаються виключно на криптографічні протоколи для забезпечення конфіденційності. Це створює потребу в детальному аналізі існуючих рішень для виявлення їхніх обмежень щодо приховування факту передачі повідомлень. У цьому контексті особливу увагу потребують Signal Protocol, Matrix Protocol, XMPP, IRC та пропрієтарний протокол MTProto. Кожен з них має свої особливості в забезпеченні безпеки комунікацій, але жоден не надає вбудованої підтримки стеганографії.

1.2.1 Signal Protocol

Signal Protocol, розроблений Open Whisper Systems, є одним з найбільш авторитетних та криптографічно стійких протоколів наскрізного шифрування, який забезпечує конфіденційність та цілісність повідомлень із зменшенням розкриття метаданих [7].

Однією з характеристик є комбіноване використання алгоритму Double Ratchet, потрійного обміну ключами Діффі-Геллмана (3-DH) та ефемерних сесійних ключів, що забезпечує високий рівень криптографічної безпеки, включаючи досконалу пряму секретність [7]. Схема роботи алгоритму Double Ratchet представлена на рисунку 1.1. Його використання запобігає дешифруванню попередньої переписки навіть у разі компрометації довгострокових ключів. Алгоритм Double Ratchet використовує два типи ключів: Root Key та Chain Keys. Root Key використовується для генерації

нових пар Chain Keys. Кожна пара Chain Keys складається з Sending Chain Key та Receiving Chain Key, які використовуються для шифрування та дешифрування окремих повідомлень. Після кожного обміну повідомленнями Chain Keys оновлюються, що забезпечує forward secrecy – властивість протоколу, при якій компрометація поточних ключів не ставить під загрозу безпеку попередніх повідомлень.

Протоколу має відкритий вихідний код та успішно пройшов незалежні аудити безпеки, що підтверджує його надійність. Signal Protocol використовується в Signal, WhatsApp та Facebook Messenger (секретні чати).

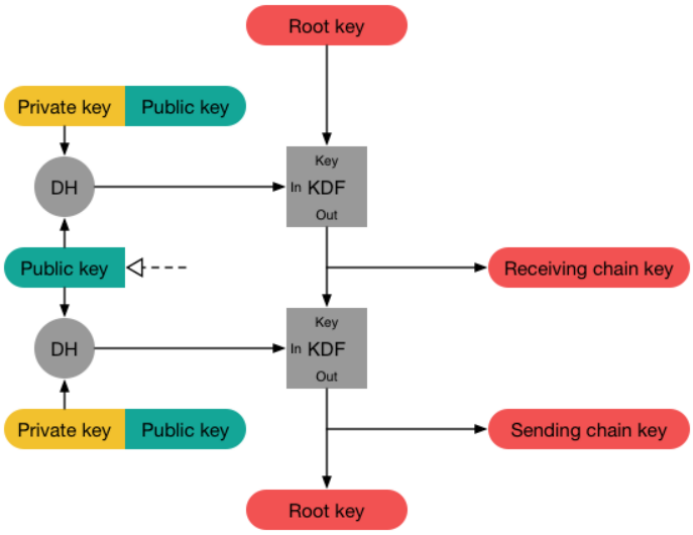

Рисунок 1.1 – Генерація ключів у Signal за алгоритмом Double Ratchet

Переваги Signal Protocol:

- забезпечення надійного наскрізного шифрування;
- позитивні результати незалежних аудитів безпеки;
- реалізація досконалої прямої секретності.

Недоліки Signal Protocol:

- залежність від централізованої інфраструктури на етапі початкового обміну ключами та виявлення контактів користувачів;
- прив’язка ідентифікатора користувача до номера телефону.

1.2.2 Matrix Protocol

Matrix Protocol є відкритим стандартом, призначеним для децентралізованої комунікації в реальному часі. Його основна мета – забезпечення сумісності між різними платформами та надання користувачам контролю над їхніми даними [8].

Децентралізована архітектура передбачає, що мережа Matrix не контролюється жодною центральною організацією. Кожен сервер у мережі є незалежним і може належати будь-кому. Федерація – це концепція, за якою різні сервери можуть взаємодіяти між собою. Користувачі на одному сервері можуть спілкуватися з користувачами на іншому сервері. Така архітектура протоколу передбачає мережу серверів, де користувачі можуть обирати власний сервер або розгортати його самостійно (рисунок 1.2).

Matrix підтримує наскрізне шифрування, яке потребує активації для кожної сесії. Olm алгоритм, розроблений для наскрізного шифрування, використовується для спілкування між двома користувачами. Він забезпечує forward secrecy та deniability (можливість заперечення авторства повідомлення). Megolm використовується для групових чатів та є розширенням Olm, оптимізованим для ефективної обробки шифрування великої кількості повідомлень для багатьох учасників. Мессенджери Element, FluffyChat, Cinny та Nheko є прикладами його використання.

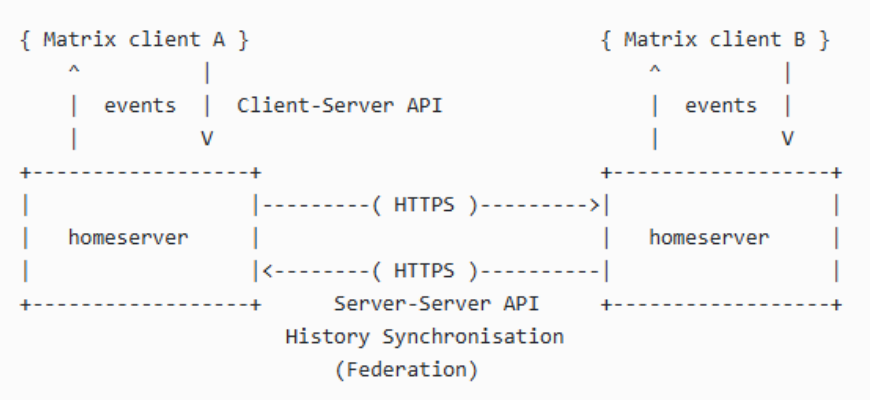

Рисунок 1.2 – Архітектура протоколу Matrix

Переваги Matrix Protocol:

- забезпечення децентралізованої та федеративної комунікаційної інфраструктури, що усуває залежність від єдиного центрального органу;
- відкритий стандарт і код забезпечують прозорість;
- надання користувачам можливості підвищити конфіденційність своїх розмов за допомогою опціонального наскрізного шифрування.

Недоліки Matrix Protocol:

- вимагає значних технічних знань та зусиль для самостійного розгортання та підтримки серверів;
- можлива фрагментація мережі;
- наскрізне шифрування не активоване за замовчуванням, що може призвести до випадкового розкриття вмісту повідомлень.

1.2.3 Extensible Messaging and Presence Protocol (XMPP)

Extensible Messaging and Presence Protocol (XMPP) являє собою відкритий та розширюваний протокол обміну повідомленнями, що історично використовувався переважно для організації миттєвих текстових комунікацій. Його децентралізована архітектура надає користувачам можливість взаємодіяти через різноманітні сервери, формуючи таким чином федеративну мережу.

Схема роботи протоколу XMPP, що ілюструє обмін повідомленнями між клієнтами через сервери, представлена на рисунку 1.3. Протокол має високу адаптивність завдяки механізму розширень XMPP Extensions Protocols (XEPs), які дозволяють додавати підтримку різноманітних функціональних можливостей, включаючи передачу мультимедійних даних та реалізацію наскрізного шифрування (наприклад, з використанням розширення OMEMO) [9]. Сфера застосування XMPP включає такі клієнти, як Conversations, Dino, Gajim та Siskin IM, що демонструє його використання в сучасних комунікаційних рішеннях.

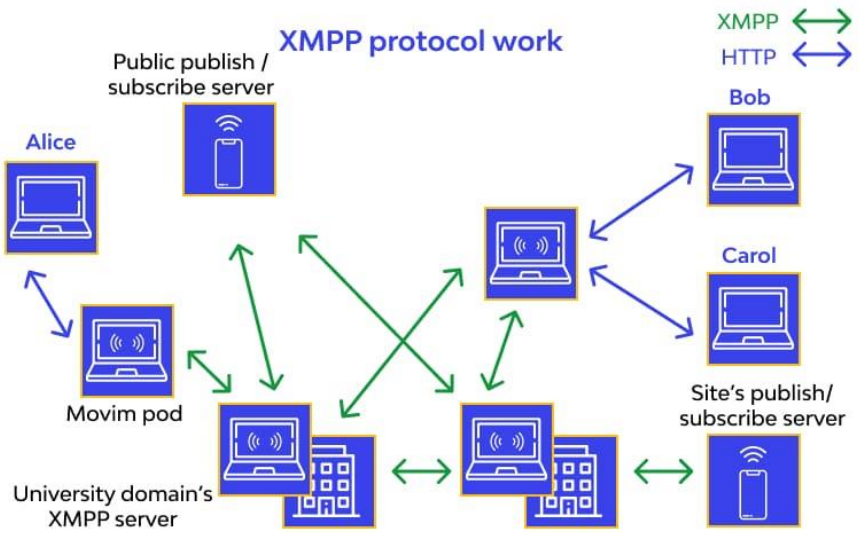

Рисунок 1.3 – Принцип роботи протоколу XMPP

Переваги XMPP:

- забезпечення відкритого стандарту та децентралізованої архітектури для незалежної комунікації;
- висока розширюваність завдяки XMPP Extensions Protocols (XEPs), що додають нові можливості;
- гнучкість у застосуванні для різних типів комунікації та інтеграцій.

Недоліки XMPP:

- наскрізне шифрування залежить від розширень;
- історичні вразливості безпеки, що потребують постійного оновлення;
- можлива несумісність між різними реалізаціями розширень, що ускладнює уніфікацію функціональності.

1.2.4 Internet Relay Chat Protocol (IRC)

Internet Relay Chat (IRC) є протоколом для обміну текстовими повідомленнями в реальному часі, що переважно використовується для організації багатокористувацьких чатів у спеціалізованих каналах. Як один з найдавніших інтернет-протоколів для комунікації, IRC вирізняється концептуальною простотою та низькими вимогами до ресурсів.

Схема роботи IRC, що ілюструє підключення клієнтів до серверів для обміну повідомленнями, представлена на рисунку 1.4. Базова специфікація протоколу не передбачає інтегрованих механізмів наскрізного шифрування, що може становити загрозу для приватності переданих даних на рівні серверів [10]. Незважаючи на це, окремі клієнтські застосунки та розширення можуть надавати додаткові засоби криптографічного захисту, хоча їх використання не є стандартизованим. До сфери застосування IRC належать такі клієнти, як mIRC, HexChat, Irssi та WeeChat.

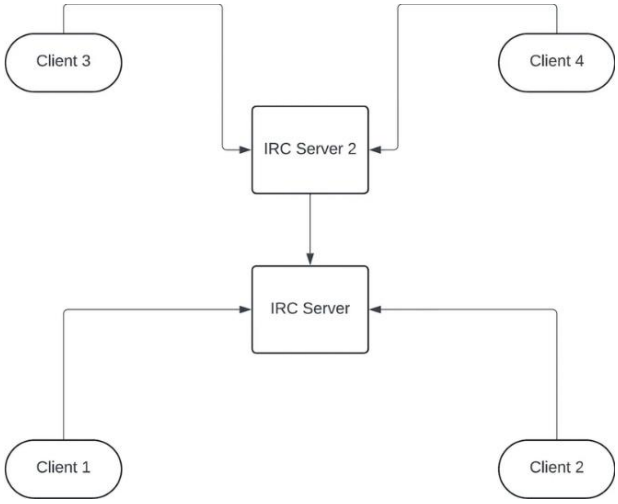

Рисунок 1.4 – Принцип роботи протоколу IRC

Переваги IRC:

- забезпечення простої архітектури та невисоких вимог до обчислювальних ресурсів;
- відкритість протоколу та наявність значної кількості клієнтських і серверних застосунків;
- децентралізована структура мережі завдяки великій кількості незалежних серверів.

Недоліки IRC:

- базовий протокол не передбачає вбудованого наскрізного шифрування.
- обмежений набір стандартних функцій;

- потенційні ризики для приватності користувачів через відсутність шифрування за замовчуванням та відкриту передачу даних на серверах.

1.2.5 MTProto

MTProto є пропрієтарним протоколом, розробленим компанією Telegram для забезпечення безпеки та швидкості обміну повідомленнями у власному месенджері. Протокол використовує багаторівневу схему шифрування, де транспортний рівень захищається TLS.

Згідно зі схемою роботи (рисунок 1.5), для наскрізного шифрування в «секретних чатах» застосовується комбінація симетричних (AES) та асиметричних (RSA) алгоритмів шифрування, а також протоколу Діффі-Геллмана для обміну ключами [11]. Однією з головних цілей розробки MTProto є стійкість до криптоаналізу та висока швидкість передачі даних, навіть при нестабільному інтернет-з’єднанні.

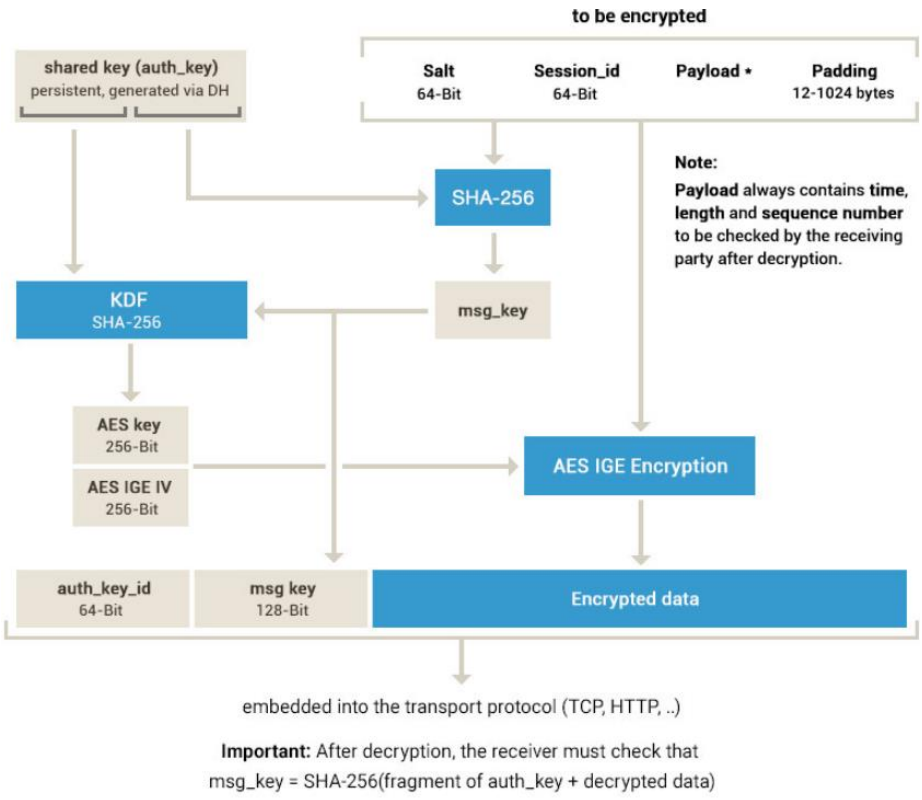

Рисунок 1.5 – Схема шифрування хмарних чатів у MTProto

Відсутність відкритого вихідного коду MTProto викликає сумніви щодо його безпеки [11]. Внаслідок пропрієтарності протоколу, його специфікації не повністю відкриті для публічного огляду та незалежного аналізу. Це ускладнює перевірку його безпеки та виявлення потенційних вразливостей. Користувачам залишається лише довіряти заявам розробників щодо його безпеки, що не завжди є прийнятим у сфері кібербезпеки [12]. Використання нестандартних режимів шифрування, таких як AES-IGE, який не пройшов такого ж рівня публічної перевірки, як більш поширені режими, може бути причиною для занепокоєння серед експертів. На відміну від інших протоколів, MTProto не має формального криптографічного доведення безпеки. Це означає, що його безпека не була математично доведена, а спирається на аналіз та тестування.

