Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
в.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: Програмний модуль забезпечення захищеного обміну миттєвими повідомленнями

Виконавець: студентка 4 курсу, групи КБ-41

_____ **Оксана ХОМЕНКО**
(підпис) (ім'я прізвище)

	Прізвище, ініціали	**Підпис**
Керівник роботи	Андрій ФЕСЕНКО	

Нормоконтроль	Андрій БІГДАН	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
в.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«21» листопада 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентці **КБ-41** **Хоменко Оксані Вадимівні**
(група) (прізвище ім'я по-батькові)

Тема дипломної роботи Програмний модуль забезпечення захищеного обміну миттєвими повідомленнями

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 17.11.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Методи обробки даних в системах обміну миттєвими повідомленнями, криптографічні алгоритми, механізми аутентифікації, інструменти програмної розробки.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно дослідити існуючі програмні засоби для обміну миттєвими повідомленнями, розглянути підходи для забезпечення захищеності обміну миттєвими повідомленнями, обрати технології та розробити програмний модуль.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Програмний модуль для захищеного обміну миттєвими повідомленнями між користувачами на основі впровадженого наскрізного шифрування та використання інфраструктури відкритих ключів

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 18 листопада 2022 року

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ім'я, прізвище)

Завдання прийняла до виконання _____ (підпис) Оксана ХОМЕНКО (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	21.11.2022 – 23.11.2022	*виконано*
2	Аналіз літератури	24.11.2022– 06.01.2023	*виконано*
3	Дослідження захищеності обміну миттєвими повідомленнями в існуючих рішеннях	07.01.2023– 10.02.2023	*виконано*
4	Проектування системи, що забезпечує захищений обмін миттєвими повідомленнями	11.02.2023– 15.03.2023	*виконано*
5	Вибір інструментів та технологій для розробки	16.03.2023 – 19.03.2023	*виконано*
6	Розробка програмного модуля	20.03.2023 –18.04.2023	*виконано*
7	Опис програмної реалізації	19.04.2023 – 30.04.2023	*виконано*
7	Оформлення пояснювальної записки	01.05.2023 – 05.06.2023	*виконано*
8	Підготовка до захисту кваліфікаційної роботи	05.06.2023 – 12.06.2023	*виконано*

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ім'я, прізвище)

Завдання прийняла до виконання _____ (підпис) Оксана ХОМЕНКО (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 81 сторінку, включає в себе зміст, вступ, три розділи дипломної роботи, висновки та список джерел. Крім того, робота містить 3 додатки із загальною кількістю сторінок 25. У пояснювальній записці дипломної роботи міститься 23 рисунки і 4 таблиці. Список використаних джерел містить 47 найменувань і займає 5 сторінок.

Метою роботи є розробка програмного модуля, що забезпечуватиме захищений обмін миттєвими повідомленнями між користувачами.

Об'єктом дослідження є процес захищеного обміну миттєвими повідомленнями.

Предметом дослідження є методи обробки даних в системах обміну миттєвими повідомленнями.

Для досягнення мети дипломної роботи були використані наступні *методи дослідження:*

- аналіз відкритих джерел;
- порівняльний аналіз існуючих програмних засобів для обміну миттєвими повідомленнями;
- моделювання і обробка даних за допомогою програмного забезпечення.

Практична цінність отриманих результатів кваліфікаційної роботи полягає в тому, що розроблено програмний модуль для захищеного обміну миттєвими повідомленнями.

Ключові слова: месенджер, захищений обмін миттєвими повідомленнями, безпека даних, конфіденційність користувачів, наскрізне шифрування, автентифікація, Signal, PGP, програмний модуль.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

2FA – Two-Factor Authentication;
API – Application Programming Interface;
E2EE – End-to-End Encryption;
FOIA Freedom of Information Act;
GDPR General Data Protection Regulation;
GUI – Graphical user interface;
HTTP – HyperText Transfer Protocol;
IETF Internet Engineering Task Force;
IMS – Instant Messaging System;
ISO International Organization for Standardization,
MITM – Man-in-the-Middle;
OSI – Open Systems Interconnection;
PCS – Post-Compromise Security;
PFS – Perfect Forward Secrecy;
PGP – Pretty Good Privacy;
RFC Request for Comments;
PKI – Public key infrastructure;
X3DH – Extended Triple Diffie-Hellman;
ПЗ – Програмний засіб, програмне забезпечення.

ВСТУП

Актуальність. За останні декілька років, месенджери стали невід'ємною частиною нашого життя та нашою повсякденністю, оскільки це доволі зручний та швидкий спосіб спілкування в режимі реального часу. Згідно з даними [1] на вересень 2021 року, більше 3 мільярдів користувачів по всьому світу використовують найпопулярніші програми для обміну повідомленнями. Це свідчить про те, що месенджери є однією з найбільш популярних та широко використовуваних технологій комунікації в наш час. За прогнозами, кількість користувачів у 2025 році досягне 3,51 мільярди, і з огляду ще на тенденції розвитку та поширення технологій, очікується подальше зростання користувацької бази.

Відтак, в повсякденні все більше і більше людей насолоджуються простотою та зручністю, яку забезпечують системи обміну миттєвими повідомленнями в реальному часі. Проте, разом з цим *постає ряд проблем безпеки*. Наприклад, окрім забезпечення захищеності обміну повідомленнями, проблемним є питання забезпечення конфіденційності даних користувачів, які збираються та зберігаються на серверах компаній. Тому такі додатки безперечно потребують уваги дослідників безпеки. Забезпечення безпеки систем обміну миттєвими повідомленнями, як і для розробників, так і для самих користувачів, які є свідомими та небайдужими до власної інформаційної безпеки, є важливою, актуальною та перспективною темою.

Аналіз останніх досліджень та літератури підтверджує, що сфера обміну миттєвими повідомленнями розвивається швидко та дозволяє відзначити, що на сьогоднішній день існує немало розробок у галузі обміну миттєвими повідомленнями та досліджень їх захищеності [2, 3, 4]. Проте деякі існуючі дослідження, такі як EFF Secure Messaging Scorecard [5], вже застаріли і не враховують останніх тенденцій та нових ризиків.

Тому, через швидкий технологічний прогрес та появу нових загроз, виникає потреба у проведенні нових досліджень з оновленою інформацією щодо захисту

приватності та безпеки в месенджерах, формуванні актуального набору вимог, яким мають відповідати сучасні системи обміну миттєвими повідомленнями та у розробці власного програмного модуля для миттєвого обміну повідомленнями з високим рівнем захисту, що відповідає вимогам, визначеним на основі аналізу існуючих рішень. Важливо зрозуміти, які технології шифрування та захисту використовують різні месенджери, які протоколи вони використовують для забезпечення безпеки, які механізми захисту вбудовані в ПЗ, а також які є слабкі місця та потенційні загрози для приватності користувачів.

Метою дослідження є розробка програмного модуля, що забезпечуватиме захищений обмін миттєвими повідомленнями між користувачами.

Для досягнення даної мети необхідно вирішити *наступні завдання:*

- проаналізувати основні аспекти захищеності систем обміну миттєвими повідомленнями на основі існуючих рішень;
- визначити вимоги, яким має відповідати програмний модуль;
- спроектувати архітектуру програмного модуля що відповідає специфікації вимог;
- розробити програмний модуль з урахуванням встановлених вимог та архітектури;
- провести оцінку результатів, визначивши переваги та недоліки.

Об’єктом дослідження є процес захищеного обміну миттєвими повідомленнями між користувачами.

Предметом дослідження є технології, протоколи, алгоритми та методи, які використовуються для забезпечення безпеки та конфіденційності при обміні повідомленнями між користувачами в системах обміну миттєвими повідомленнями.

Для досягнення мети дипломної роботи були використані наступні *методи дослідження:*

- аналіз відкритих джерел;
- порівняльний аналіз існуючих програмних засобів для обміну миттєвими повідомленнями;

- моделювання і обробка даних за допомогою програмного забезпечення.

Практична цінність отриманих результатів кваліфікаційної роботи полягає в тому, що розроблено програмний модуль для захищеного обміну миттєвими повідомленнями.

Апробація результатів роботи та публікації. Основні результати кваліфікаційної роботи представлялися на VI Міжнародній науково-практичній конференції «Прикладні системи та технології в інформаційному суспільстві» (Київ, 2022), V Міжнародній науково-практичній конференції «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» (Київ, 2022), VI Міжнародній науково-практичній конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем” (Київ, 2023) у матеріалах наукових конференцій.

РОЗДІЛ 1
АНАЛІЗ ЗАБЕЗПЕЧЕННЯ ЗАХИЩЕНОСТІ ОБМІНУ МИТТЄВИМИ ПОВІДОМЛЕННЯМИ

1.1 Основні поняття та історія становлення систем обміну миттєвими повідомленнями

Обмін миттєвими повідомленнями, що дозволяє людям спілкуватися між собою у режимі реального часу, є важливою частиною сучасної комунікації.

Система обміну миттєвими повідомленнями (IMS – Instant Messaging System), є інформаційно-комунікаційною системою, яка дозволяє користувачам обмінюватися текстовими, візуальними та аудіо-візуальними повідомленнями в режимі реального часу через комп'ютерні мережі, зазвичай через глобальну мережу Інтернет, та включає в себе клієнтські програми, сервери, протоколи комунікації та інфраструктуру мережі, що забезпечують функціонування цієї системи. Користувачам для такого обміну миттєвими повідомленнями необхідна клієнтська програма, яку часто називають *месенджером* – конкретне програмне забезпечення, яке наділяє користувачів можливістю обмінюватися миттєвими повідомленнями та власне реалізує функцію обміну миттєвими повідомленнями повідомленнями в межах системи обміну миттєвими повідомленнями.

Тобто, система обміну миттєвими повідомленнями є більш широким поняттям, яке описує інфраструктуру та функціонал, що дозволяє обмінюватися повідомленнями в режимі реального часу, тоді як месенджер - це конкретний засіб, який дозволяє виконувати цю функцію. Проте необхідно зазначити, що в контексті даної роботи, терміни «система обміну миттєвими повідомленнями» та «месенджер» можуть використовуватися як синоніми, оскільки вони є взаємопов'язаними та нерозривними складовими частинами однієї інформаційно-комунікаційної системи і у більшості випадків термін "месенджер" використовується для позначення системи

обміну миттєвими повідомленнями, що базуються на конкретному програмному забезпеченні.

Становлення систем обміну миттєвими повідомленнями відбувалася протягом багатьох десятиліть і пройшла кілька етапів з початку 60-х років до сьогодення:

1) Історія формування обміну повідомленнями починається, коли у 1961 році в обчислювальному центрі Масачусетського технологічного інституту виникла одна з перших ОС Compatible Time-Sharing System (CTSS), і до мейнфрейму на базі якої, були підключені через віддалені термінали комутованого доступу користувачі, аби працювати зі своїми файлами, створювати, запускати програми, обмінюватися повідомленнями з іншими користувачами [6].

2) *У 1980-х роках обмін повідомленнями став більш популярним із появою Інтернету.* У 1988 році у Фінляндії було представлено Internet Relay Chat (IRC), який дозволяє користувачам підключатися до мереж за допомогою відповідного клієнтського ПЗ та спілкуватися з групами у режимі реального часу через загальнодоступні канали. IRC досяг піку популярності в 1990-х роках, але й сьогодні має сотні тисяч користувачів.

3) Лише наприкінці 1990-х років обмін миттєвими повідомленнями набув справжнього поширення із появою перших великих конкуруючих за новий ринок платформ, які дозволяли користувачам обмінюватися повідомленнями в реальному часі: ICQ від ізраїльської компанії Mirabilis, що на піку розвитку мав понад 100 мільйонів зареєстрованих користувачів, AOL придбала ICQ і її базу користувачів; AOL Instant Messenger (AIM) від американської медіакомпанії та постачальника онлайн сервісів AOL, запущений в 1997 році, що є засновником концепції «buddy list»; ще MSN Messenger від Microsoft і Yahoo! Messenger від однойменної корпорації.

4) Золотим віком обміну миттєвими повідомленнями є 2000-і, коли обмін фотографіями, можливість голосового та відеозв’язку стають звичайними функціями платформ. В той час використовувалося кілька систем обміну миттєвими повідомленнями з кількома версіями для різних платформ (Windows, Mac OS, Linux). Зокрема, Apple запустив у 2002 році iChat для своєї операційної системи OS X Jaguar,

у 2003 році Skype надає можливість користувачам Інтернету спілкуватися з іншими за допомогою відеоконференцій та обміну миттєвими повідомленнями та з 2005 року Google Talk, що тісно інтегрований з Gmail, аби полегшити спілкування між контактами електронної пошти.

5) Дещо пізніше платформи обміну повідомленнями стали пов'язані з платформами соціальних мереж після запуску першої платформи обміну миттєвими повідомленнями, побудованої в соціальній мережі MySpaceIM у 2006 році, чату Facebook Chat у соціальній мережі Facebook у 2008 році та окремої версії мобільного додатку Facebook Messenger у 2011 році.