Переваги MTProto:

- забезпечення високої швидкості передачі даних;
- застосування багаторівневого шифрування для захисту комунікацій на різних етапах;
- оптимізація протоколу для ефективної роботи на мобільних пристроях з обмеженими ресурсами.

Недоліки MTProto:

- пропрієтарність протоколу обмежує можливості незалежного аналізу та вдосконалення;
- наявність критики з боку окремих експертів у галузі криптографії щодо його безпечності;
- залежність від централізованої інфраструктури серверів Telegram.

1.2.6 Результати порівняння

Порівняльний аналіз зосереджений на протоколах обміну повідомленнями, оскілки вони забезпечують основні можливості месенджерів, такі як безпека, конфіденційність і прихований обмін даними,

виявив значні відмінності між розглянутими протоколами. Зведені результати порівняння за різними критеріями наведені у таблиці 1.1.

Проведений аналіз показав, що жоден із існуючих протоколів не забезпечує повної комбінації високої безпеки, децентралізації, прозорості та анонімності.

Signal Protocol демонструє найвищий рівень криптографічного захисту, але залежить від централізованої інфраструктури. Matrix та XMPP підтримують децентралізацію, проте мають складність у налаштуванні та обмеження в реалізації наскрізного шифрування. MTProto хоча й швидкий, є закритим і менш перевіреним з боку спільноти, а IRC морально застарів і не відповідає сучасним вимогам до приватності.

У зв’язку з цим можна зробити висновок, що жоден із наявних протоколів не приховує обмін повідомленнями, що робить використання методів стеганографії особливо актуальним.

Таблиця 1.1 – Огляд аналогів

Критерій аналізу	Signal Protocol	Matrix Protocol	XMPP	IRC	MTProto
1	2	3	4	5	6
Наскрізне шифрування (E2EE) за замовчуванням	+	-	-	-	-
Є відкритий стандарт протоколу	+	+	+	+	-
Є повністю відкритий вихідний код клієнта та сервера	+	+	+	+	-
Реалізація протоколу проста для розробників	-	-	-	+	-

Продовження таблиці 1.1

1	2	3	4	5	6
Можливість значного розширення	-	+	+	-	-
Групові чати підтримують наскрізне шифрування за замовчуванням	+	-	-	-	-
Протокол забезпечує forward secrecy	+	+	-	-	+
Протокол проходив незалежний аудит безпеки	+	+	+	+	-
Є вбудована підтримка передачі файлів у протоколі	+	+	+	-	+
Клієнт протоколу має низькі вимоги до ресурсів пристрою	-	-	-	+	+
Існує розвинена екосистема бібліотек та серверів для протоколу	+	+	+	+	-
Можливість анонімної реєстрації	-	+	+	+	-

1.3 Постановка завдання

Через використання шифрування в традиційних месенджерах для захисту змісту повідомлень, сам процес передачі зашифрованих даних може

привертати увагу. З огляду на це, постає актуальне завдання розробки месенджера, що забезпечував би прихований обмін інформацією шляхом застосування стеганографічних методів.

Розроблений застосунок повинен мати наступний функціонал:

- реалізація приватного обміну текстовими повідомленнями, прихованими за допомогою стеганографії всередині цифрових зображень;
- забезпечення можливості обміну файлами з використанням методів шифрування для захисту їх вмісту;
- підтримка групових чатів для обміну повідомленнями між кількома користувачами одночасно, з використанням стеганографії;
- реалізація механізму надсилання одноразових повідомлень, які зникають після прочитання;
- збереження повідомлень та файлів для користувачів, які знаходяться офлайн, з подальшою їх доставкою після підключення до мережі;
- можливість експорту та імпорту локальних даних месенджера.

Досягнення поставленої мети передбачає виконання таких етапів:

- вибір відповідних технологій та інструментальних засобів;
- розробка та інтеграція стеганографічного методу;
- проєктування та реалізація серверної та клієнтської частин системи;
- інтеграція всіх компонентів і проведення комплексного тестування функціональності розробленого програмного забезпечення.

Реалізація зазначених етапів забезпечить успішне виконання поставлених завдань і досягнення мети проєкту.

2 ВИКОРИСТОВУВАНІ ТЕХНОЛОГІЇ

2.1 Технології розробки

Розробка програмного забезпечення передбачає попередній аналіз та вибір відповідних технологій, що відповідатимуть вимогам застосунку. Наразі існує широкий вибір фреймворків, інструментів та архітектурних рішень, кожне з яких характеризується певними особливостями та сферами застосування. Тому ще до початку реалізації проєкту важливо визначити, які саме технології будуть використані, спираючись на вимоги щодо функціональності, продуктивності та масштабованості майбутньої системи.

Для розробки месенджера з підтримкою стеганографії, необхідно враховувати додаткові вимоги до безпеки, роботи в реальному часі, а також ефективної взаємодії між клієнтською та серверною частинами. Для реалізації цих вимог слід використовувати надійні засоби шифрування, протоколи передачі даних, бібліотеки для роботи з мультимедіа, а також інструменти для зберігання та синхронізації інформації. Відповідно до поставлених вимог, обрано відповідний стек засобів розробки програмного забезпечення та технологій реалізації функціональних компонентів системи.

Засоби розробки:

- Rider – кросплатформне IDE для розробки на .NET [13];
- Visual Studio Code – редактор коду з підтримкою розширень [14];
- Pgadmin – інструмент адміністрування для роботи з базами даних PostgreSQL [15].

Технології реалізації:

- ASP.NET Core – кросплатформенний фреймворк для розробки вебзастосунків та API на .NET [16];
- React – JavaScript бібліотека для створення інтерактивних користувацьких інтерфейсів [17];

- PostgreSQL – надійна та масштабована реляційна система керування базами даних [18];
- IndexedDB – вбудована в браузер NoSQL СКБД для зберігання даних на стороні клієнта [19];
- EntityFramework Core – сучасна ORM (Object Relational Mapper) для .NET, що спрощує роботу з базами даних [20];
- Dexie.js – JavaScript бібліотека для роботи з IndexedDB [21];
- SignalR – бібліотека для додавання функціональності обміну даними в реальному часі до вебзастосунків [22];
- технології стеганографії. Метод LSB (Least Significant Bit) [23] як метод приховання текстових повідомлень всередині цифрових даних зокрема зображень шляхом маніпулювання найменш значущими бітами;
- технології шифрування AES та RSA – алгоритми симетричного (AES) та асиметричного (RSA) шифрування для забезпечення конфіденційності та безпеки файлів [24].

2.2 Технології серверної частини

Серверна частина розробленого месенджера, що включає функціональність стеганографії, реалізована з використанням .NET Core. Вибір цих технологій зумовлений їхньою продуктивністю, масштабованістю та широкими можливостями для розробки вебзастосунків.

2.2.1 ASP.NET Core

ASP.NET Core обрано як основну платформу для розробки Web API, яка забезпечує взаємодію між клієнтською та серверною частинами месенджера. Цей кросплатформний фреймворк від Microsoft використовується для створення RESTful API, обробки HTTP-запитів та керування маршрутизацією [16]. Використання ASP.NET Core дозволяє

розробити структурований та масштабований застосунок для забезпечення надійної роботи месенджера, зокрема для обробки запитів, пов’язаних з обміном повідомленнями, аутентифікацією та стеганографічними функціями, в умовах зростання кількості користувачів та обсягу даних.

2.2.2 Entity Framework Core

Entity Framework Core (EF Core) являє собою ORM для .NET Core, яка забезпечує абстракцію при взаємодії з реляційними базами даних [20]. Замість того, щоб писати SQL-запити, EF Core надає можливість оперувати даними через об’єкти .NET, використовуючи LINQ (Language Integrated Query) для формування запитів. Трансляція LINQ-запитів у відповідні SQL-інструкції виконується EF Core автоматично, що дозволяє працювати з даними на рівні об’єктної моделі. Це мінімізує необхідність написання та налагодження SQL-запитів, що спрощує розробку складних операцій з даними, характерних для функціональної частини месенджера.

2.2.3 PostgreSQL

PostgreSQL є об’єктно-реляційною системою керування базами даних з відкритим вихідним кодом [18]. Її здатність ефективно обробляти складні SQL-запити забезпечує продуктивну роботу функцій месенджера, включаючи пошук та фільтрацію даних. Гнучка архітектура PostgreSQL передбачає можливість розширення функціональності шляхом додавання нових функцій та типів даних.

Забезпечення цілісності даних у PostgreSQL досягається завдяки підтримці транзакцій, що особливо важливо при одночасній роботі багатьох користувачів та обробці великого обсягу повідомлень. Відкритий вихідний код системи не лише зменшує витрати на ліцензування, але й забезпечує підтримку проєкта великою спільнотою розробників.

2.2.4 SignalR

ASP.NET Core SignalR є бібліотекою, призначеною для спрощення розробки вебзастосунків, що потребують обміну даними в реальному часі. Замість традиційної моделі запит-відповідь, де клієнт ініціює кожен запит на оновлення даних, SignalR встановлює постійне двостороннє з’єднання між клієнтом і сервером [22]. Це дозволяє серверу передавати інформацію клієнтам одразу після її виникнення, без необхідності попереднього запиту з боку клієнта, мінімізуючи затримку.

У контексті розробки месенджера, SignalR використовується для забезпечення обміну повідомленнями в реальному часі. При надсиланні користувачем текстового повідомлення або файлу, SignalR забезпечує його доставку всім іншим учасникам відповідного чату. Завдяки своїм можливостям, ця бібліотека має ряд переваг для розробки сучасного месенджера:

- миттєвий обмін повідомленнями;
- сповіщення про події в реальному часі;
- масштабованість;
- підтримка різних транспортних механізмів;
- зменшення навантаження на сервер.

2.3 Технології клієнтської частини

Клієнтська частина розробленого месенджера, що забезпечує користувацький інтерфейс та взаємодію з сервером, побудована на основі сучасних вебтехнологій, вибір яких зумовлений популярністю та широкими можливостями для створення складних інтерфейсів користувача. Це сприяє високій швидкодії та модульності, що є важливим для підтримки та подальшого розвитку системи.

2.3.1 React

React – це JavaScript-бібліотека з відкритим вихідним кодом, розроблена Facebook (Meta), яка використовується для створення динамічних та інтерактивних користувацьких інтерфейсів (UI) [17]. В основі React лежить компонентний підхід, де UI розбивається на невеликі, незалежні та багаторазово використовувані частини – компоненти. Кожен компонент відповідає за певну частину інтерфейсу та його поведінку.

Застосування React у розробці месенджера обумовлено можливістю створення модульної структури інтерфейсу, що полегшує підтримку та масштабування проєкту в майбутньому.

2.3.2 Material UI

Material UI є бібліотекою компонентів інтерфейсу користувача для React, яка реалізує візуальний стиль Material Design, розроблений Google. Бібліотека надає широкий набір попередньо реалізованих UI-компонентів, таких як кнопки, поля введення, навігаційні панелі, діалогові вікна, іконки та інші елементи інтерфейсу. Застосування Material UI дозволяє значно прискорити процес розробки клієнтської частини, оскільки надається можливість використовувати вже готові та протестовані компоненти замість створення власних, що також сприяє покращенню загальної якості інтерфейсу користувача.

2.3.3 Dexie

Dexie.js є абстракцією над IndexedDB, розробленою для спрощення взаємодії з локальною базою даних у браузері. IndexedDB є складним у використанні низькорівневим програмним інтерфейсом для зберігання структурованих даних на стороні клієнта. Dexie надає зрозумілий об’єктно-

орієнтований інтерфейс для взаємодії з IndexedDB [21]. Бібліотека підтримує такі важливі функції, як транзакції з ACID-властивостями, керування версіями бази даних, а також методи для виконання складних запитів, що покращує швидкість роботи застосунку та забезпечує можливість обмеженого офлайн-доступу до інформації.

2.3.4 IndexedDB

IndexedDB являє собою вбудовану у браузери NoSQL базу даних, яка надає механізм для зберігання значних обсягів структурованих даних на стороні клієнта, включаючи файли та бінарні об’єкти [19]. На відміну від простішого localStorage, IndexedDB дозволяє організовувати складніші структури даних та виконувати ефективні запити з використанням індексів.

У контексті розробки месенджера, застосування IndexedDB забезпечує наступні функціональні можливості:

- зберігання великих обсягів даних;
- забезпечення офлайн-доступу;
- підтримка транзакцій;
- можливість обробки файлів.

2.4 Стеганографія

Використання стеганографії в месенджерах має свою специфіку, пов’язану з особливостями цих платформ, такими як обмежена пропускна здатність, необхідність швидкої передачі даних та підтримка певних типів файлів (текст, зображення, аудіо, відео). Розробка ефективних методів стеганографії для месенджерів вимагає врахування цих особливостей для досягнення балансу між непомітністю, ємністю приховування та стійкістю до виявлення спеціальними методами стегоаналізу.

2.4.1 Опис використовуваного методу стеганографії

Використовуваний метод стеганографії призначений для приховування інформації всередині цифрових зображень, які зазвичай використовуються для обміну в месенджерах, поєднує в собі елементи стеганографії на основі найменш значущого біта (LSB), рандомізованого вибору пікселів і шифрування для підвищення рівня безпеки та непомітності [23].

Процес вбудовування та вилучення прихованого повідомлення відбувається за наступним алгоритмом, зображеним на рисунку 2.1.

Рисунок 2.1 – Процес вбудовування та вилучення прихованого повідомлення

Описаний метод дозволяє використовувати зображення, що передаються через месенджери, як контейнери для прихованої інформації. Хоча метод не залежить від конкретного формату зображення, слід враховувати можливе стискання з втратами при використанні LSB, яке може пошкодити дані. Тому потрібно враховувати алгоритми обробки зображень месенджера при виборі параметрів вбудовування. Типи даних, які можуть бути приховані, є різними: від текстових повідомлень до різнотипних файлів.

Метод стеганографії інтегрується з функціоналом месенджера шляхом використання стандартних механізмів передачі зображень. Для передачі повідомлення, яке потрібно приховати, спочатку застосовується алгоритм

вбудовування до обраного зображення, а потім відправляється отримане стего-зображення через месенджер звичайним способом. Отримувач, який має відповідні ключі, може вилучити приховане повідомлення з отриманого зображення.

Для обґрунтування ефективності методу стеганографії доцільно розглянути його з точки зору основних технічних характеристик:

- ємність приховування;
- рівень викривлень зображення;
- стійкість до аналізу.

Ємність приховування (C) залежить від розміру зображення-контейнера та кількості найменш значущих бітів, що використовуються для вбудовування інформації. Нехай зображення-контейнер має розмір W × H пікселів. Якщо для вбудовування одного біта повідомлення використовується один найменш значущий біт (LSB) одного пікселя, то максимальна ємність приховування становить добуток ширини зображення на його висоту. У загальному випадку, при зміні n LSB кожного пікселя, ємність розраховується як добуток W, H та n. Параметр n варіюється залежно від вимог до якості зображення та обсягу повідомлення.

Для оцінки візуальних викривлень, спричинених вбудовуванням даних, часто використовується метрика пікового відношення сигнал/шум (PSNR) [25]. Для зображення розміром W × H, що має максимальне значення пікселя MAX_I, та середньоквадратичної помилки (MSE) між оригінальним зображенням I та стего-зображенням K, PSNR обчислюється за формулою:

$$PSNR = 10 \times log_{10}(\frac{MAX_I^2}{MSE}), \tag{2.1}$$

де MAX_I – максимальне значення пікселя;

MSE – середньоквадратична помилка між оригінальним і стего-зображенням:

$$MSE = \frac{1}{W \times H} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} (I(i,j) - K(i,j))^2 , \quad (2.2)$$

де W – ширина зображення;

H – висота зображення;

I(i,j) – значення пікселя в оригінальному зображенні;

K(i,j) – значення пікселя у стего-зображенні.

Розроблений метод LSB вбудовування мінімізує зміни в пікселях, що забезпечує високі значення PSNR та низьку помітність викривлень. Максимальна зміна значення пікселя при зміні одного LSB становить 1, при зміні двох LSB – 3, і так далі, до 2n−1. Такі незначні зміни не впливають на візуальне сприйняття зображення при наявності прихованих даних.

Для підвищення стійкості до аналізу, метод використовує рандомізований вибір пікселів за допомогою псевдовипадкового генератора з секретним ключем. Це гарантує рівну ймовірність вибору кожного пікселя та робить послідовність передбачуваною лише для тих, хто знає ключ.

Таким чином, запропонований метод стеганографії демонструє керовану ємність приховування, забезпечує низький рівень візуальних спотворень та стійкість до стегоаналізу завдяки рандомізації вибору пікселів.

2.4.2 Стегоаналіз та стійкість розробленого методу

Стегоаналіз – це наука про виявлення прихованих повідомлень у стего-об’єктах. Його основною метою є виявлення факту приховування інформації та, за можливості, її подальше вилучення без знання ключа або алгоритму вбудовування. Таким чином, стегоаналіз прямо протидіє стеганографії, аналогічно до того, як криптоаналіз протистоїть криптографії. Сучасні методи стегоаналізу можна класифікувати на кілька основних категорій: статистичний стегоаналіз, візуальний стегоаналіз, специфічний стегоаналіз та

стегоаналіз на основі машинного навчання [6]. Для протидії стегоаналізу, розроблений метод використовує наступні механізми:

- рандомізація пікселів – використання псевдовипадкового генератора для вибору пікселів у непередбачуваному порядку розриває кореляції між сусідніми пікселями, що ускладнює застосування статистичних методів аналізу, які часто базуються на виявленні таких кореляцій;
- шифрування повідомлення – зашифрування секретного повідомлення перед вбудовуванням гарантує, що навіть у випадку виявлення факту стеганографії, зловмисник не зможе отримати доступ до змісту повідомлення без ключа дешифрування. Використання криптографічних хеш-функцій, таких як SHA-256, для генерації ключів шифрування підвищує безпеку, оскільки хеш є односторонньою функцією, і відновити оригінальний ключ буде вкрай складно.

Хоча метод LSB-вбудовування вразливий до статистичних стегоаналітичних атак, таких як RS-аналіз, застосування псевдовипадкового вибору пікселів значно ускладнює виявлення закономірностей. Візуальний стегоаналіз неефективний через мінімальні зміни, що робить їх практично непомітними. Отже, метод забезпечує достатній рівень захисту за умови правильного налаштування параметрів.

2.5 Криптографія

У месенджері для безпечної передачі файлів застосовується криптографія. На відміну від стеганографії, яка приховує сам факт передачі повідомлення, криптографія забезпечує конфіденційність даних шляхом їх шифрування, роблячи незрозумілими для неавторизованих осіб [2, 24]. Це особливо важливо для файлів, які можуть містити чутливу інформацію та вимагають найвищого рівня захисту від несанкціонованого доступу.

Для безпечної передачі файлів у месенджері необхідні криптографічні методи, які гарантують конфіденційність та цілісність даних. На відміну від

криптографії, стеганографія, яка приховує факт комунікації, має значні недоліки для безпеки файлів. Її вразливість до модифікацій може призвести до незворотної втрати або пошкодження прихованих даних, роблячи її неоптимальною для забезпечення цілісності та конфіденційності файлів.

Обраний метод криптографії використовує комбінацію двох основних криптографічних технологій: асиметричного шифрування RSA та симетричного шифрування AES [24].

Для керування ключами і безпечного обміну ними використовується алгоритм асиметричного шифрування RSA (Rivest–Shamir–Adleman). Кожен користувач месенджера має унікальну пару ключів RSA:

- публічний ключ – загальнодоступний, використовується для шифрування даних, призначених власнику відповідного приватного ключа;
- приватний ключ – є секретним і відомий лише власнику. Він використовується для дешифрування даних, зашифрованих за допомогою відповідного публічного ключа.

Для генерації ключів RSA використовується розмір 2048 біт, що забезпечує високий рівень криптографічної безпеки. Асиметричне шифрування, хоча й є більш повільним при використанні у порівнянні з симетричними методами шифрування, є необхідним для безпечного обміну криптографічними ключами, оскільки дозволяє двом сторонам, які раніше не обмінювалися секретною інформацією, безпечно встановити спільний секрет. Для безпосереднього шифрування вмісту файлів перед їх передачею використовується алгоритм симетричного шифрування AES (Advanced Encryption Standard). Основні характеристики AES:

- симетричність;
- унікальний ключ та IV;
- швидкість та ефективність.

Процес шифрування та передачі файлу з використанням комбінації AES і RSA реалізується за алгоритмом, зображеним на рисунку 2.2. Використання асиметричного шифрування RSA для керування ключами та

симетричного шифрування AES для шифрування даних забезпечує високий рівень безпеки передачі файлів [24].

Алгоритм RSA є одним з найнадійніших асиметричних алгоритмів шифрування, особливо при достатній довжині ключів. Його безпека базується на обчислювальній складності факторизації великих цілих чисел, яка вважається нерозв'язною за прийнятний час для великих ключів [24].

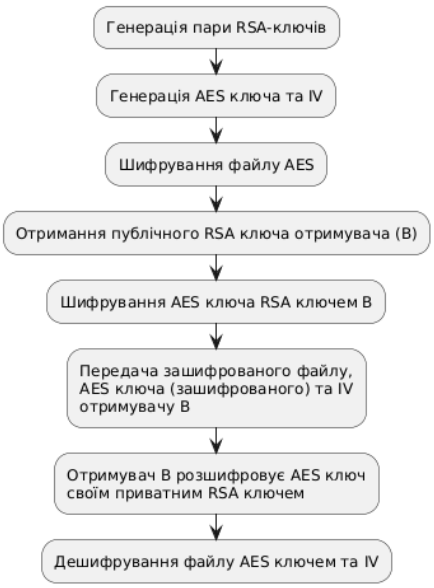

Рисунок 2.2 – Процес шифрування та передачі файлу

Алгоритм AES є стандартом симетричного шифрування та вважається стійким до сучасних методів криптоаналізу. Використання унікального ключа та вектора ініціалізації (IV) для кожного файлу підвищує безпеку, запобігаючи атакам на основі повторюваних шаблонів шифрування.

Таким чином, комбіноване застосування цих двох криптографічних примітивів забезпечує комплексний захист конфіденційності та цілісності переданих файлів у месенджері.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Архітектура вебзастосунку

Вебзастосунок реалізовано за архітектурою клієнт-сервер (рисунок 3.1), що передбачає функціональний розподіл між двома основними компонентами: клієнтською та серверною частиною. Ця архітектурна парадигма забезпечує гнучкість, масштабованість та ефективне управління системними ресурсами [26].

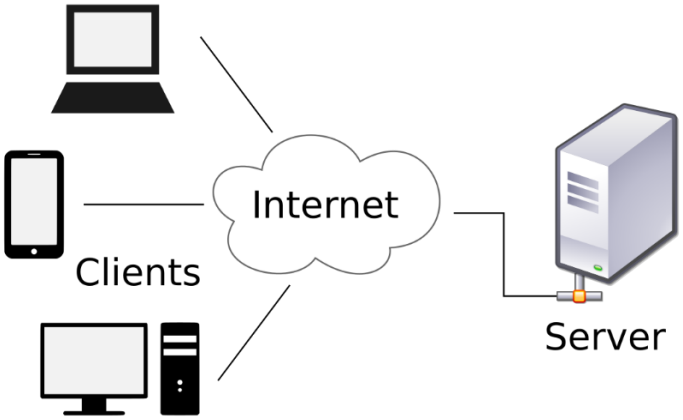

Рисунок 3.1 – Клієнт-серверна архітектура

Клієнтська частина відповідає за динамічний та інтерактивний користувацький інтерфейс. Вона побудована з використанням компонентного підходу для ефективного оновлення елементів відображення [27]. Для уніфікації візуального стилю застосовано стандартизовану бібліотеку компонентів Material UI.

Архітектура серверної частини базується на принципах чистої архітектури [28], що забезпечує чітке розділення відповідальності між компонентами, сприяючи гнучкості, тестуванню та масштабованості системи. Проект структурований на чотири шари (рисунок 3.2).

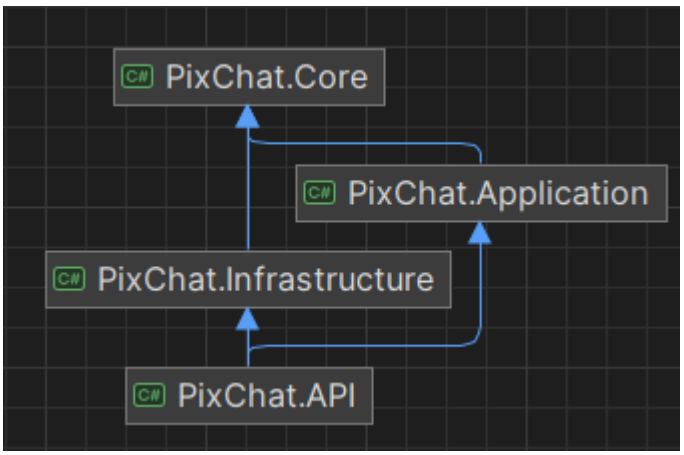

Рисунок 3.2 – Діаграма проєкту

Шар Core є ядром доменної області, що інкапсулює бізнес-сутності та інтерфейси для роботи з даними. Цей шар незалежний, забезпечує стабільність, повторне використання бізнес-логіки та обробку винятків.

Application містить прикладну бізнес-логіку та координує потік даних. Він включає об’єкти для передачі даних, інтерфейси з реалізаціями основних бізнес-правил, а також моделі вхідних запитів з валідаторами.

Infrastructure реалізує абстракції попередніх шарів, включаючи конкретні механізми для роботи з базою даних через основний контекст взаємодії. Тут також реалізовані зовнішні сервіси та допоміжні компоненти, що забезпечують стабільність і безпеку транзакцій.

API є шаром представлення та точкою входу в застосунок. Він містить програмні інтерфейси (контролери) для обробки HTTP-запитів та компонент для комунікації в реальному часі. Цей шар відповідає за представлення даних та прийом запитів, взаємодіючи з шарами Application та Infrastructure.

Зазначений підхід до архітектури забезпечує мінімальний вплив змін у зовнішніх шарах на внутрішні компоненти, що підвищує стійкість, розширюваність та адаптивність системи [29]. Це сприяє спрощенню подальшої модифікації та довгострокової підтримки програмного рішення. Такий підхід значно підвищує загальну ефективність та надійність незалежного тестування окремих модулів.

3.2 Реалізація серверної частини

Серверна частина виконує обробку бізнес-логіки, взаємодію з клієнтськими компонентами та зберігання конфіденційних даних. Її структура побудована на принципах модульного розділення, що забезпечує ізоляцію компонентів і спрощує інтеграцію нових підсистем. Такий підхід сприяє стабільній роботі системи при зміні окремих блоків без впливу на загальну архітектуру.

3.2.1 Структура бази даних

База даних у СКБД PostgreSQL є основою для зберігання всієї інформації застосунку (рисунок 3.3). Її структура визначена за допомогою Entity Framework Core, що забезпечує об’єктно-реляційне відображення.