6) Десь у 2010-ті роки: *обмін миттєвими повідомленнями переживає ренесанс, оскільки нові платформи,* такі як WhatsApp, WeChat, Slack змінюють та розширюють концепцію обміну повідомленнями. Whatsapp здійснює революцію з безкоштовним надсиланням текстів, зображень, відео та аудіо, що привабило клієнтів, які втомилися платити за надсилання текстових повідомлень. WeChat є китайським клоном WhatApp, або «китайським додатком для всього», із додатково впровадженими функціями для спілкування в чаті, як-от платежі, ігри, покупки та багато іншого. Slack є поширеним корпоративним месенджером, розроблений якраз таки для вирішення проблеми командного спілкування та спільної роботи в одному місці.

У кожному з цих етапів можна побачити зростання зручності та швидкості спілкування, проте, на жаль, розробникам не було важливо створювати додатки з урахуванням безпеки та конфіденційності. Протягом довгого часу популярні додатки для обміну повідомленнями не підтримували наскрізне шифрування, а лише стандартне шифрування клієнт-сервер, яке надає постачальникам послуг доступ до більшої кількості приватної інформації, ніж необхідно.

Коли Едвард Сноуден, колишній працівник ЦРУ та АНБ США, опублікував секретні документи [7, 8], і показав, що агентство АНБ відслідковує мільйони громадян США та інших країн, це викликало певну хвилю занепокоєння громадськості і підняло питання про право на приватність та розширення правових

обмежень на збір та обробку персональних даних. Як стверджує [9] американський криптограф Метью Грін, обурення щодо стеження, яке виявив Сноуден, точно зіграло роль, і зокрема обмін повідомленнями з наскрізним шифруванням отримав свою популярність. Тому що, раніше значна частина нашого спілкування відбувалася відкритим текстом. У 2013 році переважна більшість текстових повідомлень надсилалася через незашифровані SMS/MMS або служби обміну миттєвими повідомленнями з не зовсім вдало впровадженим рівнем шифрування, які були кошмаром конфіденційності.

1.2 Огляд існуючих месенджерів та дослідження впровадженого рівня забезпечення захищеності обміну миттєвими повідомленнями

2.1.1 Визначення методології та мети дослідження

Дослідження проводиться з метою огляду та оцінки рівня забезпечення захищеності обміну миттєвими повідомленнями в існуючих месенджерах, визначення недоліків, можливостей покращення безпеки, та, головне – специфікація вимог до розроблюваного програмного модуля за результатами.

Для досягнення цієї мети використовується методологія, що структурує процес дослідження безпеки існуючих рішень в наступні етапи:

1. Вибір існуючих месенджерів, які будуть досліджені.
2. Вивчення літератури та збір інформації з питань безпеки досліджуваних месенджерів.
3. Визначення критеріїв безпеки, що будуть використовуватися для оцінки кожного з досліджуваних месенджерів.
4. Аналіз та оцінка рівня забезпечення безпеки обміну миттєвими повідомленнями в кожному з обраних месенджерів відповідно до визначених критеріїв.

5. Оцінка результатів дослідження та порівняння безпеки різних месенджерів.

Для аналізу захищеності месенджерів використано відкриті джерела інформації, такі як:

- Офіційна документація месенджерів та їх протоколів.
- Наукові статті та публікації з предметної області.
- Результати інших досліджень безпеки месенджерів.
- Результати власних попередніх досліджень [10, 11, 12] з даної предметної області.

2.1.2 **Вибір месенджерів для дослідження**

У даній частині коротко описано вибрані додатки-месенджери, розглянуто критерії відбору месенджерів для подальшого аналізу їх рівня безпеки.

Незважаючи на те, що на ринку існує велика кількість месенджерів, у дослідженні зосередимося на деяких з них. Основними факторами вибору месенджерів стали популярність та поширеність серед користувачів, рівень заявлених захищеності та конфіденційності, доступність документації, відкритість вихідного коду, тощо.

З метою дослідження рівня забезпечення захищеності обміну миттєвими повідомленнями у існуючих рішеннях, було вибрано 6 наступних месенджерів для проведення подальшого порівняльного аналізу:

1) WhatsApp

Месенджер створений WhatsApp в 2009 році колишніми співробітниками Yahoo!, зокрема програмістом українського походження Джоном Кумом. Пізніше у 2014 році WhatsApp було продано Facebook, і з тих пір база користувачів продовжила зростати. На лютий 2023 року WhatsApp є одним з найпопулярніших додатків для обміну повідомленнями у світі, а кількість активних користувачів щомісяця досягла 2 мільярдів [13]. Загалом, якщо порівнювати WhatsApp з іншими месенджерами, то

широка аудиторія користувачів та міжнародна популярність і є його головними перевагами та особливостями. Зважаючи на те, що Meta є однією з найбільших BigTech-компаній у світі, яка відома своїми практиками збору та використання даних, можуть виникати певні занепокоєння щодо конфіденційності та безпеки використання месенджеру. У 2016 році WhatsApp перейшов на наскрізне шифрування, засноване на протоколі Signal, розробленому Open Whisper Systems. [14] містить технічне пояснення системи наскрізного шифрування WhatsApp. Також на початку 2021 року, WhatsApp оновив свою політику конфіденційності [15] та угоду користувача, що викликало багато суперечок та негативних відгуків серед користувачів, тому що змінилися правила збору та використання персональних даних користувачів [16].

2) Telegram

Як заявляють самі ж розробники [17], Telegram — це надшвидкий, простий і безкоштовний додаток для обміну повідомленнями, який орієнтований на швидкість і безпеку. На відміну від WhatsApp, Telegram — це хмарний месенджер із безперебійною синхронізацією, що дозволяє отримувати доступ до своїх повідомлень одночасно з кількох пристроїв. Також варто зазначити, що Telegram займає 3 місце в топі найпопулярніших месенджерів [18]. Схоже не варто недооцінювати анімовані емодзі, проте дійсно варто замислитися, чому Telegram потрапив в топ, не дивлячись на повну відсутність передових технологій. Хоча Telegram підтримує наскрізне шифрування, воно не впроваджено за промовчанням, а лише в режимі «секретних чатів» з використанням розробленого компанією протоколу MTProto [19]. Має дві версії, перша з яких була розкритикована експертами з безпеки [20]. Зокрема, Telegram заявляє, що він дотримується політики конфіденційності, де він зобов'язується захищати особисті дані користувачів та відмовляється від передачі їх третім сторонам без дозволу [21].

3) Signal

Signal — безкоштовний додаток для обміну миттєвими повідомленнями з відкритим вихідним кодом [22]. Розроблений американською некомерційною

організацією Signal Foundation [23] (раніше відома як – Open Whisper Systems) з криптографістом Моксі Марлінспайком та Браяном Ектоном (співзасновник WhatsApp, CEO Signal Foundation) на чолі. Додаток фінансується за рахунок грантів та пожертвувань.

Шифрування повідомлень базується на використання розробленого власного протоколу Signal Protocol, розробленому криптографістами Тревором Перріном та Моксі Марлінспайком. Цей протокол є відкритим, тому його оригінальна версія та її модифікації широко використовуються для наскрізного шифрування в інших месенджерах, роблячи Signal стандартом де-факто для месенджерів. Signal пропонує зручний інтерфейс та широкий функціонал, а також гарантує стабільність та безпеку обміну повідомленнями в інтернеті.

Відповідно до прийнятої політики конфіденційності [24], Signal розроблено таким чином, щоб ніколи не збирати та не зберігати конфіденційну інформацію.

Signal рекомендований незалежним джерелом інформації про приватність та безпеку в Інтернеті, Privacy Guides, як один з найкращих додатків для безпечного обміну повідомленнями разом із Session [25].

4) Threema

Threema – пропрієтарний месенджер розроблений під девізом «privacy by design» в Швейцарії, де також знаходяться його власні сервери. Threema обраний урядом Швейцарії та швейцарською армією [26] обов’язковим до використання для всіх офіційних комунікацій [27].

Загалом, політика конфіденційності Threema [28] має досить високий рівень захисту особистих даних та приватності користувачів. На відміну від своїх конкурентів, Threema не вимагає номеру телефону чи електронної пошти для використання, що робить його одним із додатків для обміну повідомленнями, що можна назвати безпечним. Однак, ця конфіденційність має свою ціну, і використання Threema є платним.

Threema приєднався до Privacy Pledge [29], де разом з Brave, Tor, Proton та іншими проектами протистоїть широко поширеному капіталізму стеження. Privacy

Pledge показує, що справді існують способи надання онлайн-послуг без порушення конфіденційності користувачів та пропонує стандарти, яких повинен дотримуватися кожен онлайн-сервіс, щоб надати користувачам контроль над своїми даними та належним чином захистити їх конфіденційність.

5) Wire

Платформа для обміну повідомленнями Wire [30] була розроблена компанією Wire Swiss у 2012 році, зокрема співзасновником Skype, з метою надання безпечних комунікаційних послуг для корпоративних клієнтів. Однак, з часом він також став доступний для широкої публіки. Wire вказує на своє відповідність стандартам безпеки ISO, CCPA (The California Consumer Privacy Act), GDPR та SOX (Sarbanes-Oxley Act) і також пропонує архітектуру з принципом Zero Trust, що передбачає, що всі дані, пристрої, програми та користувачі за своєю суттю є незахищеними та мають бути автентифіковані перед доступом.

Wire забезпечує наскрізне шифрування для всіх функцій за допомогою протоколу Proteus, розробленого Wire Swiss на основі Signal Protocol [31].

У політиці конфіденційності Wire [32] зазначено, що розмови користувачів належать їм і тільки, ні Wire, ні будь-якій іншій третій стороні, компанія не продає та не здає в оренду особисті дані чи вміст розмов для реклами третіх сторін, анонімна (гешована) контактна інформація з адресних книг використовуватиметься лише для підключення користувачів на Wire.

6) Session

Session є децентралізованим месенджером, розроблений австралійською некомерційною організацією Oxen Privacy Tech Foundation.

Session працює на власній реалізації Signal Protocol [33] та децентралізований на основі onion-мережі, проте з використанням власної інфраструктури.

У Session є офіційний документ із описом технічних характеристик програми та протоколу [34].

Згідно із політикою конфіденційності [35], засновники Session ставляться дуже серйозно до приватності своїх користувачів і проект розроблено таким чином, щоб не

збирати жодної інформації, яка може бути використана для відстеження користувачів. Реєстрація в Session швидка та проста, без прив’язки до номера телефону, поштової адреси, чи будь-яких інших ідентифікаційних даних. Замість цього, для створення облікового запису користувач генерує та зберігає на своєму пристрої унікальний ідентифікатор, який може використовувати для автентифікації в месенджері.

Таким чином, було описано обрані месенджері згідно інформації, яку вони надають про себе та офіційної документації.

Пізніше буде проведено детальний аналіз обраних месенджерів, їхнього рівня безпеки на відповідність певним критеріям безпеки.

2.1.3 **Визначення основних критеріїв оцінювання безпеки**

Одним із головних завдань цієї роботи є дослідження захищеності обміну миттєвими повідомленнями в різних месенджерах. Для цього потрібно визначити ключові критерії та вимоги щодо безпеки, конфіденційності та анонімності, за якими будуть оцінюватися конкретні месенджери та яким мають відповідати сучасні системи обміну миттєвими повідомленнями.

Сформовано наступний набір критеріїв оцінювання захищеності обміну миттєвими повідомлення в існуючих рішеннях:

- *Застосування надійних криптографічних примітивів.*

Методи забезпечення конфіденційності, цілісності та автентичності даних зосереджені на використанні криптографії.

Використані в програмі обміну повідомленнями криптографічні методи алгоритми формування ключів, симетричного шифрування, гешування для забезпечення аутентифікації та цілісності, повинні бути зафіксовані та пояснені в документації із зазначенням застосованих алгоритмів та параметрів, порядку генерування, використання, відкликання, обміну ключами, тощо. Важливо, щоб використовувані криптографічні примітиви були надійними, не мали відомих вразливостей, і застосовувалися правильно, щоб забезпечити ефективний захист даних.

- *Впровадження наскрізного шифрування за промовчанням*

Наскрізне шифрування – криптографічний метод захисту повідомлення між учасниками, при якому лише вони можуть розшифрувати повідомлення. За такого підходу, що в своїй основі використовує гібридну криптографію, повідомлення шифруються на пристрої користувача і розшифровуються лише на пристрої отримувача, без можливості прочитати їх на проміжному сервері чи іншій третій стороні і, таким чином, забезпечується безпека спілкування, високий рівень конфіденційності та захисту від перехоплення повідомлень. Оскільки повідомлення шифруються та розшифровуються виключно на кінцевих пристроях користувачів, тому і *приватні ключі мають генеруватися та зберігатися на кінцевих пристроях,* а не на централізованому сервері, який, належить потенційно ненадійній третій стороні, інакше компанія матиме доступ до ключів шифрування, а отже і до вмісту повідомлень.