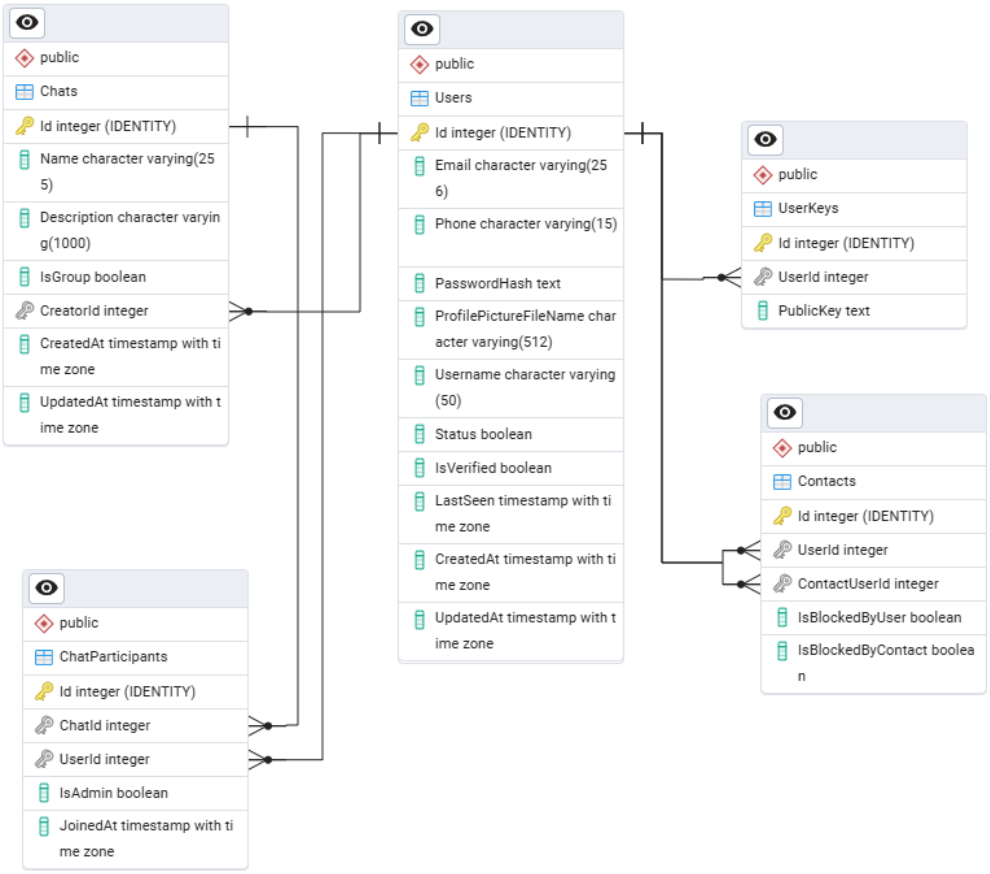

Рисунок 3.3 – ER-діаграма бази даних

Структура таблиць включає:

- Users містить дані про користувачів системи, включаючи ідентифікатор, електронну пошту, номер телефону, хеш пароля, ім’я користувача, статус, ознаку верифікації, час останнього входу та часові позначки створення й оновлення;

- Contacts зберігає інформацію про контакти між користувачами, включаючи ідентифікатори користувача та контактного користувача, а також статуси блокування з обох сторін;

- Chats описує чати, включаючи їх ідентифікатор, назву, опис, тип (груповий/приватний), ідентифікатор ініціатора та часові позначки;

- ChatParticipants зв’язує користувачів з чатами, в яких вони беруть участь, вказуючи роль учасника та час приєднання;

- UserKeys зберігає публічні криптографічні ключі RSA для кожного користувача.

Окрім основних таблиць, були передбачені окремі таблиці, призначені для тимчасового зберігання інформації, яку клієнт не зміг отримати одразу через перебування в офлайн-режимі або інших технічних обставин:

- FriendRequests тимчасово фіксує запити на дружбу між користувачами, вказуючи ідентифікатор відправника та отримувача;

- OfflineMessages та OfflineMessageFiles призначені для зберігання повідомлень. Ці таблиці містять ідентифікатори відправника та отримувача, ідентифікатор чату, дані стего-зображення або файлу та часові позначки;

- OneTimeMessages використовується для зберігання метаданих одноразових повідомлень до моменту їх прочитання. Містить ідентифікатор повідомлення, відправника, отримувача, ідентифікатор чату, дані стего-зображення та позначки стану.

Зазначені таблиці не мають зовнішніх ключів, оскільки зберігають дані з обмеженим життєвим циклом і видаляються одразу після обробки. Такий підхід дозволяє уникнути надлишкових каскадних операцій при видаленні основних сутностей і підвищити загальну стійкість системи до помилок. Дані

зберігаються ізольовано та контролюються виключно програмною логікою. Всі сутності конфігуруються за допомогою Fluent API в ApplicationDbContext та відповідних конфігураційних файлах, що забезпечує гнучке відображення об’єктів на таблиці бази даних та визначення їх властивостей. Такий підхід дозволяє детально налаштовувати гнучкі взаємозв’язки, обмеження та поведінку сутностей.

3.2.2 Аутентифікація та реєстрація

Серверна частина реалізує повний цикл аутентифікації та реєстрації користувачів у файлі AuthController.cs. Процес реєстрації передбачає отримання даних користувача, хешування пароля за допомогою стандартних механізмів безпеки та збереження інформації про нового користувача (лістинг 3.1). Після успішної реєстрації користувачу надсилається код двофакторної аутентифікації на електронну пошту, що реалізується відповідним сервісом. Верифікація реєстрації підтверджується шляхом введення отриманого коду підтвердження, після чого обліковий запис користувача активується.

Лістинг 3.1 – Метод контролера реєстрації користувача

```
public async Task<IActionResult> Register([FromBody]
RegisterRequest request)
{
    var user = new UserDto()
    {
        Email = request.Email,
        Username = request.Username,
        Phone = request.Phone,
        IsVerified = false,
        Status = true,
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow,
    };
    user.PasswordHash = _passwordHasher.HashPassword(user,
request.Password);
    await _userService.AddAsync(user);
    await _twoFactorService.SendTwoFactorCodeAsync(user.Email);
    return Ok(new { message = "Code sent to email"});}
```

Процес аутентифікації (входу в систему) включає перевірку облікових даних користувача (електронна пошта та пароль). У разі успішної перевірки, сервіс токенів у файлі JwtTokenService.cs генерує JSON Web Token (JWT), який містить інформацію про користувача та надає йому права доступу до захищених ресурсів API (лістинг 3.2). Цей токен використовується для подальших авторизованих запитів до серверної частини.

Лістинг 3.2 – Метод генерації JWT

```
public string GenerateToken(UserDto user)
{
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.Email),
        new Claim("id", user.Id.ToString()),
        new Claim("username", user.Username)
    };
    var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:
Key"]));
    var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
    var token = new JwtSecurityToken(
        issuer: _configuration["Jwt:Issuer"],
        audience: _configuration["Jwt:Audience"],
        claims: claims,
        expires: DateTime.Now.AddHours(3),
        signingCredentials: creds);
    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

3.2.3 Управління даними та бізнес-логіка

Для кожної основної сутності бази даних у шарі Infrastructure було створено репозиторій, який реалізує базові CRUD-операції (створення, читання, оновлення, видалення).

Репозиторії забезпечують абстракцію над механізмами зберігання даних, дозволяючи шару Application працювати з даними, без інформації про деталі їхнього зберігання. Даний підхід забезпечує відокремлення бізнес-логіки від технологій доступу до даних [29].

У шарі Application реалізовано сервіси, що інкапсулюють основну бізнес-логіку застосунку, взаємодіючи з репозиторіями та іншими сервісами. Кожен сервіс відповідає за окрему функціональну область:

- сервіс користувачів відповідає за керування користувацькими даними, включаючи оновлення профілю, зміну статусу (онлайн/офлайн) та завантаження фотографій;
- сервіс чатів керує створенням, оновленням та видаленням чатів, а також функціональністю приватних та групових чатів;
- сервіс контактів відповідає за управління зв’язками між користувачами, включаючи додавання/видалення контактів, обробку запитів на дружбу та блокування;
- сервіс учасників керує додаванням та видаленням користувачів з групових чатів;
- сервіс ключів управляє зберіганням публічних криптографічних ключів користувачів;
- сервіс офлайн-повідомлень забезпечує тимчасове зберігання та доставку повідомлень і файлів користувачам, які були офлайн;
- сервіс одноразових повідомлень керує життєвим циклом повідомлень, що зникають після перегляду;
- сервіс електронної пошти відповідає за надсилання кодів верифікації.

3.2.4 Реалізація стеганографії

Стеганографічний модуль, який відповідає за приховану передачу інформації, реалізовано у файлі SteganographyService.cs.

Метод EmbedMessage відповідає за вбудовування текстового повідомлення у піксельні дані зображення-контейнера. Процедура включає конвертацію текстового повідомлення у бітову послідовність. Далі, використовуючи псевдовипадковий генератор, ініціалізований динамічним ключем, відбувається вибір дискретних пікселів зображення та їхніх

колірних каналів (червоного, зеленого або синього). Найменш значущі біти (LSB) обраних пікселів модифікуються для приховування бітів повідомлення (лістинг 3.3). Для коректного вилучення до повідомлення додається спеціальний маркер кінця. Модифіковане зображення зберігається у форматі PNG для мінімізації втрат даних, обумовлених стисненням.

Лістинг 3.3 – Модифікація найменш значущих бітів пікселів

```
foreach (var (x, y, channel) in pixelIndices)
{
    Color pixel = bitmap.GetPixel(x, y);
    int r = pixel.R;
    int g = pixel.G;
    int b = pixel.B;
    if (channel == 0) // R
        r = (r & 0xFE) | (messageBits[bitIndex] == '1' ? 1 : 0);
    else if (channel == 1) // G
        g = (g & 0xFE) | (messageBits[bitIndex] == '1' ? 1 : 0);
    else // B
        b = (b & 0xFE) | (messageBits[bitIndex] == '1' ? 1 : 0);
    bitmap.SetPixel(x, y, Color.FromArgb(r, g, b));
    bitIndex++;
}
```

Метод ExtractFullMessage отримує приховане повідомлення зі стего-зображення. Процес є інверсією вбудовування: на основі того ж динамічного ключа генерується ідентична послідовність пікселів. З найменш значущих бітів цих пікселів вилучаються біти, які формують приховане повідомлення. Процес вилучення триває до виявлення маркера кінця повідомлення. Після вилучення, бітова послідовність декодується в текстовий формат.

Генерація динамічного ключа реалізована функцією у файлі ChatHub.cs (лістинг 3.4). Цей ключ створюється шляхом хешування комбінації ідентифікаторів відправника та отримувача (або ідентифікатора чату для групових повідомлень) за допомогою алгоритму SHA-256.

Використання такого ключа гарантує, що послідовність пікселів, обраних для вбудовування, є псевдовипадковою та унікальною для кожної комунікації.

Лістинг 3.4 – Метод генерації динамічного ключа для стеганографії

```
private string GenerateDynamicKey(string senderId, string
receiverIdOrChatId)
{
    using (SHA256 sha256 = SHA256.Create())
    {
        string input = $"{senderId}{receiverIdOrChatId}";
        byte[] hash =
sha256.ComputeHash(Encoding.UTF8.GetBytes(input));
        return Convert.ToBase64String(hash);
    }
}
```

3.2.5 Реалізація криптографії

Реалізація криптографічного модуля забезпечує конфіденційність даних шляхом їх шифрування перед передачею. Усі криптографічні операції виконуються безпосередньо на пристрої користувача.

Генерація RSA-ключів виконується методом generateRSAKeyPair у файлі cryptoService.js, який відповідає за створення асиметричних пар RSA-ключів (публічного та приватного) для кожного користувача (лістинг 3.5). Ключі генеруються з довжиною 2048 біт для забезпечення високого рівня криптографічної безпеки. Приватний ключ, який використовується для дешифрування, зберігається локально в IndexedDB, публічний ключ, призначений для шифрування повідомлень отримувачу, відправляється на сервер та зберігається для доступу іншим користувачам.

Лістинг 3.5 – Метод генерації пари RSA-ключів

```
export async function generateRSAKeyPair() {
 const keyPair = await crypto.subtle.generateKey(
  {
   name: "RSA-OAEP",
   modulusLength: 2048,
   publicExponent: new Uint8Array([0x01, 0x00, 0x01]), //65537
   hash: "SHA-256",
  },
  true,
  ["encrypt", "decrypt"]
 );
 const publicKeyJwk = await crypto.subtle.exportKey("jwk",
```

```
keyPair.publicKey);
 const privateKeyJwk = await crypto.subtle.exportKey("jwk",
keyPair.privateKey);
 return {
  publicKey: JSON.stringify(publicKeyJwk),
  privateKey: JSON.stringify(privateKeyJwk),
 };
}
```

Шифрування та дешифрування AES-ключів за допомогою RSA реалізується методами сервісу шифрування у файлі cryptoService.js. Публічний RSA-ключ отримувача використовується для шифрування симетричного AES-ключа (лістинг 3.6), а приватний RSA-ключ отримувача – для його дешифрування. Цей механізм забезпечує безпечну передачу сесійного ключа, необхідного для шифрування безпосередньо даних.

Лістинг 3.6 – Метод шифрування AES-ключа публічним RSA-ключем

```
export async function encryptAESKeyWithRSA(aesKeyBase64,
publicKeyJwk) {
 const aesKeyBuffer = base64ToArrayBuffer(aesKeyBase64);
 const publicKey = await crypto.subtle.importKey(
  "jwk",
  JSON.parse(publicKeyJwk),
  { name: "RSA-OAEP", hash: "SHA-256" },
  false,
  ["encrypt"]
 );
 const encryptedKeyBuffer = await crypto.subtle.encrypt(
  { name: "RSA-OAEP" },
  publicKey,
  aesKeyBuffer
 );
 return arrayBufferToBase64(encryptedKeyBuffer);
}
```

Шифрування та дешифрування даних за допомогою симетричного алгоритму AES реалізується методами сервісу cryptoService.js. Для кожної операції генерується унікальна пара ключ-вектор ініціалізації, що підвищує стійкість та запобігає повторним атакам (лістинг 3.7). Це забезпечує різні результати шифрування навіть для однакових відкритих текстів, зменшуючи ризик виявлення закономірностей.

Лістинг 3.7 – Метод симетричного шифрування даних за допомогою AES

```
export async function encryptDataAES(data) {
 const key = await crypto.subtle.generateKey({ name: "AES-GCM",
length: 256 }, true, ["encrypt"]);
 const iv = crypto.getRandomValues(new Uint8Array(16));
 const encryptedBuffer = await crypto.subtle.encrypt(
  { name: "AES-GCM", iv: iv },
  key,
  data
 );
 const exportedKey = await crypto.subtle.exportKey("raw", key);
 return {
  encryptedData,
  key,
  iv,
 };
}
```

3.2.6 Реалізація обміну повідомленнями в реальному часі

Механізм обміну повідомленнями в реальному часі реалізовано за допомогою бібліотеки SignalR, інтегрованої у клас ChatHub.cs.

Метод відправки повідомлення обробляє вхідні повідомлення (лістинг 3.8). Попередньо зашифроване повідомлення, AES-ключ та IV сервіс стеганографії вбудовує у стего-контейнер. Після обробки зображення дані передаються отримувачам.

Лістинг 3.8 – Метод відправки текстового повідомлення

```
var receiverUser = await
_userService.GetByEmailAsync(receiverId);
string fullMessage = $"{encryptedMessageData}|" +
                     $"{timestamp.ToString("O")}|" +
                     $"{encryptedAESKey}|" +
                     $"{iv}|" +
                     $"X7K9P2M|";
byte[] stegoImage = _steganographyService.EmbedMessage(image,
fullMessage, dynamicKey);
if (ConnectedUsers.TryGetValue(receiverId, out var receiver)){
    await
Clients.Client(receiver.ConnectionId).SendAsync("ReceiveMessage"
, chatId, senderId, Convert.ToBase64String(stegoImage));
}else{
    await
_offlineMessageService.SaveMessageAsync(offlineMessage);}
```

Метод відправки файлу спеціалізований для передачі файлів у класі ChatHub.cs. Він передає клієнтам вже попередньо зашифрований файл, AES-ключ та IV (лістинг 3.9).

Лістинг 3.9 – Метод відправки зашифрованого файлу

```
var receiverUser = await
_userService.GetByEmailAsync(receiverId);
if (ConnectedUsers.TryGetValue(receiverId, out var receiver))
{
    await Clients.Client(receiver.ConnectionId).SendAsync(
        "ReceiveFile",
        chatId,
        senderId,
        fileName,
        fileType,
        encryptedMessageData,
        encryptedAESKey,
        iv,
        timestamp
    );
}
else
{
    await
_offlineMessageService.SaveMessageAsync(offlineFileMessage);
}
```

Методи обробки підключення та відключення у класі ChatHub.cs відстежують статус підключення користувачів до хабу у реальному часі (лістинг 3.10). При підключенні статус користувача оновлюється на «онлайн», і всім його контактам надсилається відповідне сповіщення. При відключенні статус змінюється на «офлайн», і контакти отримують відповідне сповіщення, що забезпечує актуальність інформації про присутність користувачів у системі.

Обробка офлайн-повідомлень виконується сервісом офлайн-повідомлень у класі ChatHub.cs. Якщо отримувач повідомлення перебуває офлайн, повідомлення та файли зберігаються у базі даних (у відповідних сутностях для офлайн-повідомлень та файлів) і автоматично доставляються йому при наступному підключенні (лістинг 3.10).

Лістинг 3.10 – Метод обробки підключення користувача

```
public override async Task OnConnectedAsync()
{
    ConnectedUsers.AddOrUpdate(userId,
        new ConnectedUser
        {
            UserId = userId,
            ConnectionId = Context.ConnectionId
        },
        (key, existingUser) =>
        {
            existingUser.ConnectionId = Context.ConnectionId;
            return existingUser;
        });
    await _userService.UpdateUserStatusAsync(userEntity.Id,
"true");
    await SendPendingMessages(userId);
    await SendOneTimeMessages(userId);
    await SendPendingFriendRequest(userId);
    await base.OnConnectedAsync();
}
```

Сервіс одноразових повідомлень керує життєвим циклом конфіденційних даних, що зникають після перегляду. Система сповіщає про їхнє існування у файлі ChatHub.cs, передаючи лише метадані, а вміст (стего-зображення) надсилається за запитом (лістинг 3.11). Повідомлення не зберігається у відкритому вигляді на сервері, що виключає можливість його перехоплення або перегляду третіми особами. Після одноразового перегляду повідомлення автоматично видаляється з бази даних, що гарантує його повну конфіденційність та неможливість повторного доступу. Цей підхід забезпечує додатковий рівень захист даних.

Лістинг 3.11 – Метод відправки одноразового повідомлення

```
public async Task ReadOneTimeMessage(string messageId)
{
    var message = await
_oneTimeMessageService.GetMessageByIdAsync(messageId);
    if (!message.Read)
    {
        await Clients.Client(Context.ConnectionId).SendAsync(
            "ReceiveFullOneTimePendingMessage",
            message.SenderId,
            message.ChatId,
```

```
            Convert.ToBase64String(message.StegoImage),
            message.Id,
            message.CreatedAt
        );
        await
_oneTimeMessageService.DeleteMessageAsync(messageId);
    }
}
```

Для ефективної взаємодії користувачів важливе управління списком контактів. Функціональність запитів на дружбу реалізована методами відправки, підтвердження та відхилення запиту на дружбу у класі ChatHub.cs (лістинг 3.12). Це дозволяє користувачам відправляти, підтверджувати або відхиляти запити на дружбу, з відповідними сповіщеннями в реальному часі, забезпечуючи ефективне управління контактами.

Лістинг 3.12 – Методи підтвердження та відхилення запитів на дружбу

```
public async Task ConfirmFriendRequest(string userId, string
contactUserId)
{
    await _contactService.ConfirmFriendRequest(contact.Id,
user.Id);
    if (ConnectedUsers.TryGetValue(contactUserId, out var
contactUser))
    {
        await
Clients.Client(contactUser.ConnectionId).SendAsync("FriendReques
tConfirmed", userId);
    }
}
public async Task RejectFriendRequest(string userId, string
contactUserId)
{
    await _contactService.RejectFriendRequest(contact.Id,
user.Id);
    _logger.LogInformation($"Friend request rejected between
userId: {userId} and contactUserId: {contactUserId}");
    if (ConnectedUsers.TryGetValue(contactUserId, out var
contactUser))
    {
        await
Clients.Client(contactUser.ConnectionId).SendAsync("FriendReques
tRejected", userId);
    }
}
```

3.3 Реалізація клієнтської частини

Клієнтська частина розробленого месенджера забезпечує функціональність користувацького інтерфейсу та його взаємодію із серверною частиною. Вибір технологій обумовлений вимогами до швидкодії та модульності, що в сукупності забезпечує високу продуктивність системи та ефективність її подальшого супроводження.

3.3.1 Структура інтерфейсу користувача

Інтерфейс користувача реалізовано за допомогою бібліотеки React, для візуального оформлення застосовано бібліотеку компонентів Material UI, що забезпечує відповідність стилю Material Design та прискорює процес розробки завдяки наявності готових елементів інтерфейсу.

Структура інтерфейсу користувача базується на компонентному підході, де кожен елемент функціональності або візуального представлення відокремлений у окремий компонент, як у файлі ChatItem.js (лістинг 3.13). Це сприяє модульності та можливості повторного використання коду. Такі основні елементи, як списки контактів, чатів, відображення повідомлень та модальні вікна, реалізовані як самостійні компоненти.

Лістинг 3.13 – Приклад React-компонента елемента чату

```
import React from 'react';
import { ListItem, ListItemText, Avatar } from '@mui/material';
const ChatItem = ({ chat, onClick }) => {
  return (
    <ListItem button onClick={() => onClick(chat.id)}>
      <Avatar>{chat.name[0]}</Avatar>
      <ListItemText primary={chat.name}
secondary={chat.lastMessage} />
    </ListItem>
  );
};
export default ChatItem;
```

Хук управління SignalR (лістинг 3.14) у файлі useSignalR.js відповідає за керування з’єднанням у реальному часі та обробку всіх вхідних подій, таких як нові повідомлення, файли, зміни статусів і запити на дружбу. Він створює та підтримує з’єднання з хабом, автоматично намагаючись повторно підключитись у разі розриву.

Лістинг 3.14 – Фрагмент ініціалізації з’єднання та підписки на події SignalR

```
if (!connectionRef.current) {
    const newConnection = new signalR.HubConnectionBuilder()
     .withUrl('http://localhost:5038/chatHub', {
      accessTokenFactory: () => token,
      skipNegotiation: true,
      transport: signalR.HttpTransportType.WebSockets,
     })
     .withAutomaticReconnect([0, 1000, 5000, 10000])
     .configureLogging(signalR.LogLevel.Information)
     .build();
    newConnection.on('ReceiveMessage', onMessageReceived);
    newConnection.on('ReceiveGroupMessage',
onGroupMessageReceived);
     if (isMounted) {
      startConnection(newConnection);
     }
    });
    connectionRef.current = newConnection;
    setConnection(newConnection);
   }
```

3.3.2 Локальне зберігання даних

Локальне зберігання даних на клієнтській стороні реалізовано за допомогою IndexedDB, яка є вбудованою у веббраузери NoSQL базою даних, призначеною для збереження значних обсягів структурованих даних, включаючи файли та бінарні об’єкти. Такий підхід забезпечує підвищену конфіденційність за рахунок розміщення чутливої інформації безпосередньо на пристрої користувача.

Структура локальної бази даних, визначена у файлі db.js, включає декілька таблиць для організації різних типів даних (лістинг 3.15).

Лістинг 3.15 – Визначення схеми локальної бази даних у Dexie.js

```
db.version(8).stores({
 messages: '++id, userId, senderId, receiverId,
decryptedMessage, timestamp, isRead, isSent, messageId',
 chatmessages: '++id, userId, senderId, chatId,
decryptedMessage, timestamp, isRead, isSent, messageId',
 oneTimeMessages: '++id, messageId, chatId, senderId, timestamp,
isRead',
 files: '++id, userId, senderId, receiverId, chatId, fileName,
fileType, fileData, encryptedAESKey, aesIV, timestamp, isRead,
isSent, messageId, isGroup',
 keys: 'userId, publicKey, privateKey',
});
export default db;
```

Таблиця для приватних повідомлень (messages) призначена для зберігання текстових повідомлень, що надсилаються та отримуються у приватних чатах. Вона містить ідентифікатор повідомлення, ідентифікатори відправника та отримувача, розшифрований текст повідомлення, часову позначку, а також статуси прочитання (isRead) та відправки (isSent).

Для повідомлень, що належать груповим чатам, використовується таблиця для повідомлень чатів (chatmessages), яка містить аналогічні поля, доповнені ідентифікатором чату.

Метадані для одноразових повідомлень зберігаються у таблиці для одноразових повідомлень (oneTimeMessages), що включає ідентифікатор повідомлення, ідентифікатор чату, ідентифікатор відправника, часову позначку та статус прочитання.

Файли, передані у чатах, зберігаються локально у таблиці для файлів (files). Ця таблиця містить ідентифікатор файлу, ідентифікатори відправника та отримувача, ідентифікатор чату, ім’я файлу, тип файлу, дані файлу, зашифрований AES-ключ, вектор ініціалізації, часову позначку, статуси прочитання та відправки, а також прапорець, що вказує на приналежність файлу до групового чату.

Ключі для шифрування та дешифрування повідомлень зберігаються в таблиці для ключів (keys). Ця таблиця містить ідентифікатор користувача, його публічний ключ та відповідний приватний ключ.

Локальне зберігання даних на основі IndexedDB та Dexie.js значно скорочує запити до сервера, підвищуючи швидкодію інтерфейсу та ефективно обробляючи великі обсяги даних.

3.3.3 Обробка вхідних повідомлень та файлів

Клієнтська частина месенджера відповідає за комплексну обробку всіх вхідних повідомлень та файлів, що включає дешифрування, вилучення прихованої інформації та локальне зберігання даних.

При отриманні текстового повідомлення, яке надходить у вигляді стего-зображення, клієнтська частина ініціює процес вилучення прихованого тексту (лістинг 3.16). Для цього генерується динамічний ключ, який базується на ідентифікаторах відправника та отримувача (або ідентифікаторі чату для групових повідомлень) за допомогою алгоритму хешування SHA-256. Цей ключ потім використовується для взаємодії із серверним сервісом стеганографії, який здійснює вилучення оригінального повідомлення.

Спочатку отримувач запитує свій приватний RSA-ключ із локального сховища IndexedDB. Отриманий приватний ключ використовується для дешифрування симетричного AES-ключа. Для приватних повідомлень, отримувач дешифрує єдиний AES-ключ, що був зашифрований публічним ключем отримувача відправником. У випадку групових чатів, encryptedAesKey містить словник зашифрованих AES-ключів, де кожен ключ призначений для конкретного учасника групи; отримувач вибирає та дешифрує свій власний AES-ключ зі цього словника. Отриманий AES-ключ разом з вектором ініціалізації (IV) потім застосовуються для дешифрування вмісту файлу або текстового повідомлення.

Лістинг 3.16 – Фрагмент обробки вхідних зашифрованих даних

```
const responseEM = await axios.post(
 `http://localhost:5038/api/users/${user.id}/receiveMessage`,
 { base64Image, encryptedKey },
```