Реалізація наскрізного шифрування є поширеною та необхідною, проте недостатньою вимогою для будь-яких застосунків обміну миттєвими повідомленнями, які позиціонують себе як безпечні. Важливо враховувати, чи активовано цей параметр за промовчанням. Наприклад, в Telegram наскрізне шифрування за промовчанням доступне тільки в режимі секретного чату, а не в звичайних чатах. У сучасному світі немає і не має бути причин робити наскрізне шифрування необов’язковим!

Крім того, також важливо розуміти, які криптографічні алгоритми використовуються, їх надійність і чи вдало впроваджено.

- *Підтримка Perfect Forward Secrecy*

Perfect Forward Secrecy (PFS) — це властивість криптографічних протоколів, відповідно до якої, повідомлення мають бути зашифровані за допомогою унікальних регулярно новозгенерованих ключів (точніше сеансових ключів) для окремих транзакцій або коротких періодів часу замість постійного використання одного й того самого ключа. У випадку компрометації ключа, це не обов’язково скомпрометує

попередні повідомлення, адже вони були зашифровані за допомогою іншого унікального ключа шифрування.

PFS зазвичай використовується в захищених системах обміну повідомленнями, щоб запобігти довгостроковим спостереженням за комунікаційним вмістом. Тому це необхідний критерій для довгострокових та конфіденційних комунікацій.

- *Використання TLS/Noise для шифрування мережевого трафіку*

Важливо аби мережевий трафік між програмою та серверами чату був зашифрований, що може бути здійснене за допомогою TLS або Noise. Зокрема, Noise Protocol Framework, представлений Тревором Перріном [36], розроблений для встановлення конфіденційних каналів між двома сторонами в різних прикладних сценаріях.

- *Використання certificate pinning*

Certificate pinning - це механізм безпеки, який дозволяє забезпечити довірче з'єднання між клієнтом і сервером. В основі цього механізму лежить використання конкретного SSL/TLS сертифіката, який "закріплюється" (pinning) в коді клієнтської програми.

У випадку використання certificate pinning, додаток не довіряє стандартному механізму перевірки сертифікатів SSL/TLS у кореневому сховищі. Замість цього, програма довіряє лише сертифікатам, які зберігаються в її власному "закріпленому" сховищі, і не приймає з'єднання, якщо сертифікат не відповідає закріпленому. Немає необхідності встановлювати додатковий сертифікат на пристрій користувача.

Цей механізм забезпечує захист від атак типу MITM, коли зловмисник перехоплює з'єднання та підроблює сертифікат, що використовується для шифрування. Крім того, він дозволяє програмі бути впевненою, що вона спілкується лише з офіційним сервером, що підвищує рівень безпеки та надійності системи.

- *Попереднє шифрування повідомлень при хмарному резервному копіюванні*

Деякі програми під час резервного копіювання в хмару повідомлень не шифрують їх попередньо, або ж зберігають копію ключа. Успішна атака на хмарну інфраструктуру може призвести до витоку конфіденційних даних. Враховується ще

те, що інформація про те, чи дійсно резервна копія зашифрована, не є загальнодоступною у всіх месенджерах.

Наприклад, WhatsApp використовує Google Drive або iCloud для резервного копіювання даних медіа та чатів, проте резервні копії медіафайлів і повідомлень в iCloud не захищено наскрізним шифруванням WhatsApp [37].

На відміну від WhatsApp, Telegram не дозволяє користувачам створювати резервні копії чатів у хмарному сервісі тому, що Telegram і є хмарною платформою, яка автоматично зберігає всі ваші дані на власних хмарних серверах. Таким чином, чати не будуть видалені у разі зміни пристрою без створення резервної копії, проте тільки секретні чати захищені наскрізним шифруванням.

Орієнтований на захист конфіденційності користувачів, Signal не зберігає жодних даних на своїх серверах або сторонніх хмарних службах, натомість створюючи резервні копії на пристрої користувача.

Відносно деяких месенджерах оцінити відповідність даної вимоги неможливо з тієї причини, що вони просто не підтримують хмарні сервіси.

- *Відсутність збору даних і метаданих*

Хоча багато програм для обміну повідомленнями використовують наскрізне шифрування, це забезпечує захист конфіденційності лише контенту повідомлень і не заважає можливому серверу збирати іншу цікаву інформацію про користувачів, таку як загальні контакти або ж регулярність спілкування. До прикладу, залежно від конкретного месенджера, дані, які можуть збиратися, можуть включати, але не обмежуватися, такими відомостями про користувача: ім'я, адреса електронної пошти та номер телефону, тобто загалом дані, які надаються під час реєстрації. Також щоразу, під час надсилання чи відправки повідомлення, все одно можуть збиратися метадані.

Метадані – будь-які дані про дані, яка надсилається та отримується, фактично, це всі дані, окрім змісту самого повідомлення, наприклад, будь-яка доступна

інформація про відправника та одержувача, властивості повідомлення, час відправки, час прочитання, тощо.

Більше того, метадані набагато легше обробляти та доцільніше, ніж фактичний вміст, якому вони відповідають. Едвард Сноуден висловився про це так [38]: «Metadata is extraordinarily intrusive. As an analyst, I would prefer to be looking at metadata than looking at content because it's quicker and easier, and it doesn't lie».

Детальніше, дані, які збирає месенджер, зокрема метадані – поведінково-характерні дані, включають:

- *Технічна інформація про використовуваний пристрій*, його тип, назва, модель, IMEI та інші подробиці;
- номер телефону, електронна пошта;
- дані про ОС: архітектура, версія й інше;
- місцезнаходження: *IP-адреса* і геолокація пристрою;
- інформацію про дату та час повідомлення – *timestamps*;
- дані DNS-сервера;
- cookie (дані аутентифікації);

Збір та обробка такої інформації можуть порушувати приватність користувачів та створювати певні ризики для безпеки користувачів та їх даних. Додатки також можуть потребувати доступ до файлової системи пристрою, абонентської книги, функціоналу та ресурсів, наприклад камери, мікрофону і батареї. Хоча вбудована система безпеки ОС Android може забороняти несанкціоноване виконання тих чи інших операцій, однак немає можливості заборонити месенджеру зберігати і синхронізувати те, що потрапляє до його каналу.

Систематично збираючи інформацію, яка є набагато детальнішою та точнішою, ніж та, яка базується лише на вмісті повідомлень, та поєднуючи їх із даними з інших служб можна сформувати повноцінний профіль користувача, що може використовуватися далі в рекламних цілях.

Ці дані та метадані можуть збиратися і для інших цілей, включаючи:

- для побудови власних інформаційних систем, нейронних зв’язків, штучного інтелекту та машинного навчання на основі профілів та шаблонів (інтересів, дій, логіки), що у майбутньому стануть складовими повної автоматизації;
- для покращення сервісу, враховуючи спектри ідей, інтереси, вподобання, коло знайомств користувачів;
- для співпраці з різними рекламними і комерційними компаніями, інформаційно-аналітичними агенціями, яким потрібні аналітичні дані з різних куточків світу (прикладом може бути відомий скандал із Cambridge Analytica [39]);
- для співпраці з державними органами і спеціальними службами, щоби попередити можливі злочини, теракти, катастрофи, інциденти або інші дії, які можуть бути загрозою національній та суспільній безпеці.

Так, майже всі програми збирають дані про своїх користувачів, але існує тонка грань між оптимізацією та порушенням конфіденційності. Принаймні збиратися мають лише дані, необхідні для роботи служби, і вони ніколи не мають зберігатися довше, ніж технічно необхідно. Проблемою є те, що деякі месенджери збирають дані про користувачів без їхнього належного повідомлення та згоди, що порушує принципи приватності та конфіденційності. Крім того, збирається значно більше даних, ніж необхідно для надання основної послуги – обміну повідомленнями.

Бізнес-моделі деяких компаній засновані на зборі інформації про користувачів, що чудово підкреслюють вислів про те, що якщо ви не платите за товар, знач ви і є товар. Наприклад, відповідно до політики конфіденційності WhatsApp [15], Meta та відомі дочірні компанії збирають велику кількість метаданих, які можуть бути використані в рекламних цілях: «Будучи частиною Meta-компаній, WhatsApp отримує інформацію від інших Meta-компаній і ділиться інформацією з ними. Ми можемо використовувати інформацію, яку ми отримуємо від них, і вони можуть використовувати інформацію, якою ми ділимося з ними. Це включає показ відповідних пропозицій і реклами в продуктах компанії Meta».

Так, одними з важливих аспектів безпеки програм обміну повідомленнями, у якому потрібно бути впевненим, є те, які дані та метадані збираються, як

обробляються і захищаються, і хто має доступ. Додатково необхідно деталізувати критерій збору даних, розглянувши також чи виконується збір користувацьких даних та/або метаданих на рівні додатку та/або на рівні компанії для різних описаних цілей, в тому числі, чи надсилаються дані та/або метадані можливій холдинговій компанії та/або третім сторонам, зокрема спецслужбам.

- *Відмова від передачі даних/метаданих клієнтів третім особам*

Критерій пов'язаний зі збором даних і цілей, для яких вони збираються. Месенджер не повинен збирати, зберігати і передавати персональні дані своїх користувачів третім особам без належної їх згоди. Третіми сторони, у даному випадку, можуть бути холдингові компанії, державні органи, спецслужби, рекламодавці, тощо. Цей критерій є важливим для забезпечення конфіденційності та приватності користувачів, а також для забезпечення довіри до месенджера.

Оцінка відмови від передачі даних третім особам може бути складною, оскільки месенджери не завжди стверджують прямо про передачу даних третім сторонам. Однак, допомогти в оцінці практики щодо передачі даних третім особам можуть допомогти політики конфіденційності, звіти про прозорість, незалежні аудити, репутація месенджерів, а також дослідження організацій з питань конфіденційності та приватності користувачів. Наприклад, у 2021 році FOIA-запит від американської комерційної організації Property of the People виявив навчальний документ ФБР, який узагальнює рівень доступу правоохоронних органів США до різноманітних служб обміну повідомленнями із вказівками щодо типу інформації, яку можна отримати та поради по юридичним процесам, які потрібно пройти [40].

- *Шифрування/гешування персональних даних/метаданих*

Розуміння метаданих має важливе значення для розуміння безпеки, конфіденційності та анонімності обміну повідомленнями.

Збір метаданих месенджерами є необхідним для забезпечення функціональності програми, наприклад, для відображення списку контактів або відслідковування доставки повідомлень. Проте, збір метаданих може також

представляти загрозу конфіденційності користувача, якщо ці дані використовуються без його згоди або надто широко поширюються.

Щоб захистити метадані, месенджер може використовувати різні технології, такі як шифрування або гешування.

Шифрування метаданих означає, що дані зашифровуються таким чином, що їх можуть розшифрувати тільки користувачі, які мають потрібні ключі або паролі. Це дозволяє зберегти конфіденційність метаданих, що збираються месенджером.

Гешування метаданих означає, що дані перетворюються на геш, який неможливо відновити у вихідні дані. Це забезпечує анонімність користувача, оскільки неможливо пов'язати геш з його особою.

У кожному випадку використання шифрування або гешування метаданих залежить від потреб користувачів та політики безпеки месенджера. Однак, в будь-якому випадку, месенджер повинен розробити механізми захисту метаданих та забезпечити їх відповідну реалізацію.

Також зібрані персональні дані (номер мобільного телефону, адреса електронної пошти, список контактів тощо) мають гешуватися. Якщо дані гешуються, компаніям їх неможливо прочитати, оскільки номеру телефону, наприклад, йому надається унікальне, незворотне представлення (геш). Цей метод можна використовувати для захисту списків контактів: замість завантаження списку, безпечніше завантажити геш кожного контакту і, якщо один із контактів має ту саму програму, гешований номер телефону користувача у контактів збігатиметься з його гешованим номером телефону на серверах компанії.

Насправді компаніям не потрібно мати будь-яку інформацію про особу для обміну повідомленнями, організація анонімної реєстрації є практичним способом.

- *Анонімна реєстрація*

Критерій оцінює можливість користувача створити анонімний обліковий запис без вимоги надання у якості унікальних ідентифікаторів таких персональних даних як номер телефону та адреса електронної пошти, що можуть бути використані для

ідентифікації його особистості. Чим більше даних потрібно під час реєстрації, тим менша анонімність.

Обхідні шляхи реєстрації із використанням «анонімної» поштової адреси або номером телефону не враховуються тому, що навіть їх можна відстежити і загалом це не вирішує проблему, яка закладена в дизайні системи. Філософія «Privacy by Design» вимагає від розробників додатків і сервісів враховувати приватність користувачів з самого початку процесу розробки, найкращий спосіб знизити ризики, пов'язані з конфіденційністю, — це насамперед не створювати їх. Розробники повинні докладати зусиль для створення систем, які забезпечують захист персональних даних користувачів і недопущення їхнього неправомірного використання.

Аби не повідомляти жодних даних та використовувати додаток користувач має бути унікально ідентифікованим сервером, а отже, для того, щоб програма працювала, кожному користувачеві має бути призначено якийсь випадковий ідентифікатор. В багатьох службах не потрібно нічого, крім вибору імені користувача.

Допускається варіант із гешуванням персональних даних, в таких випадках компанія їх не читає. Геш-функції — це незворотні функції одностороннього шифрування, які можуть надати кожному номеру мобільного телефону чи адресі електронної пошти унікальне значення, яке, по суті, є тарабарщиною. Номер мобільного телефону або адреса електронної пошти гешується на пристрої, а потім завантажується на сервер каталогу. У свою чергу, кожен, хто використовує додаток, має геші всіх своїх контактів, обчислені на своєму пристрої, а потім завантажені на сервер. Якщо два геші збігаються, тоді сервер каталогів знає, що у вашого контакту встановлено програму, не знаючи вашої (або їх) електронної адреси чи номера мобільного телефону.

- *Надійний механізм аутентифікації – 2FA*

Безпека месенджера також залежить від системи аутентифікації користувачів. Підтримка двофакторної автентифікації є важливим додатковим елементом безпеки. Месенджери, що пропонують 2FA, є більш безпечними, оскільки забезпечують

додатковий рівень захисту облікового запису користувача від несанкціонованого доступу.

- *Підтримка повідомлень, що самознищуються*

Описує можливість відправки повідомлень, що автоматично видаляються з пристрою одержувача через певний період часу. Насправді, це мало додає конфіденційності, оскільки можна просто робити скріншоти повідомлень. Проте автоматичне видалення певних частин розмови може бути корисним.

- *Відкритість всього вихідного коду і перевірка на відповідність*

Існує тверде переконання, що відкритість і прозорість створюють більш безпечні засоби. Відкритість всього вихідного коду ПЗ разом із можливою серверною частиною - це один з найважливіших критеріїв оцінювання безпеки програмного забезпечення. Відкритий вихідний код сам по собі не може гарантувати безпеку даних користувача, але він безумовно сприяє цій умові.

Прозоро кажучи про те, як працює їхня система, розробники надають можливість незалежним дослідникам перевірити код на вразливості, підтвердити чи спростувати певний рівень безпеки, будь-хто може створити систему, яку він сам не зможе зламати. Організації, які вибирають відкритий вихідний код свого коду, демонструють добросовісні зусилля для підвищення довіри між ними та користувачем.

Для перевірки того, що версія доступної до завантаження програми дійсно відповідає оригінальному вихідному коду, можна використовувати відтворювані збірки, що, таким чином гарантуючи, що до програм не було внесено шкідливих змін.

- *Проходження незалежних аудитів безпеки*

При вибору месенджеру важливо звернути увагу також на регулярне проходження незалежних аудитів безпеки, щоб підтвердити або спростувати якість безпеки додатків. Лише у разі успішного проходження відкритих аудитів безпеки, можна вже хоч якось стверджувати, що вислови про наскрізне шифрування і всі інші характеристики не є маркетинговим ходом.

- *Звітування про прозорість*

Звіт про прозорість - це важливий критерій, який демонструє дотримання принципу відкритості та відповідальність компанії перед клієнтами та суспільством. У звіті про прозорість компанія повинна надати детальну інформацію про те, як вона збирає та використовує дані своїх клієнтів, а також як вона реагує на запити урядових органів, описуючи типи запитів, отриманих від урядів, звітуючи скільки запитів було зроблено, тощо.

- *Джерела фінансування*

Це важливо, оскільки, як кажуть, «гроші говорять». Якщо компанія чи особа, яка стоїть за грошима, ймовірно, мають причини не захищати конфіденційність клієнтів, це важливо знати. Це може свідчити про те, що компанія діє не так, як вона каже або передумали, коли вони залучили достатньо клієнтів, на яких вони можуть заробляти гроші.

Наприклад, якщо месенджер фінансується виключно рекламою, це може бути показником того, що компанія пріоритетно спрямовує зусилля на збільшення кількості користувачів і певну монетизацію, а не на захист приватності. З іншого боку, якщо месенджер фінансується від підписки або прямих пожертвувань користувачів, це може свідчити про те, що компанія пріоритетно спрямовує зусилля на забезпечення приватності та безпеки користувачів. Тому, джерела фінансування можуть бути важливим критерієм при виборі месенджера, залежно від особистих пріоритетів користувача.

- *Юрисдикція компанії та інфраструктури*

Юрисдикція компанії та інфраструктури є важливим критерієм, який визначає те, які закони та правила діють для компанії та її інфраструктури із серверами, а також як вони реагують на запити на доступ до користувацької інформації від урядів та правоохоронних органів.

Це важливо, оскільки компанії, що базуються в різних країнах, можуть мати різні правові стандарти, що стосуються зберігання, обробки, захисту даних

користувачів. Багато країн використовують широкі механізми стеження або мають тісні стосунки з компаніями, коли справа доходить до отримання доступу до даних.

Наприклад, Five Eyes – це спільна розвідувальна програма, яка була створена між Австралією, Канадою, Новою Зеландією, Великою Британією та Сполученими Штатами Америки в 1946 році. Ці п'ять країн утворили союз з метою обміну розвідувальною інформацією, зокрема в області кібербезпеки та боротьби з тероризмом. Організація Five Eyes займається збором і аналізом різноманітної інформації, включаючи комунікації, перехоплення сигналів, зображення, документи та інші дані, які можуть бути корисними в області національної безпеки. Однак, програму можна покритикувати за недостатню прозорість та можливу порушення приватності користувачів в Інтернеті.

- *Загальна позиція компанії щодо конфіденційності клієнтів*

Вказує на те, як серйозно компанія ставиться до захисту конфіденційної інформації своїх клієнтів. Важливо, щоб компанії не просто говорили про свої зобов'язання забезпечення безпеки, але й насправді дотримувалися цих зобов'язань і приділяли цьому велику увагу.

Загальна позиція компанії щодо конфіденційності клієнтів може бути визначена через її політику конфіденційності, процедури збереження і захисту даних, відповідність різних стандартів безпеки, включаючи GDPR, тощо. Також можна дослідити історії компанії щодо витоків даних, порушення безпеки, рівень прозорості щодо звітування про подібні інциденти та співпрацю зі спецслужбами.

Транспарентність – це якість, що означає чесність, відкритість та прозорість в діяльності, ділових відносинах та прийнятті рішень. У контексті конфіденційності та захисту даних, транспарентність означає, що компанії відкрито надають інформацію про те, які дані збираються, як вони використовуються та як вони захищені від несанкціонованого доступу. Це також може включати детальну інформацію про джерела фінансування, юрисдикцію компанії та інфраструктуру, стандарти конфіденційності та безпеки, які використовуються компанією, тощо.

Транспарентність допомагає збільшити довіру користувачів до компаній та зменшити ризик порушення конфіденційності та безпеки даних.

Для проведення оцінки рівня безпеки обміну повідомленнями в різних месенджерах на основі визначених критеріїв для кращої ілюстрації результатів оцінки попередньо розділити рівні відповідності. таблиця, що розміщена у Додатку Б наводить оцінку критеріїв, які визначено важливими в контексті захищеного обміну миттєвими повідомленнями в додатках. Кожен критерій оцінюється за трьома рівнями безпеки: низький, середній та високий. Додатково критерії було прокатегоризовано на такі домени критеріїв:

1) пов'язані з використанням криптографічних методів для забезпечення безпеки обміну повідомленнями;

2) пов'язані зі збором та передачею даних та метаданих під час обміну повідомленнями;

3) пов'язані з транспарентністю та позицією компанії стосовно безпеки та конфіденційності в обміні миттєвими повідомленнями.

Таким чином, було визначено та описано критерії оцінювання захищеності обміну миттєвими повідомленнями та безпеки месенджерів загалом. Критерії важливі при формулюванні вимог до систем обміну миттєвими повідомленнями та виборі месенджеру за необхідним рівнем безпеки відповідно до моделі загроз кожного.

2.1.4 **Порівняння поширених месенджерів за визначеними критеріями**

Уявлення про те, як кожен месенджер відповідає визначеним критеріям щодо криптографічної складової та рівням безпеки можна отримати за допомогою таблиці 1.1. Дана таблиця слугує зручним засобом порівняння між різними месенджерами та відображає результати оцінки їхньої безпеки. Оцінювання відповідності визначено на основі джерел визначених в описаній методиці.

Таблиця 1.1

Оцінювання відповідності месенджерів визначеним критеріям щодо криптографічної складової та відповідним рівням безпеки

Критерій	WhatsApp	Telegram	Signal	Threema	Wire	Session
I. Криптографічна складова						
Використання надійних криптографічних примітивів	Curve25519 / AES-256 / HMAC-SHA256	RSA 2048 / AES 256 / SHA-256	Curve25519 / AES-256 / HMAC-SHA256	Curve25519 256 / XSalsa20 256 / Poly1305-AES 128	Curve25519 / ChaCha20 / HMAC-SHA256	X25519 / XSalsa20 256 / Poly1305
Впровадження E2EE за промовчанням (протокол)	Так, за умови що пристрій підтримує (Signal Protocol)	Частково (MTProto)	Так (Signal Protocol)	Так (NaCl)	Так (Proteus (Signal-based))	Так Signal-based
Створення та зберігання приватних ключів на кінцевому пристрої	Так	Так	Так	Так	Так	Так
Підтримка PFS	Так	Ні (сеансові ключі змінюються після 100 разів використання)	Так	Так	Так	Ні
Попереднє шифрування повідомлень перед cloud backup	iOS: Так Android: Ні		N/A, виключено з iCloud/ iTunes & Android систем бекапу	Yes	N/A, виключено з iCloud/ iTunes & Android систем бекапу	N/A, виключено з iCloud/ iTunes & Android систем бекапу
Шифрування метаданих	Ні	Ні	Так	Так	Переважно	Так

продовження табл.1.1

Критерій	WhatsApp	Telegram	Signal	Threema	Wire	Session

Гешування персональних даних	Ні	Ні	Переважно	Так	Переважно	N/A
Використання TLS/Noise для шифрування мережевого трафіку	Так	Ні	Так	Так	Так	Так
Використання certificate pinning			Так	Так	Так	Так
Надійна система (2FA)	Так	Так	Так	Так	Так	Так

Уявлення про те, як кожен месенджер відповідає визначеним критеріям щодо збору та передачі даних та метаданих та рівням безпеки можна отримати за допомогою таблиці 1.2.

Таблиця 1.2

Оцінювання відповідності месенджерів визначеним критеріям щодо збору та передачі даних та метаданих та відповідним рівням безпеки

Критерій	**WhatsApp**	**Telegram**	**Signal**	**Threema**	**Wire**	**Session**
II. Збір та передача даних та метаданих						
Збір компанією даних користувачів	Так	Так	Ні	Ні	Ні	Ні
Збір додатком даних користувачів	Контактна інформація ідентифікатори покупки фінансова інформація місцезнаходження дані про використання	Контактна інформація контакти ідентифіка-тори	Контакт. інформа-ція	Контактна інформація ідентифі-катори	Контактна інформація ідентифікатори дані про використа-ння	Ні

продовження табл.1.2

Критерій	WhatsApp	Telegram	Signal	Threema	Wire	Session
Передача даних користувачів та/або метаданих третім особам	Так, Meta	Так	Мінімальний (обов'язковий номер мобільного телефону, надісланий третій стороні для реєстрації та відновлення)	Немає (додатковий номер мобільного телефону, надісланий третій стороні для реєстрації)	Так	Ні
Причетність до передачі даних користувачів спецслужбам	Ні	Ні	Ні	Ні	Ні	Ні
Логування позначок часу (timestamp) та IP-адрес	Ні	Ні	Ні	Ні	Трішки	Ні
Анонімна реєстрація	Ні	Ні	Ні	Так	Ні	Так
Повідомлення, що самознищуються	Так	Так	Так	Ні	Так	Так

Таблиця. 1.3 представляє те, як кожен месенджер відповідає визначеним критеріям щодо транспарентності та позиції компанії та рівням безпеки.