```
 { headers: { Authorization: `Bearer ${token}`, 'Content-Type':
'application/json' } }
);
const { message, timestamp: serverTimestamp, encryptedAesKey,
aesIv } = responseEM.data;
let aesKeyForDecryption;
if (isGroupChat) {
 const allEncryptedAesKeys = JSON.parse(encryptedAesKey);
 aesKeyForDecryption = allEncryptedAesKeys[user.id];
} else {
 aesKeyForDecryption = encryptedAesKey;
}
const privateKey = await getLocalPrivateKey(user.id);
const decryptedAesKey = await
decryptAESKeyWithRSA(aesKeyForDecryption, privateKey);
const decryptedContent = await decryptDataAES(message,
decryptedAesKey, aesIv);
```

Після успішного дешифрування та вилучення, всі отримані текстові повідомлення та файли зберігаються в локальній базі даних IndexedDB.

3.4 Тестування розробленого застосунку

Для забезпечення надійності та стабільності функціонування розробленого програмного забезпечення, а також для підтвердження його відповідності встановленим вимогам, необхідним є проведення всебічного тестування. Цей процес є невід’ємною частиною розробки програмного забезпечення, що дозволяє виявляти дефекти та вразливості на різних етапах, мінімізувати ризики та забезпечити якість кінцевого продукту. Тестування спрямоване на перевірку коректності роботи всіх компонентів системи, їхньої взаємодії, продуктивності та безпеки. В результаті підвищується ефективність розробки та зменшується час на усунення помилок.

3.4.1 Юніт-тестування

Юніт-тестування – це підхід до тестування програмного забезпечення, який полягає у перевірці найменших, ізольованих компонентів (юнітів)

системи. Його основна мета – верифікація коректної та автономної роботи кожного модуля. У рамках вебзастосунку для серверної частини реалізоване значне покриття юніт-тестами, що перевищує 60% для головних компонентів (рисунок 3.4). Тести охоплюють функціональність аутентифікації, управління чатами, повідомленнями, криптографічними ключами та стеганографією. Застосування методів ізоляції, таких як моки та заглушки, дозволило виявляти дефекти на різних етапах розробки, підвищуючи якість програмного продукту.

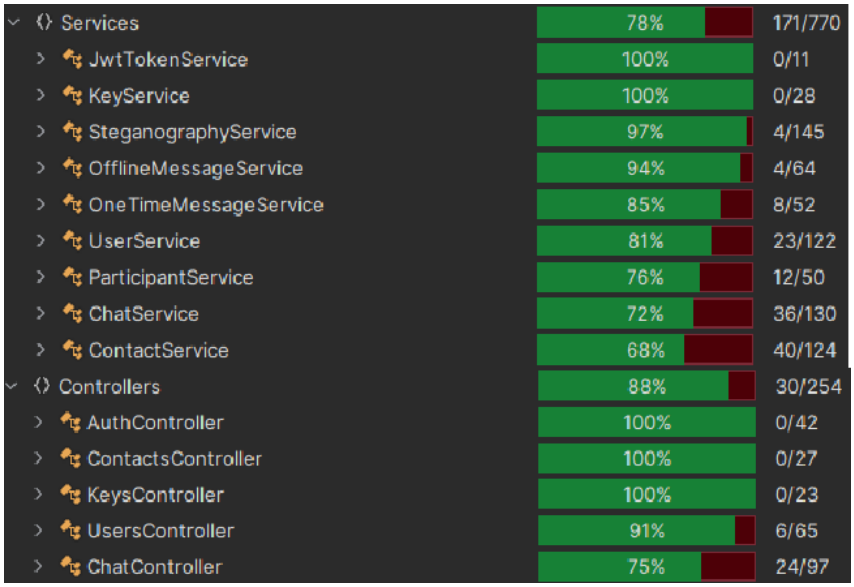

Рисунок 3.4 – Покриття юніт-тестами

3.4.2 Тестування стеганографії

Тестування реалізованого стеганографічного модуля проводилося з метою верифікації його коректної роботи, а також оцінки ефективності приховування інформації.

Візуальний аналіз (рисунок 3.5), що є первинним методом перевірки непомітності прихованих даних, не виявив помітних відмінностей між оригінальними та стего-зображеннями. Це свідчить про високий рівень прихованої передачі інформації, оскільки будь-які модифікації піксельних даних є настільки незначними, що не можуть бути сприйняті людським оком.

Відсутність візуальних артефактів підтверджує, що вбудовані дані не впливають на сприйняття зображення. Метод забезпечує високу оптичну прозорість, що є важливою характеристикою для стеганографії.

а) б)

Рисунок 3.5 – Візуальний аналіз: а) оригінального зображення; б) стего-зображення

Аналіз гістограм розподілу пікселів оригінального та стего-зображення (рисунок 3.6) виявив візуально майже ідентичні розподіли в RGB каналах.

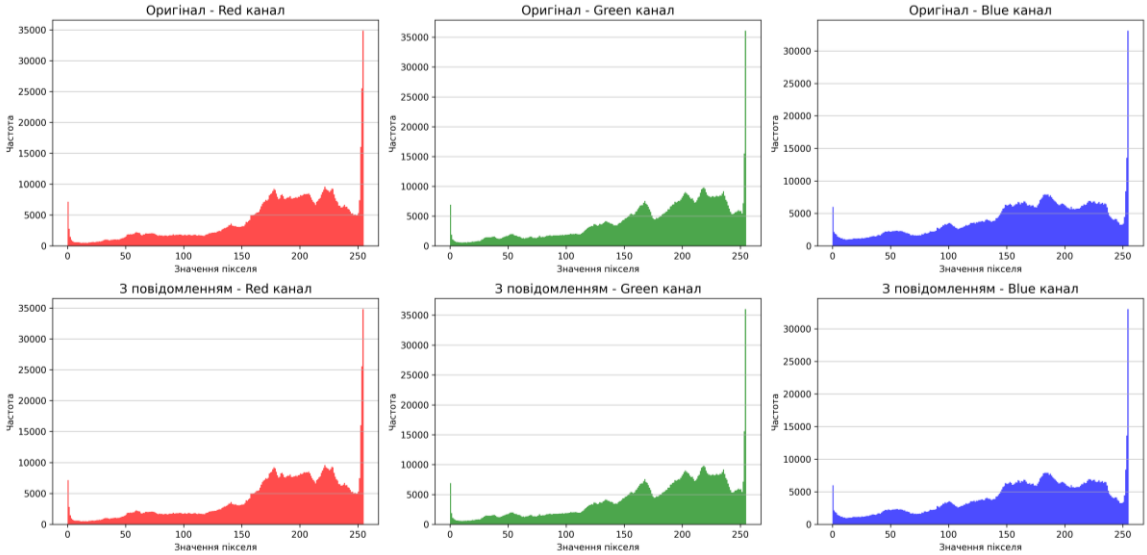

Рисунок 3.6 – Аналіз гістограм розподілу значень пікселів

Аналіз гістограм розподілу пікселів підтверджує високу візуальну непомітність застосованого стеганографічного методу, оскільки людське око

не здатне розрізнити зображення до та після вбудовування повідомлення. Однак гістограмний аналіз недостатньо чутливий для виявлення незначних модифікацій, характерних для низькоємкісної стеганографії на основі LSB.

Наступним етапом застосовано парний аналіз (Pairs Analysis), який дозволяє виявляти статистичні аномалії, пов’язані з модифікаціями найменш значущих бітів пікселів. Цей метод досліджує частотний розподіл парних та непарних значень пікселів для кожного колірного каналу. На графіках, що відображають ці поканальні різниці, спостерігаються невеликі коливання навколо нульової лінії, що є наслідком зміни парності пікселів під час вбудовування інформації (рисунок 3.7). Вони свідчать про те, що внесені модифікації є мінімальними і цілеспрямованими, впливаючи лише на найменш значущі біти піксельних значень.

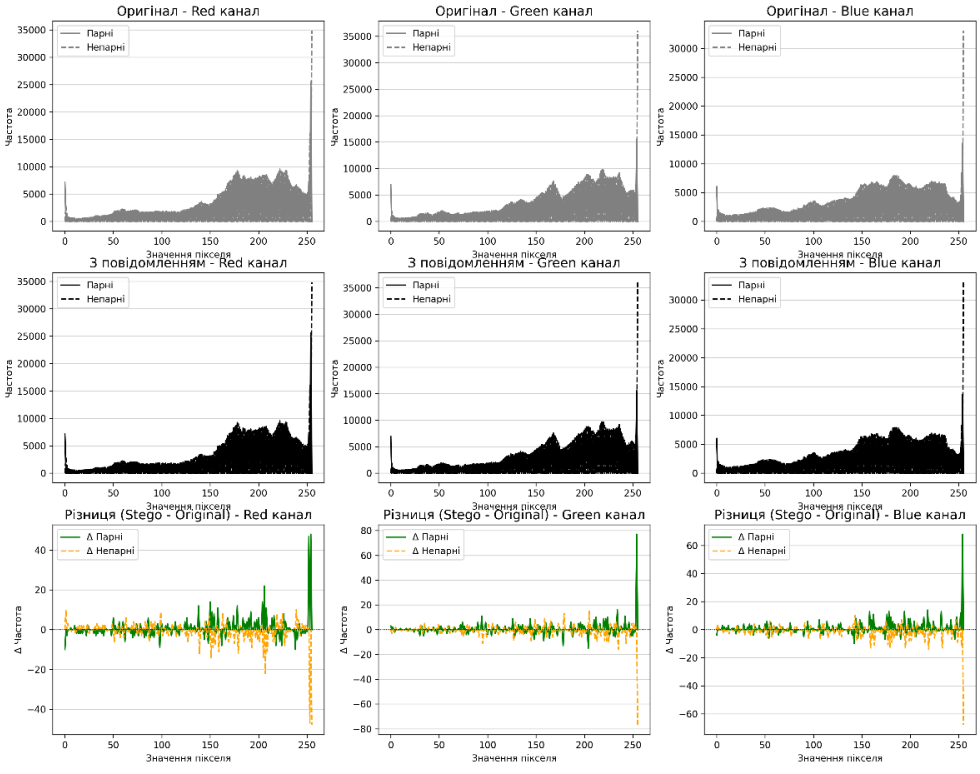

Рисунок 3.7 – Парний аналіз

Для узагальнення спостережень побудовано графік сумарної різниці частот парних та непарних пікселів по всім каналам. Загальні зміни частот коливаються близько нуля, що вказує на мінімальний вплив методу на

статистику зображення. Проте, при значенні пікселя 255 виявлено збільшення сумарної різниці для парних пікселів та зменшення для непарних. Цей аномальний пік є індикатором наявності вбудованої інформації, оскільки такі зміни на крайніх значеннях інтенсивності характерні для LSB стеганографії через обмеженість діапазону пікселів (рисунок 3.8).

Для значень, крім максимального 255, зміна найменш значущого біта призводить до збільшення або зменшення значення пікселя на 1, зберігаючи баланс між парними та непарними значеннями. При значенні 255 збільшення неможливе, тому зміна LSB викликає лише зменшення значення до 254. Це призводить до дисбалансу у частотах парних і непарних пікселів на рівні 255, що проявляється як аномальний пік на графіку. Водночас такі зміни не викликають помітних візуальних викривлень зображення.

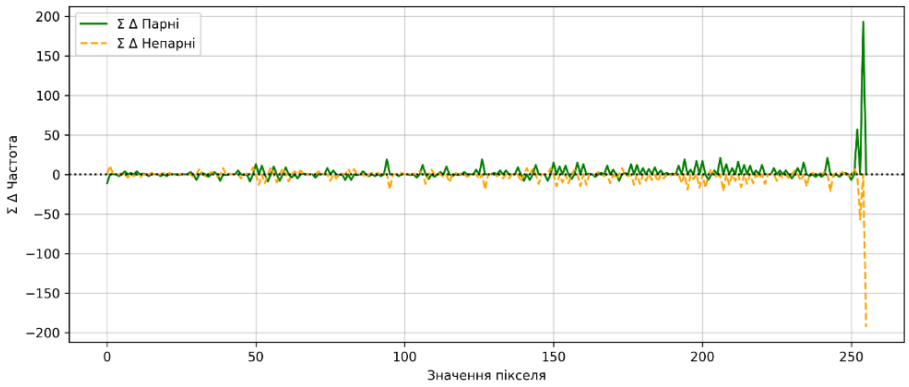

Рисунок 3.8 – Графік сумарної різниці частот парних та непарних пікселів

Додатково проведено Регулярний-Сингулярний аналіз – статистичний метод виявлення стеганографії на основі модифікації LSB, що ґрунтується на концепції гладкості груп пікселів. Вбудовування даних порушує природну статистичну гладкість, що дозволяє виявити приховану інформацію. Цей метод є широковживаним завдяки своїй простоті та високій ефективності, дозволяючи оцінити вплив модифікацій LSB на структуру зображення.

Для аналізу зображення розбивається на групи, де вимірюється гладкість до та після інверсії LSB пікселів. Групи класифікуються як регулярні, сингулярні або невизначені.

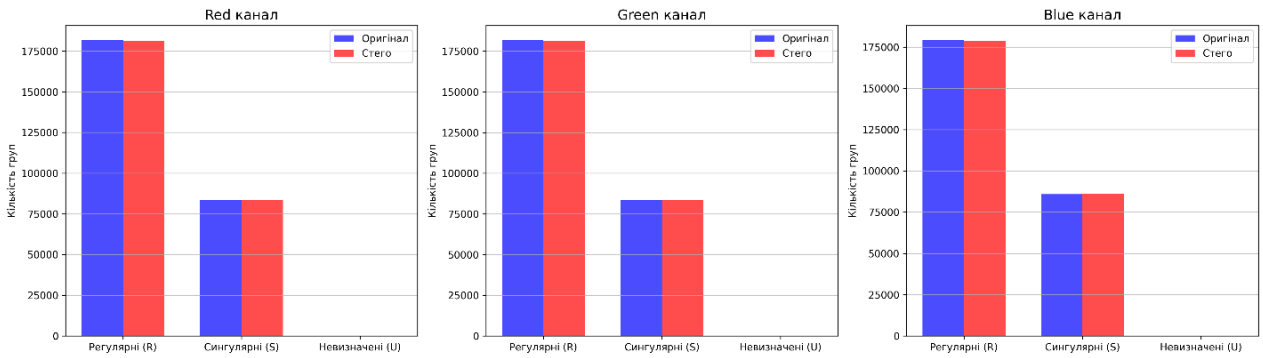

Рисунок 3.9 – RS-аналіз

За результатами RS-аналізу, кількість регулярних, сингулярних та невизначених груп в оригінальному та стего-зображенні для всіх каналів залишилась майже ідентичною (рисунок 3.9). Відсутність значних розбіжностей свідчить, що використаний стеганографічний алгоритм мінімізує вплив на RS- статистику.

Підсумовуючи результати тестування, реалізований стеганографічний модуль демонструє високий рівень візуальної непомітності прихованих даних. Отже, стеганографічний модуль забезпечує надійне приховування інформації без суттєвого впливу на візуальну якість зображення, що робить його придатним для застосування у вебзастосунках з високими вимогами до непомітності та якості передачі даних.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Реєстрація та авторизація

Доступ до функціоналу месенджера вимагає попередньої реєстрації нового облікового запису або авторизації наявного (рисунок 4.1).

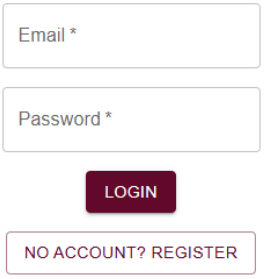

Рисунок 4.1 – Сторінка входу та реєстрації

У випадку введення некоректних облікових даних (електронної пошти або пароля) під час процедури авторизації, система генерує повідомлення про помилку, інформуючи користувача про недійсні дані. Зокрема, якщо не вірно вказана електронна пошта або не вірний пароль, відображається сповіщення про помилку (рисунок 4.2). Це дозволяє користувачу ідентифікувати причину відмови в доступі та вжити відповідні заходи, такі як перевірка введених даних або завершення процедури верифікації.

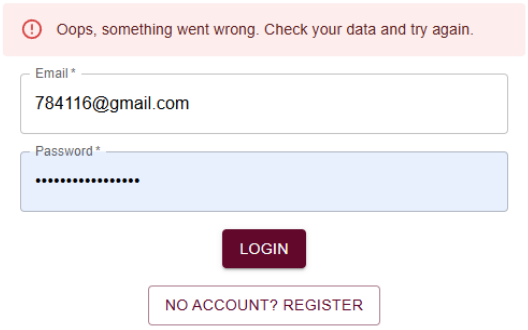

Рисунок 4.2 – Повідомлення про помилку авторизації

На екрані реєстрації користувачу необхідно заповнити обов’язкові поля, що включають Username (ідентифікатор), Email (електронна адреса для авторизації), Phone (номер телефону) та Password (пароль). Забезпечення відповідності пароля встановленим вимогам безпеки, а саме наявність мінімум 6 символів, однієї великої літери, однієї малої літери та однієї цифри, є необхідною умовою. Після внесення всіх даних, для надсилання коду верифікації, слід активувати кнопку «Send code» (рисунок 4.3).