Таблиця 1.3

Оцінювання відповідності месенджерів визначеним критеріям щодо транспарентності та позиції компанії та відповідним рівням безпеки

Критерій	WhatsApp	Telegram	Signal	Threema	Wire	Session
IV. Транспарентність та позиція компанії						
Повна відкритість вихідного коду	Ні	Ні (лише клієнтська частина та API)	Так	Ні (тільки додаток)	Так	Так
Відтворювані збірки для перевірки	Ні	Так, iOS Android	Тільки для Andoid	Тільки для Andoid	Ні	Ні

продовження табл.1.3

Критерій	**WhatsApp**	**Telegram**	**Signal**	**Threema**	**Wire**	**Session**
Нещодавній публічний аудит безпеки	Ні	Так (2015)	Так (2016)	Так (2020)	Так (2018)	Так (2021)
Звіт про прозорість	Так	Ні	Так	Так	Так	Так
Джерела фінансування	Facebook	Павло Дуров	Freedom of the Press Foundation / the Knight Foundation / the Shuttleworth Foundation / the Open Technology Fund / Signal Foundation	Користувачі / Afinum Management AG	Janus Friis / Iconical / Zeta Holdings Luxembourg / Morpheus Ventures	LAG Foundation Ltd
Юрисдикція компанії	США	США / Великобританія / Беліз / ОАЕ	США	Швейцарія	США / Швейцарія	Австралія
Юрисдикція інфраструктури	США (невідомо щодо інших)	Великобританія, Сінгапур, США, Фінляндія	США	Швейцарія	ЄС	Повідомлення: у всьому світі (використовуються децентралізовані сервери) Додатки: централізований сервер у Канаді
Загальна позиція компанії щодо конфіденційності клієнтів	Погана	Погана	Хороша	Хороша	Хороша	Хороша

2.1.5 **Оцінка результатів дослідження**

На основі проведеного порівняння поширених месенджерів за визначеними критеріями необхідно підсумувати результати дослідження з метою визначення рівня захищеності обміну миттєвими повідомленнями в розглянутих месенджерах, недоліків та можливих покращень кожного месенджера з точки зору захищеності обміну миттєвими повідомленнями. Загальний результат оцінки результатів дослідження представлено в табл.1.4:

Таблиця 1.4

Загальний результат оцінювання відповідності месенджерів визначеним критеріям та відповідним рівням безпеки

Критерій	**WhatsApp**	**Telegram**	**Signal**	**Threema**	**Wire**	**Session**
Загальна оцінка						
Чи рекомендується програма для захищеного обміну миттєвими повідомленнями?	Ні	Ні	Так	Так	Так	Так

Додаток WhatsApp не рекомендується з таких основних причин:

1) Facebook пов'язували з партнерством з Національним агентством з безпеки США (NSA) у зв'язку зі злиттям інформації від Едварда Сноудена. Це створює сумніви щодо приватності та безпеки даних користувачів.

2) Бізнес-модель Facebook базується на зборі інформації користувачів, що зменшує стимул для Facebook забезпечувати дійсно надійний захист інформації користувачів.

3) Facebook має погану репутацію щодо цифрової приватності, оскільки їхня бізнес-модель ґрунтується на зборі інформації про користувачів.

4) Метадані не зашифровані, що означає, що часові мітки, місцезнаходження, відправник, отримувач тощо не зашифровані. Це надає розвідувальним агентствам велику кількість інформації для визначення того, що люди роблять за їхнім місцезнаходженням, з ким вони спілкуються та коли вони це роблять.

5) Дані користувачів (номери телефонів, контактна інформація) не захищені, тому Facebook має доступ до номерів телефонів та імен контактів та їхніх номерів телефонів.

6) Додаток та сервери не мають відповідного доступного відкритого коду, тому ніхто не знає якість коду, чи реалізовано шифрування правильно та чи існують серйозні вразливості.

7) Не проводилося перевірки коду та незалежного аналізу безпеки, тому ми маємо вірити лише словам Facebook. Ніхто не може перевірити свої роботи сам, включаючи Facebook.

8) Повідомлення можуть бути прочитані Facebook, якщо вони відзначені як "образливі".

Що стосується месенджеру Telegram, він також не рекомендується з таких причин:

1) Використання власного криптографічного алгоритму не є найкращою практикою, а реалізація шифрування в Telegram була піддана критиці криптографами.

2) Шифрування не включено за замовчуванням, що означає, що користувачі повинні вручну ввімкнути шифрування для своїх розмов.

3) Дані користувачів (номери телефонів, контактна інформація) не захищені, тому Telegram має доступ до номерів телефонів та імен контактів та їхніх номерів телефонів.

4) Юрисдикція Telegram є неясною, що перебуває поза моєю експертизою.

Незважаючи на те, що багато статей стверджують інше, WhatsApp та Telegram не повинні вважатися безпечними. Враховуючи ці фактори, рекомендується звернути увагу на інші месенджери, які мають кращу репутацію щодо захищеності і приватності даних користувачів. Інші з проаналізованих месенджерів, такі як Signal, Threema, Wire та Session, вже виявилися вартою уваги і пропонують покращені функції безпеки та приватності.

У наведеному нижче списку пропонуються рекомендації щодо покращень для кожного з досліджуваних месенджерів, які можуть сприяти поліпшенню рівня захищеності та приватності користувачів:

Рекомендовані можливі покращення для Signal:

1) Видалити обов'язкову вимогу до користувачів реєструватися з мобільним номером.

2) Надати більш повні незалежні оцінки безпеки та приватності.

Рекомендовані можливі покращення для Threema:

1) Зробити відкритим кодом API та серверне ПЗ.

2) Надати більш повні незалежні оцінки безпеки та приватності.

Рекомендовані можливі покращення для Wire:

1) Додатково обмежити зберігання та логування метаданих.

2) Надати більш повні незалежні оцінки безпеки та приватності.

Рекомендовані можливі покращення для Session:

1) Впровадити PFS на рівні кінцевого шифрування.

2) Надати більш повні незалежні оцінки безпеки та приватності.

Загалом, результати оцінки рівня захищеності та відповідності кожного месенджера встановленим критеріям безпеки, виявлені недоліки та можливості покращення надають цінну інформацію для користувачів та розробників систем обміну миттєвими повідомленнями. Однак важливо зазначити, що жоден месенджер не є абсолютно безпечним, і користувачі повинні завжди бути уважними та свідомими щодо захисту своїх особистих даних. Проведений комплексний огляд сприяє вибору найбільш підходящого месенджера залежно від потреб користувача та вимог до безпеки обміну повідомленнями відповідно до моделі загроз.

Висновки за розділом 1

В ході першого розділу було досліджено рівень захищеності обміну миттєвими повідомленнями в існуючих месенджерах. На початку було проведено аналіз

основних понять та розглянуто становлення систем обміну миттєвими повідомленнями. Це дозволило отримати загальне уявлення про еволюцію цих систем та їх вплив на комунікацію.

Здійснений вибір месенджерів для дослідження, враховуючи їх популярність, поширеність серед користувачів, позиціонування щодо захищеності та заявлений рівень безпеки. Це забезпечило репрезентативність дослідження та дозволило отримати об'єктивну оцінку безпеки цих месенджерів.

Проведено огляд існуючих месенджерів та проведено їх порівняльний аналіз рівня забезпечення безпеки обміну миттєвими повідомленнями з використанням методології, яка полягала в оцінці безпеки месенджерів на основі визначених критеріїв оцінювання безпеки обміну миттєвими повідомленнями. Ці критерії включають криптографічну складову, збір та передачу даних та метаданих, функціональність та безпеку взаємної ідентифікації користувачів, а також транспарентність та позицію компанії. За результатами огляду складено таблиці із порівняльними характеристиками захисних механізмів у зазначених месенджерах.

Порівняння поширених месенджерів за визначеними критеріями дало змогу оцінити їх відповідність вимогам безпеки та визначити переваги, недоліки, можливі покращення кожного месенджера з точки зору захищеності обміну миттєвими повідомленнями.

В ході проведеного аналізу було виявлено, що у процесі користування месенджером за різних причин можуть виникати загрози для цілісності, доступності та конфіденційності інформації у листуванні.

Цей розділ є важливим в контексті подальшого розроблення програмного модуля, оскільки надає специфікацію вимог до безпеки обміну миттєвими повідомленнями та визначає основні критерії для оцінки месенджерів.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ

2.1 Постановка задачі та специфікація основних вимог

У першому розділі було досліджено забезпечення захищеності обміну миттєвими повідомленнями в існуючих рішеннях, що, зокрема, дозволило визначити потенційні загрози та проблеми, що стосуються безпеки обміну миттєвими повідомленнями, які необхідно враховувати при розробці програмного модуля. Далі розглянемо архітектурні та функціональні аспекти програмного модуля для забезпечення захищеного обміну миттєвими повідомленнями.

Головною метою даної роботи є розробка зручної у використанні системи для захищеного миттєвого обміну повідомленнями.

Визначена мета підкреслює наступні основні цілі розробки:

- *Безпека даних:* забезпечити надійний захист повідомлень, їх конфіденційності та цілісності.
- *Конфіденційність користувачів:* захист конфіденційності користувачів, завдяки використанню мінімальної кількості персональних даних.
- *Зручність використання:* зручна навігація та простий функціонал.

На основі цілей та попереднього огляду проблем можна сформувати специфікацію вимог до програмного модуля забезпечення захищеного обміну миттєвими повідомленнями.

Специфікація вимог до ПЗ є важливим етапом у проектуванні програмного продукту, оскільки вона визначає основні параметри і вимоги, на яких буде базуватися подальша робота з розробки, тестування та впровадження ПЗ. Вона допомагає забезпечити зрозумілість та однозначність вимог, а також є основою для оцінки виконання вимог та контролю якості розроблюваного ПЗ.

Даний проект мав вимоги, які були розділені на наступні основні категорії:

1) Функціональні вимоги

Функціональні вимоги описують очікувану поведінку системи, функціональність, яку користувачі очікують від програмного забезпечення.

Функціональні вимоги, що визначені для програмного модуля забезпечення захищеного обміну миттєвими повідомленнями, включають наступні елементи:

- Надання графічного інтерфейсу користувача (GUI).
- Реалізація реєстрації та автентифікації в системі.
- Приватні чати (важливо зауважити, що увага в даному проекті приділяється саме особистому обміну повідомленнями, а не груповому обміну, оскільки ці дві ситуації вимагають різних підходів та розглядаються окремо).
- Обмін текстовими повідомленнями (можливість надсилання та отримання текстових повідомлень з використанням E2E- шифруванням з усіма властивостями безпеки, що характерні для існуючих систем).

2) Нефункціональні вимоги

Нефункціональні вимоги включають в себе широкий спектр характеристик системи, що не пов'язані безпосередньо з її функціональністю.

Нефункціональні вимоги, що стосуються програмного модуля забезпечення захищеного обміну миттєвими повідомленнями, включають наступні аспекти:

- Система повинна використовувати сучасні криптографічні алгоритми для захищеного обміну повідомленнями.
- Система повинна забезпечувати швидку доставку повідомлень та відсутність значних затримок.
- Механізм реєстрації з обмеженою кількістю персональних даних.
- Інтерфейс месенджера повинен бути інтуїтивно зрозумілим, простим та зручним для використання.
- Підтримка різних платформ (система повинна підтримувати різні платформи, такі як мобільні пристрої (Android, iOS) та комп'ютери (Windows, MacOS)).

Дані вимоги є основою для подальшого проектування програмного модуля та визначення архітектури та функціональності системи.

2.2 Основні компоненти програмного рішення

Оскільки однією з визначених вимог є підтримка різних платформ, то обрано підхід реалізації системи у формі вебдодатку. Вебдодаток дозволяє користувачам отримувати доступ до системи з різних платформ та операційних систем, використовуючи веббраузер. Це забезпечує широку сумісність і зручність використання для користувачів, оскільки вони можуть отримати доступ до системи без необхідності встановлювати окремий додаток на свої пристрої. Також оновлення вебдодатків здійснюється на серверному рівні, що означає, що користувачам не потрібно встановлювати оновлення на свої пристрої. Вони автоматично отримують доступ до оновленої версії при наступному відвідуванні вебдодатку. А завдяки веб-технологіям та стандартам, розробка вебдодатків стає відносно простою із застосуванням різних фреймворків та бібліотек, що дозволяє швидко розширювати та вдосконалювати функціонал додатку.

Обрано, що розроблюваний вебдодаток базується на клієнт-серверній архітектурі, де клієнтська сторона взаємодіє з сервером для обміну даними та виконання різних операцій відповідно до вимог. Використання цієї архітектури дозволяє розділити відповідальності та забезпечити масштабованість та ефективність системи.

Таким чином, система обміну миттєвими повідомленнями складається з таких компонентів:

- *Клієнтська сторона додатку*, що надає весь необхідний інтерфейс, який доступний користувачам через веббраузер, та функціональність системи безпосередньо кінцевому користувачеві аби здійснювати обмін миттєвими повідомленнями; відповідає за збір та відображення даних, взаємодію з користувачем та відправку запитів на сервер для отримання або оновлення даних.

- *Серверна сторона додатку,* що відповідає за обробку запитів від клієнтів, збереження та управління даними, дозволяє користувачам встановлювати чати та з'єднуватися один з одним та обмінюватися повідомленнями.
- *База даних* використовується для зберігання інформації про користувачів, повідомлення, інші відомості, необхідні для роботи системи.

2.3 Процес взаємодії клієнтської та серверної частини

Відповідно до вимог, необхідно реалізувати систему надсилання та отримання даних у реальному часі через мережу між клієнтам

Нехай в програмі клієнтська та серверна сторони взаємодіють за допомогою HTTP-протоколу (рисунок 2.1). Клієнтська сторона відправляє HTTP-запити на сервер, а сервер обробляє ці запити та надсилає HTTP-відповіді з необхідною інформацією. Коли клієнт хоче переглянути вебсторінку, він генерує HTTP-запит на сервер. Сам сервер не може надсилати дані клієнту, якщо клієнт попередньо не зробив запит.

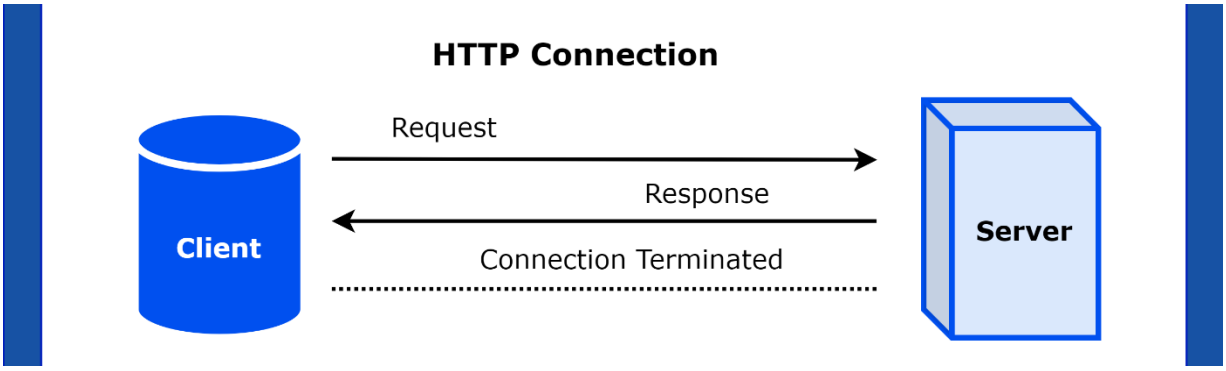

Рисунок 2.1 – Взаємодія клієнта та сервер через HTTP-з'єднання

Проте традиційна модель "запит-відповідь" непридатна для реалізації частини із обміном миттєвими повідомленнями. Для цього буде потрібно встановлювати з'єднання TCP щоразу, коли дані надсилаються на сервер. Будучи одностороннім протоколом синхронного зв'язку, це може призвести до великих накладних витрат під час створення та руйнування TCP-з'єднання кожного разу, коли повідомлення

надсилається в реальному часі. Також клієнт повинен буде щосекунди опитувати сервер щодо нових повідомлень, що призводить до неефективності, а для програми в режимі реального часу затримка є центральною проблемою.

У розробці системи миттєвого обміну повідомленнями в реальному часі за допомогою мережі передачі даних між клієнтами, найефективнішим рішенням є використання концепції WebSocket, що дозволяє безперервно надсилати або передавати ці дані у вже відкритому постійному з'єднанні між клієнтом та сервером (рисунок 2.2).

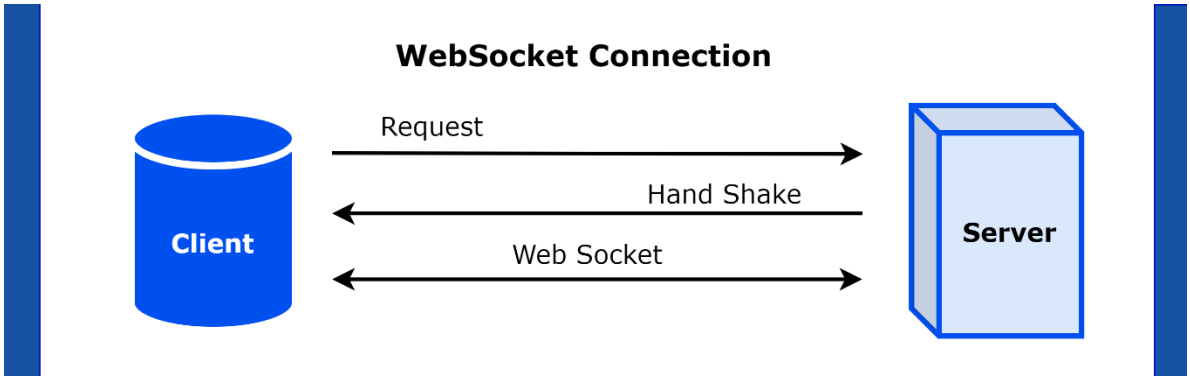

Рисунок 2.2 – Взаємодія клієнта та сервер через WebSocket-з'єднання

WebSocket – це мережевий протокол, що дозволяє двонаправлену передачу даних між браузерами та серверами в режимі реального часу. IETF стандартизував протокол як RFC 6455 [41]. Як і HTTP, він розташований на прикладному рівні мережевої моделі OSI. Для WebSocket лише перше підключення (фаза рукостискання) виконується за допомогою методу запит-відповідь протоколу HTTP. Після встановлення з'єднання для взаємодії використовується двонаправлений зв'язок, між клієнтом і сервером, дозволяючи одночасно передавати повідомлення між ними, що, зокрема, досягається шляхом дозволу серверу передавати вміст клієнту без попереднього отримання запиту від клієнта. Коли з'єднання з користувачем розривається (тобто користувач від'єднується), сервер миттєво дізнається, що користувач вийшов з мережі.

Socket.io – це бібліотека, яка побудована на основі протоколу WebSocket і спрощує роботу з WebSockets. Вона забезпечує зручний механізм для створення

двонаправленого зв'язку в реальному часі між клієнтом та сервером на основі подій, що дозволяє надсилати та отримувати повідомлення у режимі реального часу.

Socket.io забезпечує підтримку кімнат (rooms), які дозволяють групувати користувачі. Така концепція використовуються для організації приватного чату (рисунок 2.3).

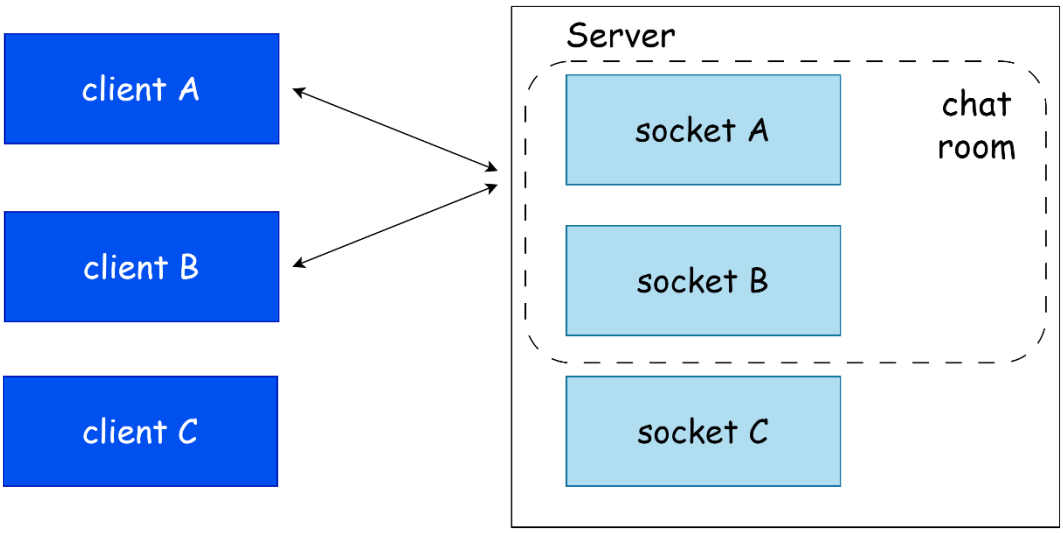

Рисунок 2.3 – Схема організації чату двох користувачів для обміну миттєвими повідомленнями з використанням кімнат

Користувачі, які знаходяться в одній кімнаті, можуть обмінюватися повідомленнями між собою. Це особливо корисно для реалізації приватних чатів або групових чатів, де користувачі можуть взаємодіяти тільки з тими, хто перебуває в тій самій кімнаті.

2.4 Використання наскрізного шифрування

Наскрізне шифрування є ключовою складовою розробки месенджера з урахуванням безпеки та конфіденційності.

Наскрізне шифрування (end-to-end encryption – E2EE) – криптографічний метод захисту повідомлення між учасниками, при якому лише вони можуть розшифрувати повідомлення. Це – механізм, за допомогою якого повідомлення захищені таким

чином, що жодна третя сторона не може отримати до них доступ або змінити їх. Тільки відправник і одержувач можуть отримати доступ до цих повідомлень.

Хоча існує незліченна кількість реалізацій E2EE, усі вони використовують загальний дизайн E2EE, коли для кожного повідомлення отримують інший ключ шифрування та автентифікують його (перевіряють джерело) за допомогою криптографії з відкритим ключем. Загальна схема реалізації наскрізного шифрування представлена на рисунок 2.4:

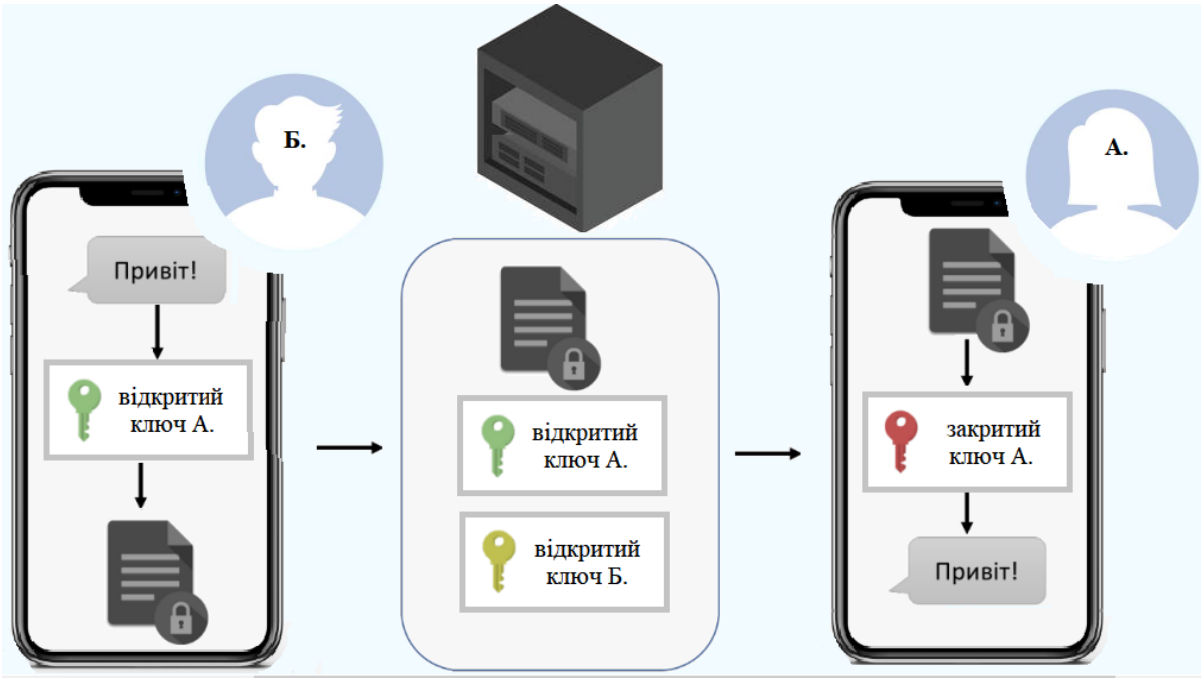

Рисунок 2.4 – Загальна схема наскрізного шифрування

Наскрізне шифрування відбувається на рівні пристрою. Кожен користувач програми чату має власну пару відкритих і закритих ключів. Перед тим, як повідомлення або файли покидають кінцевий пристрій, вони шифруються за допомогою відкритого ключа, який доступний для кожного. Одержувач може розшифрувати повідомлення лише за допомогою відповідного закритого ключа відправника.

Більшість поширених застосунків підтримують наскрізне шифрування (проте не завжди за промовчанням) на основі існуючих протоколів, що розглянуто в [11].

Найбільш безпечним і найсучаснішим дизайном E2EE є протокол Signal з відкритим кодом від Open Whisper Systems. Даний протокол є де-факто сучасним

стандартом наскрізного шифрування, тому його обрано до впровадження в систему обміну повідомленнями.

Протокол Signal складається з наступних основних кроків: реєстрація, налаштування сесії, обмін повідомленнями. Розглянемо етапи дещо детальніше:

1) Реєстрація

Сторони, які хочуть безпечно обмінюватися повідомленнями одна з одною, обидві реєструються на центральному сервері системи обміну повідомленнями, використовуючи деяку особисту інформацію, якою зазвичай в більшості рішеннях є їхній номер телефону.

Потім генерується наступний набір пар ключів, а їхні відкриті ключі завантажуються на центральний сервер:

1. *Пара ключів ідентифікації (Identity Key Pair):* довгострокова пара ключів Curve25519, що представляє «ідентичність» сторони. Називається просто ідентифікаційним ключем (Identity Key).

2. *Підписаний попередній ключ (Signed Pre Key)* : середньострокова пара ключів Curve25519, підписана ідентифікаційним ключем Identity Key і змінювана на періодичній основі.

3. *Одноразові попередні ключі (One-Time Pre Keys):* черга пар ключів Curve25519 для одноразового використання, створена під час встановлення та оновлюється за вимогою.