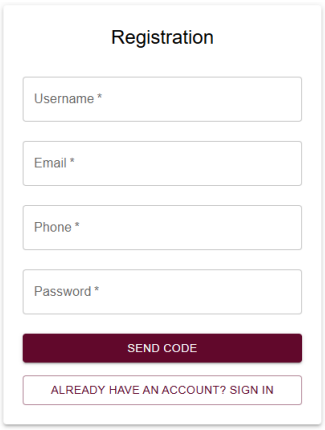

Рисунок 4.3 – Форма реєстрації нового користувача

У випадку, якщо введені користувачем дані не відповідають встановленим вимогам валідації, система відображає індикатори помилок безпосередньо під відповідними полями вводу.

Це можуть бути ситуації, коли введено некоректний формат електронної пошти, надто короткий або занадто простий пароль, що не відповідає критеріям безпеки, або телефонний номер, формат якого не відповідає стандартам. У таких випадках повідомлення про помилки з’являються безпосередньо під полями вводу, в яких було допущено неточності, що дозволяє миттєво локалізувати проблему (рисунок 4.4) та користувач має можливість виправити некоректно заповнені поля, дотримуючись рекомендацій, що відображаються, і повторно спробувати завершити реєстрацію.

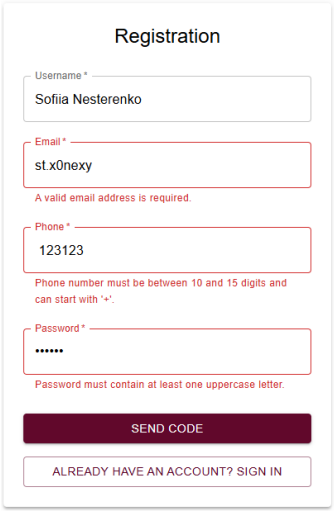

Рисунок 4.4 – Виведення помилок валідації під час реєстрації

Після успішного відправлення реєстраційної форми, на вказану електронну пошту буде надіслано шестизначний верифікаційний код (рисунок 4.5). Введення отриманого коду у спеціальне поле на екрані та подальше натискання кнопки «Confirm registration» (рисунок 4.6) завершує процес активації облікового запису.

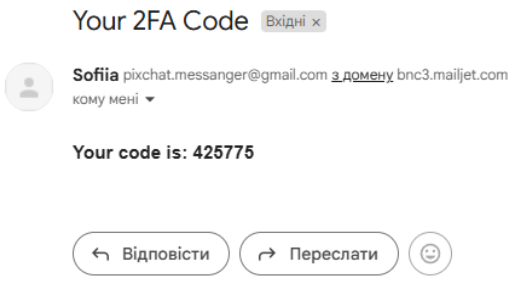

Рисунок 4.5 – Надісланий верифікаційний код

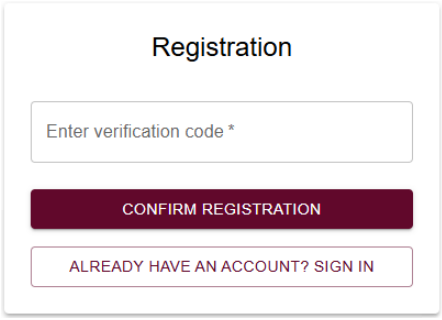

Рисунок 4.6 – Інтерфейс підтвердження реєстрації

4.2 Керування контактами

Функціонал месенджера передбачає можливості ефективного управління списком контактів, включаючи додавання нових користувачів, видалення існуючих, блокування небажаних контактів та перегляд детальної інформації про них.

Для ініціації процесу додавання нового контакту, користувачу необхідно перейти до бічної панелі навігації месенджера та вибрати іконку «Add contact», яка символізує додавання нового контакту (рисунок 4.7).

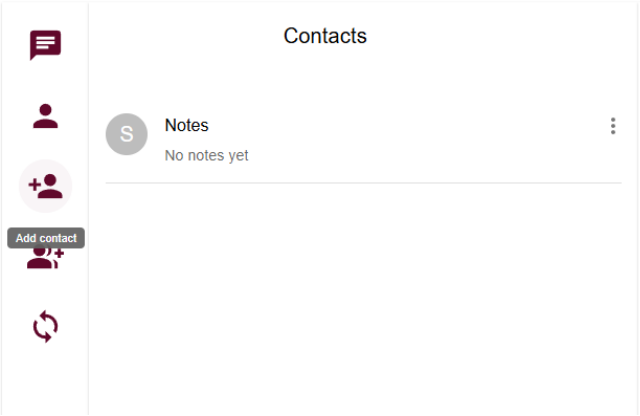

Рисунок 4.7 – Елемент інтерфейсу для додавання контакту

Після активації зазначеної іконки з’являється модальне вікно. У текстове поле «Enter contact email» слід ввести повну адресу електронної пошти цільового користувача. Після введення даних, для надсилання запиту на дружбу, необхідно натиснути кнопку «Add Contact». Цей запит підлягає підтвердженню з боку цільового користувача (рисунок 4.8).

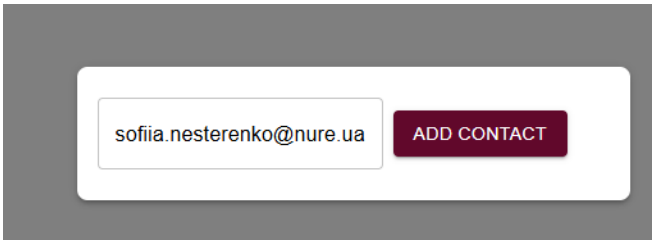

Рисунок 4.8 – Діалогове вікно додавання контакту

У випадку отримання запиту на дружбу, він з’явиться у розділі запитів на бічній панелі. Користувачу надається можливість прийняти запит, натиснувши галочку, або відхилити його, натиснувши хрестик (рисунок 4.9). Після прийняття запиту відповідний користувач автоматично додається до списку контактів, і стає доступною можливість надсилати йому повідомлення або ініціювати нову розмову. Це забезпечує зручне керування колом спілкування та дозволяє зберігати контроль над особистими контактами.

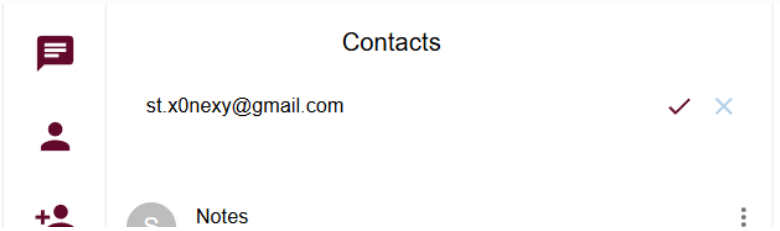

Рисунок 4.9 – Список вхідних запитів на дружбу

Відображення списку всіх активних контактів та групових чатів здійснюється на лівій бічній панелі головного інтерфейсу месенджера (рисунок 4.10). Статус присутності користувача та індикатор кількості непрочитаних повідомлень відображаються поряд з кожним елементом списку.

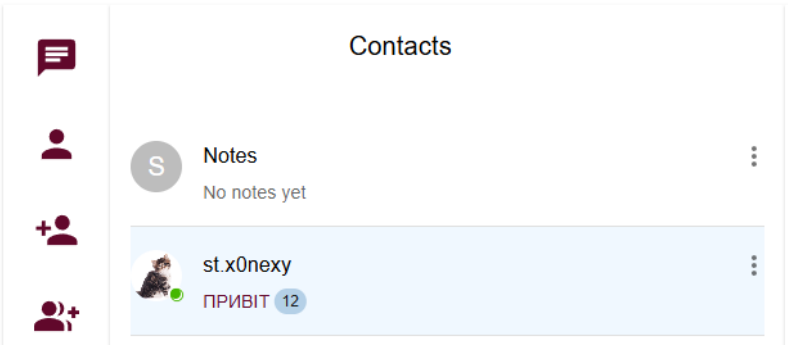

Рисунок 4.10 – Перелік контактів та чатів

При натисканні на зображення профілю контакту, відбувається відображення детальної інформації про користувача (рисунок 4.11).

Рисунок 4.11 – Детальна інформація про користувача

Для доступу до розширених опцій управління окремим контактом або чатом, таких як «Delete contact» (видалення контакту зі списку), «Delete conversation» (очищення історії листування) або «Block Contact»/«Unblock Contact» (керування блокуванням), необхідно натиснути на три вертикальні крапки, розташовані поруч із відповідним контактом (рисунок 4.12).

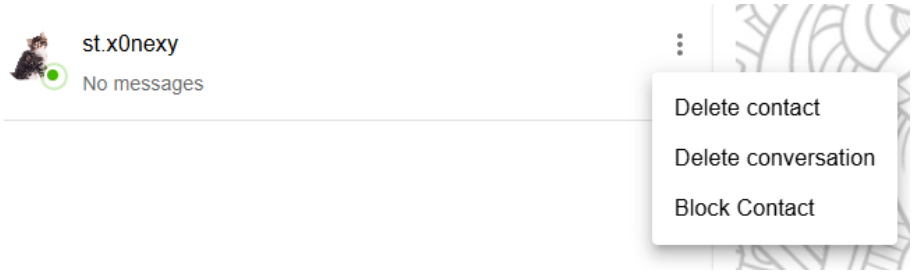

Рисунок 4.12 – Контекстне меню управління контактом

У випадку взаємного блокування або одностороннього блокування з боку співрозмовника, функціональність надсилання повідомлень деактивується, а замість поля для введення тексту відображається відповідне сповіщення про статус блокування (рисунок 4.13).

Рисунок 4.13 – Сповіщення про блокування користувача

4.3 Керування персональними даними користувача

Месенджер надає користувачам можливість переглядати та модифікувати власні персональні дані, включаючи основну інформацію профілю та зображення профілю. Доступ до сторінки керування профілем здійснюється шляхом активації іконки, розташованої на бічній панелі навігації застосунку (рисунок 4.14).

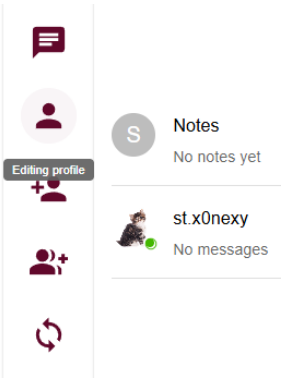

Рисунок 4.14 – Елемент інтерфейсу для переходу до профілю

На сторінці профілю користувачу відображається поточна інформація, що включає зображення профілю, ім’я користувача, адресу електронної пошти та номер телефону. Для ініціації режиму редагування цих даних необхідно натиснути кнопку «Edit» (рисунок 4.15).

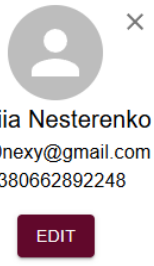

Рисунок 4.15 – Відображення профілю користувача в режимі перегляду

У режимі редагування можливо змінити Username, Email та Phone. Щоб оновити фотографію профілю, необхідно натиснути на область для

завантаження та виберіть нове зображення зі свого пристрою. Після внесення всіх необхідних змін натиснути кнопку «Save». У випадку необхідності скасування всіх змін та повернення до попереднього стану даних, слід скористатися кнопкою «Cancel» (рисунок 4.16).

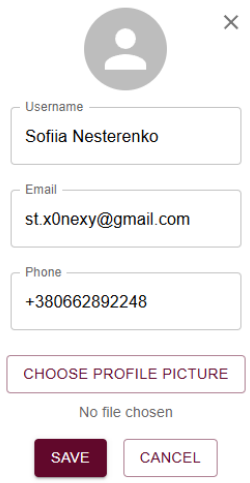

Рисунок 4.16 – Відображення профілю користувача в режимі редагування

4.4 Управління публічними чатами

Месенджер підтримує функціональність створення та управління груповими чатами, що дозволяє організовувати колективне спілкування між декількома користувачами.

Для ініціації процесу створення нового групового чату користувачу необхідно вибрати іконку «Create chat», розташовану на бічній панелі месенджера (рисунок 4.17).

Після активації іконки з’являється форма створення чату. Користувачу необхідно ввести обов’язкові дані: Chat Name (назва групового чату) та Description (опис чату, є необов’язковим). Далі слід обрати учасників нового чату зі списку контактів, відмічаючи відповідні прапорці. Після завершення вибору учасників та заповнення необхідних полів, натискання кнопки «Create Chat» завершує процес створення групового чату (рисунок 4.17).

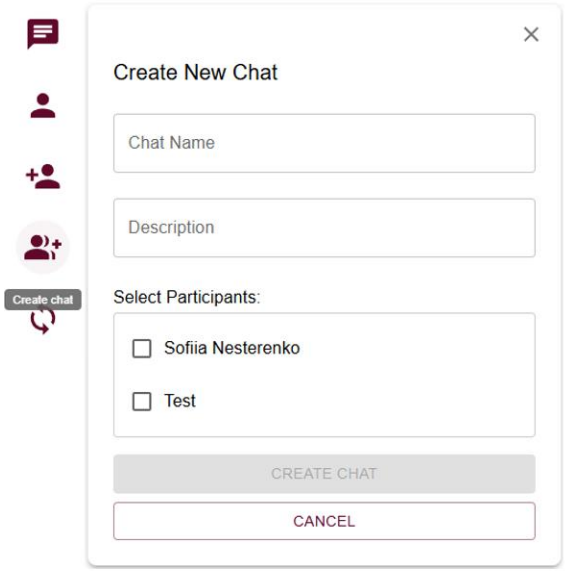

Рисунок 4.17 – Елемент інтерфейсу та форма для створення групового чату

4.5 Обмін повідомленнями

Для початку спілкування необхідно обрати відповідний контакт або груповий чат зі списку, розташованого на бічній панелі інтерфейсу. Введення текстового повідомлення здійснюється у полі введення, розташоване в нижній частині екрану чату (рисунок 4.18). Для відправлення повідомлення слід скористуватись кнопкою «Send» або натиснути клавішу Enter.

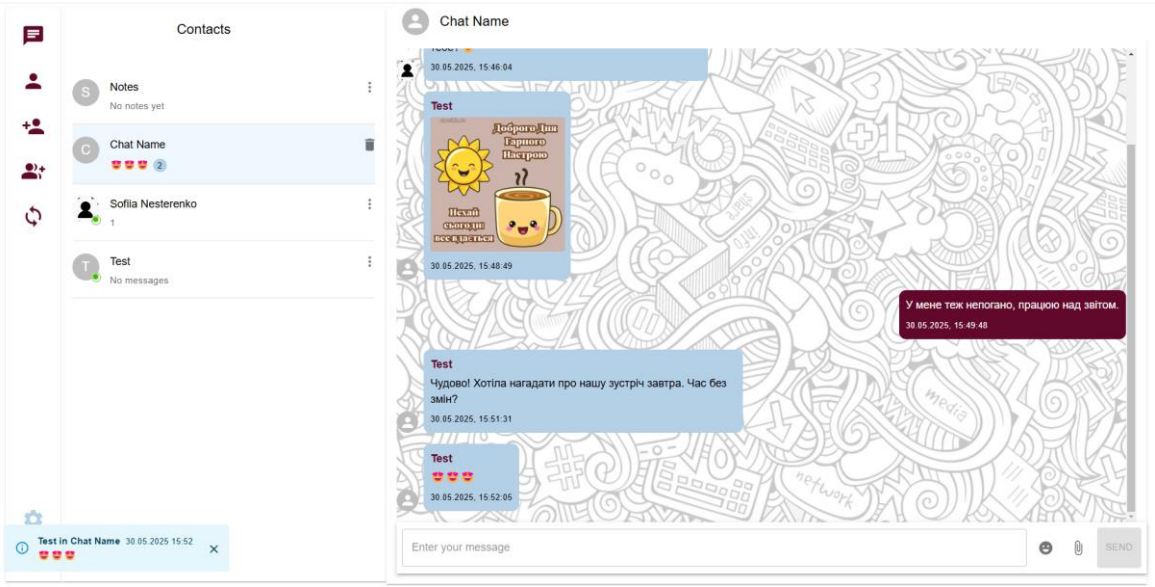

Рисунок 4.18 – Інтерфейс введення та надсилання повідомлення

4.6 Відправка одноразових повідомлень

Месенджер реалізує функціональність надсилання одноразових повідомлень, що автоматично видаляються після перегляду отримувачем, забезпечуючи підвищений рівень конфіденційності.

Для активації режиму надсилання одноразового повідомлення в приватному чаті, необхідно натиснути іконку «1X», розташовану поруч з полем для введення тексту. Зміна кольору іконки візуально підтверджує перехід до даного режиму (рисунок 4.19).

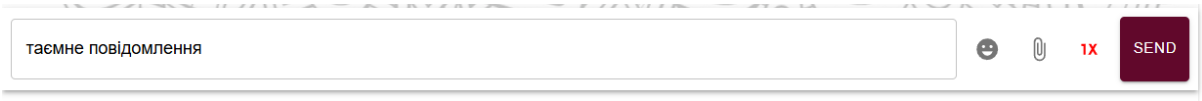

Рисунок 4.19 – Надсилання одноразового повідомлення

При отриманні одноразового повідомлення, його вміст не відображається, а з’являється повідомлення про наявність. Для розкриття вмісту повідомлення необхідно натиснути на цю позначку (рисунок 4.20).

Рисунок 4.20 – Приклад одноразового повідомлення в інтерфейсі

Натискання на це сповіщення призводить до відображення окремої панелі, де представлений перелік отриманих одноразових повідомлень, кожне з яких позначено як «New secret message».