Вони генеруються на клієнтській стороні, для кожного пристрою та зберігаються лкально на пристрої (передається лише публічна частина). Це означає, що вони є ефемерними, оскільки існують лише протягом короткого часу або існують, поки програму встановлено. Усі надіслані повідомлення криптографічно підписуються за допомогою ключа ідентифікації Identity Key, а підписи всіх вхідних повідомлень перевіряються за допомогою автентифікації в протоколі Signal.

2) Налаштування сеансу сесії

Протокол Signal дозволяє двом сторонам обмінюватися зашифрованими повідомленнями на основі ефемерного спільного секретного ключа, початкового

кореневого ключа (*initial root key*). Цей спільний секретний ключ погоджується обома сторонами за допомогою протоколу узгодження ключів.

Реалізації використовують Extended Triple Diffie-Hellman (X3DH), де сторона-ініціатор запитує та отримує набір відкритих ключів іншої сторони від центрального сервера та використовує їх для налаштування сеансу шляхом отримання початкового кореневого ключа з них. Не будемо вдаватися в подробиці X3DH, оскільки це не має значення для нашого проекту.

3) Обмін повідомленнями

Після встановлення початкового кореневого ключа (initial root key) сторони використовують алгоритм Double Ratchet для надсилання та отримання наскрізно зашифрованих повідомлень.

В алгоритмі Double Ratchet кожне надіслане повідомлення шифрується за допомогою свіжого ключа шифрування повідомлення, отриманого з ефемерного загального секретного ключа, який періодично оновлюється. Це означає, що попередні ключі шифрування повідомлень не можна обчислити з пізніших, що забезпечує високий ступінь прямої секретності (forward secrecy). Алгоритм можна візуалізувати двома «храповиками» (ratchets), що насправді є ланцюжками функції формування ключів (KDF).

Використання E2E-протоколу Signal забезпечує такі властивості безпеки [42]:

- конфіденційність: тільки відправник і одержувач можуть прочитати повідомлення;
- цілісність даних: гарантія точності та послідовності повідомлення;
- аутентифікація даних: підтвердження джерела повідомлення (тобто відправника).
- forward secrecy: компрометація поточного повідомлення не ставить під загрозу минулі повідомлення;
- post-compromise security: компрометація поточного повідомлення не ставить під загрозу майбутні повідомлення;

Відповідно до протоколу Signal, взаємодія абонентів та серверів відбувається у ненадійному середовищі, що створює ризик атаки MITM. Це дозволяє зловмиснику замінити ключі під час передачі та видати себе за будь-яку зі сторін комунікації. Часто ця проблема вирішується за допомогою цифрової сертифікації та PKI.

2.5 Процес ідентифікації та аутентифікації з використанням PGP

Як елемент PKI для реалізації ідентифікації та аутентифікації користувачів обрано до впровадження систему Pretty Good Privacy (PGP). Кожен користувач системи обміну повідомленнями має довготривалу пару ключів PGP для універсального представлення своєї особи.

Система, що використовується, демонструє модель довіри, яка центрується навколо користувача [43] що представлена на рисунок 2.5. У цій моделі кожен користувач самостійно приймає рішення про те, на які сертифікати він спирається та які сертифікати вважає недостатньо надійними. Це рішення залежить від різних факторів, але початковий набір довірених ключів користувача часто включає відкриті ключі членів родини, друзів або колег, з якими користувач особисто знайомий.

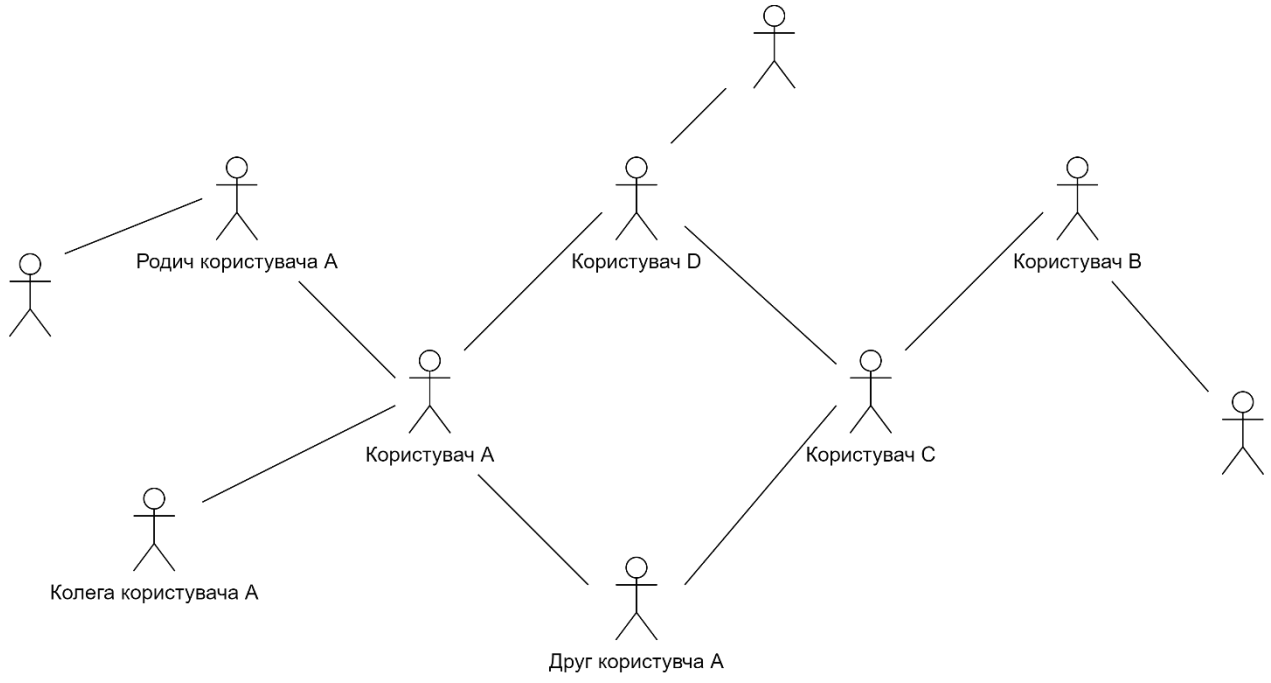

Рисунок 2.5 – Модель довіри, сконцентрованої навколо користувача

Однак, через залежність від дій та рішень користувачів, модель довіри, що центрується навколо користувача, може бути застосована тільки в обмеженому колі спільноти і не є ефективною для загальної публіки, де більшість користувачів не мають достатнього рівня знань про безпеку та технологію PKI. Більше того, така модель не підходить для галузей, де потрібний контроль над тим, з ким користувачі взаємодіють і кому вони довіряють, таких як корпоративний, фінансовий або урядовий сектори.

Також у подальших дослідженнях можливе використання PGP для удосконалення протоколу Signal. Інтеграція в протокол Signal із використанням ключів PGP замість ключів ідентифікації може допомогти усунути наявні вразливості протоколу Signal.

Проста ідентифікації та автентифікація буде виконана, коли користувач підключається до сервера, щоб підтвердити особу користувача (тобто його ключ PGP). Підключившись користувач має надати своє ім’я, ключі PGP та пароль (passphrase)

Описаний алгоритм аутентифікації можна представити наступною схемою, зображеною на рисунок 2.6.

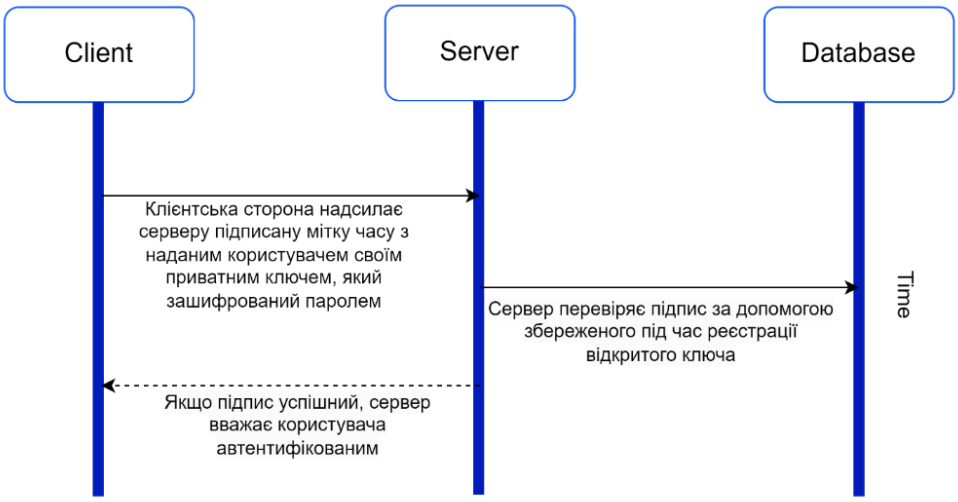

Рисунок 2.6 – Схема аутентифікації на основі перевірки підпису

Для автентифікації користувач має підписати мітку часу своїм приватним ключем PGP, а сервер перевірить його підпис за використання публічного ключа

користувача, що був збережений на сервері під час реєстрації. Якщо підпис перевірений успішно, сервер вважає користувача автентифікованим та надає доступ до системи чату. Інакше користувач повідомляється про помилку у введених даних.

Визначено, що термін дії підпису закінчується через 1 хвилину (якщо мітка часу старша за цей час), тож якщо підпис пізніше буде скомпрометовано зловмисником, його неможливо буде використати для імітації користувача та порушення автентифікації.

Обробка реєстрації та аутентифікації вимагає деякої логіки із перевіркою та збереженням користувацьких даних.

2.6 **Вибір технологічного стеку додатку**

Для розробки додатку варто ретельно розглянути і вибрати відповідні засоби та технології:

1. Середовище розробки

З урахуванням різних факторів, таких як зручність, продуктивність та підтримка потрібних технологій, було обрано використання Microsoft Visual Studio Code (VS Code) як основне середовище розробки. VS Code - це інтегроване середовище розробки, яке набуло великої популярності серед розробників завдяки своїм багатофункціональним можливостям та розширенням. Одним з основних переваг VS Code є його зручний інтерфейс, який сприяє зосередженості на розробці та підвищує продуктивність.

2. Мова програмування та фреймворки:

Для реалізації програмного засобу обміну миттєвими повідомленнями була обрана мова програмування JavaScript (JS). JS є широко використовуваною мовою у веброзробці, і вона пропонує ряд переваг, які роблять її ідеальним вибором для даного проекту. Оскільки JS є стандартом веброзробки і дозволяє створювати динамічні та інтерактивні вебдодатки безпосередньо в браузері користувача, це забезпечує широку доступність програмного засобу на різних пристроях та операційних системах. JS

також має велику кількість розширень, бібліотек і фреймворків, що спрощують розробку вебдодатків.

Зважаючи на визначені вимоги, для розробки серверної частини програмного засобу захищеного обміну миттєвими повідомленнями, обрано використання фреймворків Node.js та Express.js.

Express.js є популярним фреймворком для розробки вебдодатків на платформі Node.js. Він надає простий та ефективний спосіб створення серверної частини додатку, а також обробки HTTP-запитів та відповідей та маршрутизації, дозволяє швидко налаштувати сервер та почати розробку без зайвих складнощів.

Використання мови програмування JS разом з такими технологіями як HTML та CSS надає можливість створити базовий графічний інтерфейс клієнтської частини вебдодатку. HTML (HyperText Markup Language) використовується для структуризації контенту сторінки, тобто для створення розмітки CSS (Cascading Style Sheets) – для оформлення і стилізації. JavaScript: для реалізації динамічного інтерфейсу користувача та взаємодії з сервером.дозволяє додати динаміку та взаємодію до графічного інтерфейсу. З його допомогою можна створювати анімації, обробляти події, змінювати вміст сторінки без перезавантаження та багато іншого.

3. Бібліотеки для використання

Для створення системи, що відповідає описаним вимогам та реалізує весь описаний функціонал необхідно використати:

- Express.js: популярний фреймворк для побудови вебдодатків на Node.js, дозволяє створювати сервер, визначати маршрутизацію, обробляти запити та відповіді.
- Socket.IO [44]: бібліотека для реалізації двонаправленої WebSocket-комунікації між клієнтом і сервером з використанням подій. Має дві частини: бібліотеку на стороні клієнта, яка працює в браузері, і бібліотеку на стороні сервера для Node.js.

- Libsignal-protocol.js [45]: JavaScript-реалізація Signal Protocol для забезпечення безпеки комунікації міє клієнтами, яка надає функціональність для генерації ключів, шифрування та розшифрування повідомлень, обмін ключами тощо.
- Sqlite3: пакет для роботи з базою даних SQLite у Node.js [46].
- OpenPGP.js — це реалізація протоколу OpenPGP на JavaScript [47], яка використовується для виконання всіх дій, пов'язаних із PGP, необхідних програмі. OpenPGP - відкритий стандарт програмного забезпечення для шифрування PGP.
- Cookie-parser: middleware для роботи з куками (схоже на сесії) у Express.js.
- Jsonwebtoken: бібліотека для роботи з JWT (JSON Web Tokens), що використовуються для аутентифікації та авторизації.