Для доступу до вмісту конкретного одноразового повідомлення необхідно натиснути на відповідний елемент у списку. Після чого система виводить попередження про те, що повідомлення буде автоматично видалено

після його перегляду (рисунок 4.21). Користувачу слід підтвердити свою згоду на цю дію.

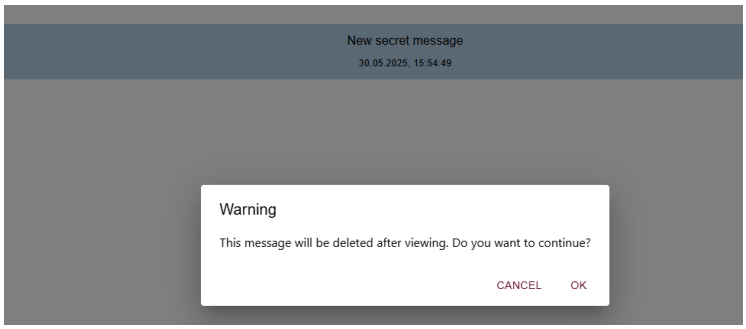

Рисунок 4.21 – Процес підтвердження перегляду одноразового повідомлення

Лише після підтвердження попередження на екрані відображається текстовий вміст повідомлення (рисунок 4.22). Важливо зазначити, що після перегляду повідомлення автоматично та безповоротно видаляється з бази даних, забезпечуючи максимальну конфіденційність.

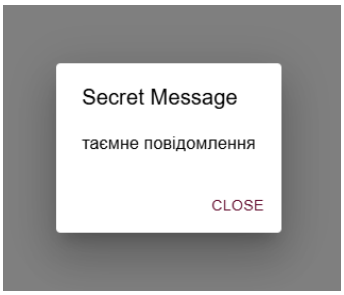

Рисунок 4.22 – Процес відображення одноразового повідомлення

4.7 Синхронізація даних

Месенджер надає користувачам можливість здійснювати експорт та імпорт даних чатів. Цей функціонал є важливим для створення резервних копій та забезпечення її перенесення між різними пристроями. Для ініціації процесу синхронізації даних, що включає операції експорту та імпорту, користувачу потрібно активувати іконку «Data synchronization», розташовану на бічній панелі навігації месенджера (рисунок 4.23).

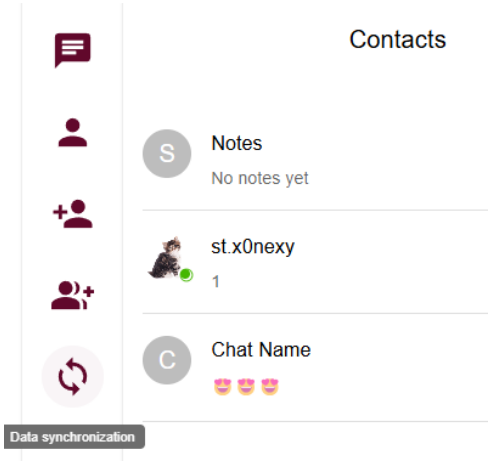

Рисунок 4.23 – Кнопка для доступу до функціоналу синхронізації даних

Після активації з’явиться модальне вікно «Data synchronization». У цьому вікні необхідно натиснути кнопку «Export data». Дані чатів, включаючи текстові повідомлення та прикріплені файли, будуть завантажені на локальний пристрій у форматі JSON-файлу (рисунок 4.24).

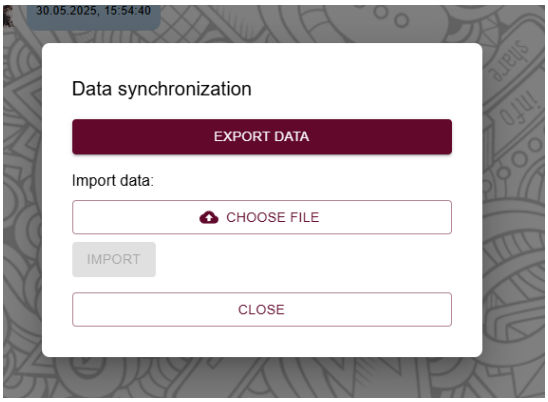

Рисунок 4.24 – Діалогове вікно синхронізації даних з опцією експорту

У тому ж модальному вікні «Data synchronization» передбачено функціонал імпорту. Користувачу слід вибрати файл JSON, який містить попередньо зарезервовані дані, використовуючи поле для вибору файлу. Після вибору файлу, необхідно натиснути кнопку «Import». Дані з обраного файлу будуть інтегровані у локальну базу даних IndexedDB клієнтської частини, що дозволить оновити або відновити історію чатів (рисунок 4.25).

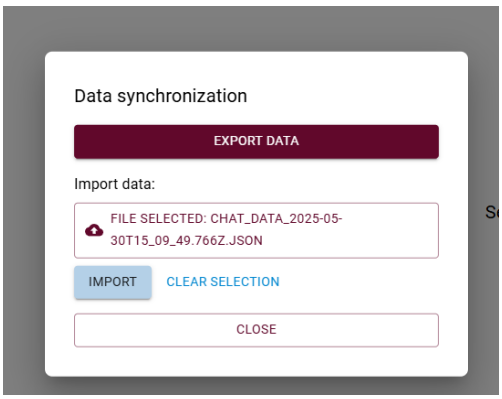

Рисунок 4.25 – Діалогове вікно синхронізації даних з опцією імпорту

Для завершення сеансу роботи з месенджером та виходу з облікового запису, користувачу необхідно натиснути кнопку «Log out», яка розташована у нижній лівій частині бічної панелі навігації (рисунок 4.26).

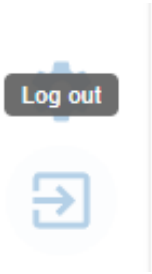

Рисунок 4.26 – Кнопка виходу з облікового запису

Таким чином, процеси реєстрації, керування контактами та персональними даними реалізовані інтуїтивно зрозумілим чином і забезпечують користувачу зручний доступ до функціоналу месенджера.

ВИСНОВКИ

У даній роботі було проведено аналіз існуючих протоколів та методів, що використовуються для безпечного обміну повідомленнями, а також розроблено вебзастосунок, який забезпечує прихований обмін інформацією з використанням стеганографії. Застосування стеганографії в месенджерах є актуальним рішенням для багатьох проблем, пов’язаних з конфіденційністю та безпекою користувачів. Розроблений вебзастосунок демонструє можливість ефективної інтеграції стеганографічних методів у засоби онлайн-комунікації.

Розроблений застосунок використовує комбінацію стеганографії та криптографії для забезпечення максимального рівня безпеки та конфіденційності. Стеганографія дозволяє приховати сам факт передачі повідомлення, маскуючи його у звичайні цифрові зображення, тоді як криптографія забезпечує захист вмісту файлів шляхом їх шифрування.

Поставлені на початку дослідження завдання були реалізовані в повному обсязі. Розроблений програмний застосунок успішно забезпечує функціональність прихованої передачі повідомлень, що є основною метою даної роботи. Функціональність прихованої передачі повідомлень досягається завдяки інтеграції методів стеганографії, зокрема, модифікованого методу найменш значущого біта (LSB), у процес обміну даними. Це дозволяє маскувати факт передачі інформації, вбудовуючи її у цифрові зображення, що знижує ймовірність виявлення прихованої комунікації.

За результатами роботи були опубліковані тези на п’ятнадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. General Data Protection Regulation GDPR. URL: https://gdpr-info.eu (дата звернення: 06.05.2025).

2. Electronic Frontier Foundation: What should I know about encryption? URL: https://ssd.eff.org/module/what-should-i-know-about-encryption (дата звернення: 06.05.2025).

3. Нестеренко С. М., Іващенко Г. С. Використання графічних зображень у якості носія для прихованої передачі даних. *Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління* : тези доп. п'ятнадцятої міжнар. наук.-тех. конф., м. Харків, 24-25 квіт. 2025 р. Харків, 2025. С. 21.

4. Drummond M. An Introduction to cryptography. *IDPro Body of Knowledge*. 2024. Vol. 1, No 13. P. 1-10.

5. Morkel T., Eloff J. H. P., Olivier M. S. An overview of image steganography. *Issa*. 2005. Vol. 1, No. 2. P. 1-11.

6. Fridrich J. Steganography in Digital Media: Principles, Algorithms, and Applications. Cambridge : Cambridge University Press, 2009. 466 с. DOI: https://doi.org/10.1017/CBO9781139192903

7. Signal Technology Foundation: The Signal protocol. URL: https://signal.org/docs/ (дата звернення: 07.05.2025).

8. The Matrix.org Foundation: Matrix specification. URL: https://spec.matrix.org/latest/ (дата звернення: 07.05.2025).

9. The XMPP Standards Foundation: Specifications. URL: https://xmpp.org/extensions/ (дата звернення: 07.05.2025).

10. Internet relay chat protocol. URL: https://www.rfc-editor.org/rfc/rfc1459 (дата звернення: 07.05.2025).

11. Telegram: MTProto Mobile Protocol. URL: https://core.telegram.org/mtproto (дата звернення: 07.05.2025).

12. Запорожець В. Ю., Опірський І. Р. Небезпека використання Telegram та його вплив на українське суспільство. *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»*. 2024. Т. 1, № 25. С. 59-78.

13. Rider JetBrains. URL: https://www.jetbrains.com/help/rider/Introduction.html. (дата звернення: 10.05.2025).

14. Visual Studio Code documentation. URL: https://code.visualstudio.com/docs/ (дата звернення: 10.05.2025).

15. pgAdmin 4 documentation. URL: https://www.pgadmin.org/docs/pgadmin4/latest/ (дата звернення: 10.05.2025).

16. ASP.NET Core documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0 (дата звернення: 10.05.2025).

17. React – The library for web and native user interfaces. URL: https://react.dev/ (дата звернення: 10.05.2025).

18. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 10.05.2025).

19. IndexedDB API. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/ (дата звернення: 12.05.2025).

20. Entity Framework Core documentation. URL: https://learn.microsoft.com/en-us/ef/core/?view=efcore-8.0 (дата звернення: 12.05.2025).

21. Dexie.js. URL: https://dexie.org/ (дата звернення: 12.05.2025).

22. Overview of ASP.NET Core SignalR. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-8.0 (дата звернення: 12.05.2025).

23. Watermarking and Steganography. Morgan Kaufmann / I. J. Cox et. al. San Francisco : Morgan Kaufmann, 2008. 593 p. DOI: https://doi.org/10.1016/B978-0-12-372585-1.X5001-3

24. Katz J., Lindell Y. Introduction to modern cryptography: principles and protocols. Boca Raton : Chapman and Hall/CRC, 2007. 552 p. DOI:

https://doi.org/10.1201/9781420010756

25. Ahmed S. S., Memon M., Jaffari R., Jawaid M. StegoBound: A Novel Image Steganography Technique Using Boundary-Based LSB. *Journal of Hunan University Natural Sciences*. 2021. Vol. 48, No. 6. P. 1–11.

26. Fowler M. Patterns of enterprise application architecture. Reading : Addison-Wesley, 2002. 576 p.

27. Components and Props. URL: https://legacy.reactjs.org/docs/components-and-props.html. (дата звернення: 12.05.2025).

28. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Upper Saddle River : Prentice Hall Press, 2017. 432 p.

29. Esposito D. Clean Architecture with. NET. Redmond : Microsoft Press, 2024. 318 p.