Комбінація цих бібліотек дозволяє побудувати безпечну комунікаційну систему, де повідомлення шифруються на клієнтській стороні і передаються через сервер, а потім розшифровуються на отримувачі.

Висновки за розділом 2

У даному розділі було розглянуто ключові аспекти проектування системи для захищеного миттєвого обміну повідомленнями.

Розділ починається з постановки задачі, що полягає в розробці програмного модуля для захищеного обміну миттєвими повідомленнями. У цьому контексті були визначені основні вимоги до програмного модуля, включаючи функціональні, та нефункціональні вимоги, які було враховано під час проектування системи. Головними цілями розробки визначено безпеку даних, конфіденційність користувачів і зручність використання. При формуванні функціональних вимог було враховано необхідність графічного інтерфейсу користувача, реєстрації та автентифікації, приватних чатів і обміну повідомленнями. Нефункціональні вимоги, такі як використання сучасних криптографічних алгоритмів, швидка доставка повідомлень, мінімальна кількість персональних даних при реєстрації, інтуїтивно зрозумілий

інтерфейс та підтримка різних платформ, дозволяють забезпечити виконання виділених цілей.

Концепція WebSocket обрано до використання для забезпечення миттєвого обміну повідомленнями в реальному часі. Впровадження наскрізного шифрування з протоколом Signal та ідентифікація користувачів за допомогою PGP-ключів забезпечують високий рівень захисту даних та користувачів.

Загальний результат проектування системи для захищеного миттєвого обміну повідомленнями полягає в проектуванні безпечного і зручного програмного засобу, що дозволяє користувачам захищено обмінюватися миттєвими повідомленнями в режимі реального часу через веббраузер, який забезпечує безпеку їхніх даних та конфіденційність під час спілкування. Також при реалізації програмного модуля у формі вебдодатку забезпечуються необхідні універсальність та зручність використання.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ

3.1 Проектна структура додатку

Усі деталі попереднього розділу були використані для розробки системи обміну захищеного миттєвими повідомленнями – EncryptoChat. Лістинг програмного коду клієнтської та серверної частини розміщено в додатку В.

Додаток має наступну проектну структуру, зображену на рисунок 3.1:

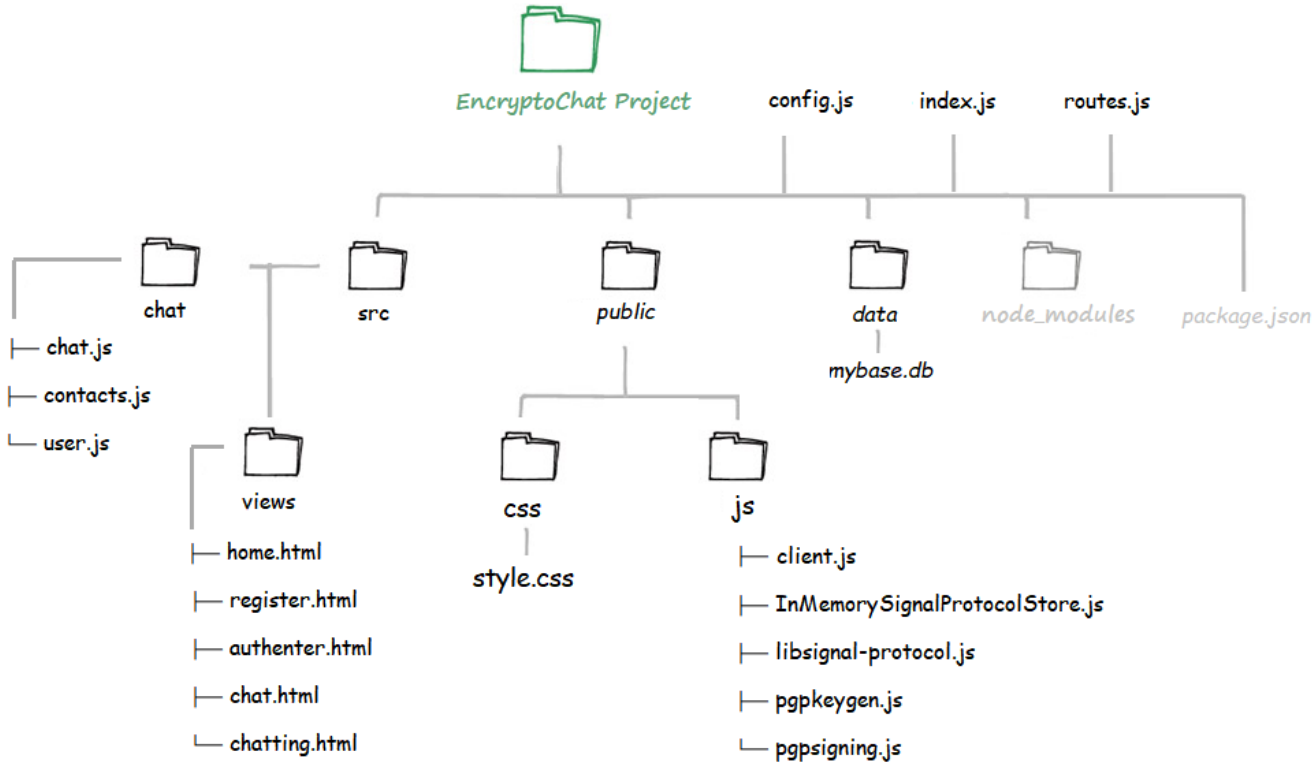

Рисунок 3.1 – Проектна структура розробленого рішення

Ця структура показує організацію файлів та каталогів у проекті. Кожен файл або каталог відповідає певному функціоналу або компоненту проекту:

- *node_modules:* каталог, де зберігаються залежності проекту, які були встановлені за допомогою пакетного менеджера npm.

- *package-lock.json:* файл, який фіксує точні версії всіх пакетів, встановлених у проекті, щоб забезпечити повторюваність встановлення пакетів на різних системах.
- *package.json:* файл, який містить метадані проекту, включаючи назву, версію, авторів, залежності та скрипти, пов'язані з проектом.
- *data/mydbase.db:* файл бази даних, який використовується в проекті.
- *public/css/style.css:* файл стилів CSS для вебсторінок проекту.
- *public/js/client.js:* скрипт, який містить логіку клієнтської сторони проекту.
- *public/js/InMemorySignalProtocolStore.js:* скрипт, який містить клас SignalProtocolStore для зберігання даних протоколу Signal.
- *public/js/libsignal-protocol.js:* бібліотека, яка реалізує протокол Signal.
- *public/js/pgpkeygen.js:* скрипт, що містить логіку генерації ключів PGP.
- *public/js/pgpsigning.js:* скрипт, що містить логіку підписування повідомлень за допомогою PGP.
- *src/chat/chat.js:* скрипт, що містить логіку чату.
- *src/chat/contacts.js:* скрипт, що містить логіку роботи з контактами в чаті.
- *src/chat/user.js:* скрипт, що містить логіку користувача в чаті.
- *src/views/home.html:* HTML-шаблон сторінки домашньої сторінки проекту.
- *src/views/register.html:* HTML-шаблон сторінки реєстрації користувача.
- *src/views/authenter.html:* HTML-шаблон сторінки автентифікації користувача.
- *src/views/chat.html:* HTML-шаблон сторінки чату.
- *src/views/chatting.html:* HTML-шаблон сторінки спілкування в чаті.
- *src/config.js:* містить конфігураційні параметри проекту, зокрема, забезпечує підключення до бази даних
- *src/index.js:* головний JavaScript-файл проекту, який також забезпечує підключення до бази даних

- *src/routes.js:* скрипт, який містить маршрутизаційні правила для серверної частини проекту.

Приведений нижче код (рисунок 3.2) файлу index.js представляє створення та ініціалізацію сервера для застосунку EncryptoChat. Цей код встановлює необхідні зв'язки і налаштовує серверну частину EncryptoChat для прийому та обробки запитів від клієнтів, обміну повідомленнями через Socket.IO та взаємодії з базою даних.

src > JS index.js > ...

```
1   // index.js відповідає за створення сервера
2   const http = require('http')
3   const express = require('express')
4   const cookieParser = require('cookie-parser')
5   const socketio = require('socket.io')
6   const config = require('./config')
7   const router = require('./routes')
8
9   const app = express()
10  const server = http.createServer(app)
11  const io = socketio(server)
12  require('./chat/chat')(io)
13
14  const userModule = require('./chat/user');
15  const contactModule = require('./chat/contacts');
16  const sqlite3 = require('sqlite3').verbose();
17
18  const DB_PATH = "D:\\!KSENNYA\\4курс\\4.2\\!дипломне проектування\\EncryptoChat\\data\\mybase.db";
19  const db = new sqlite3.Database(DB_PATH);
20  // Передача підключення до бази даних в модуль user.js
21  userModule.init(db);
22  //
23  app.use(express.static(config.pubDirPath))
24  app.use(cookieParser())
25  app.use((express.urlencoded({ extended: false })))
26  app.use(router)
27  //
28  server.listen(config.port, () => {
29      console.log('Server is up on port ' + config.port)
30  })
31
```

Рисунок 3.2 – Лістинг коду index.js, представляє створення та ініціалізацію сервера для застосунку EncryptoChat

У наведеному вище лістингу коду створюється HTTP-сервер для застосунку EncryptoChat. Спочатку підключаються необхідні модулі, такі як:

- http – модуль Node.js, що дозволяє створювати HTTP-сервери і обробляти HTTP-запити. Використовуючи цей модуль, ми можемо створити сервер, який слухатиме певний порт і оброблятиме запити від клієнтів.

- express – це популярний фреймворк для розробки вебдодатків на Node.js. Він надає розширені можливості для обробки маршрутів, шаблонів, статичних файлів та інших задач, що пов'язані з розробкою вебсервера.
- cookie-parser – модуль, який допомагає обробляти та парсити HTTP-кукі, що передаються між клієнтом і сервером. Це дозволяє зберігати інформацію про сесію користувача або інші дані, які потрібні для ідентифікації та взаємодії з клієнтом.
- socket.io – це бібліотека, яка надає можливості для реалізації багатокористувацьких додатків з миттєвим обміном повідомлень. Використовуючи Socket.IO, ми можемо створити двостороннє з'єднання між сервером і клієнтом, що дозволяє передавати повідомлення в реальному часі.

Загалом, ці модулі надають нам засоби для створення та керування сервером, обробки HTTP-запитів, роботи з cookie, а також засоби для миттєвого обміну повідомленнями між клієнтами за допомогою технології WebSockets.

Відповідно до коду, здійснюю ініціалізацію Express.js додатку та створюємо HTTP-сервер, використовуючи http.createServer(app), де app - це екземпляр Express.js додатку. Також створюємо об'єкт Socket.IO за допомогою socketio(server), щоб забезпечити миттєвий обмін повідомленнями між клієнтами.

Для нашого EncryptoChat використовуємо модуль chat/chat, який встановлює зв'язок з Socket.IO і налаштовує обробку повідомлень між клієнтами.

Далі, додаємо модулі - userModule і contactModule. Вони відповідають за роботу з користувачами та контактами в чаті. Ми також підключаємо базу даних SQLite за допомогою модуля sqlite3 і ініціалізуємо з'єднання з базою даних у модулі user.

Далі, використовуємо express.static для обслуговування статичних файлів з вказаного шляху config.pubDirPath. Це дозволяє надавати клієнту доступ до ресурсів, таких як HTML, CSS і JavaScript файли.

Також використовуємо cookieParser для обробки кукі, які можуть передаватися між клієнтом і сервером. Це дозволяє зберігати інформацію про сесію користувача.

Для обробки даних форми з POST-запитів використовуємо express.urlencoded. Це дозволяє отримувати дані, які були надіслані з форми на клієнтській стороні.

Маршрутизація запитів відбувається за допомогою router, який містить набір маршрутів і відповідних обробників запитів.

Нарешті, ми запускаємо сервер, вказуючи йому порт, на якому він буде слухати запити. Після запуску серверу виводиться повідомлення про те, що сервер працює і слухає певний порт (рисунок 3.3).

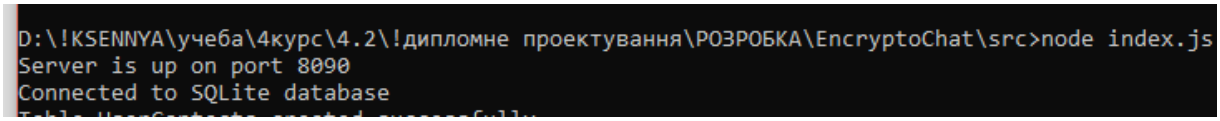

Рисунок 3.3 – Успішний запуск серверу та встановлення з’єднання з базою даних

Після підняття сервера і прослуховування певного порту, користувач може взаємодіяти з вебдодатком за допомогою веббраузера. У результаті (рисунок 3.4) на екрані з'явиться початкова сторінка вашого вебдодатку.

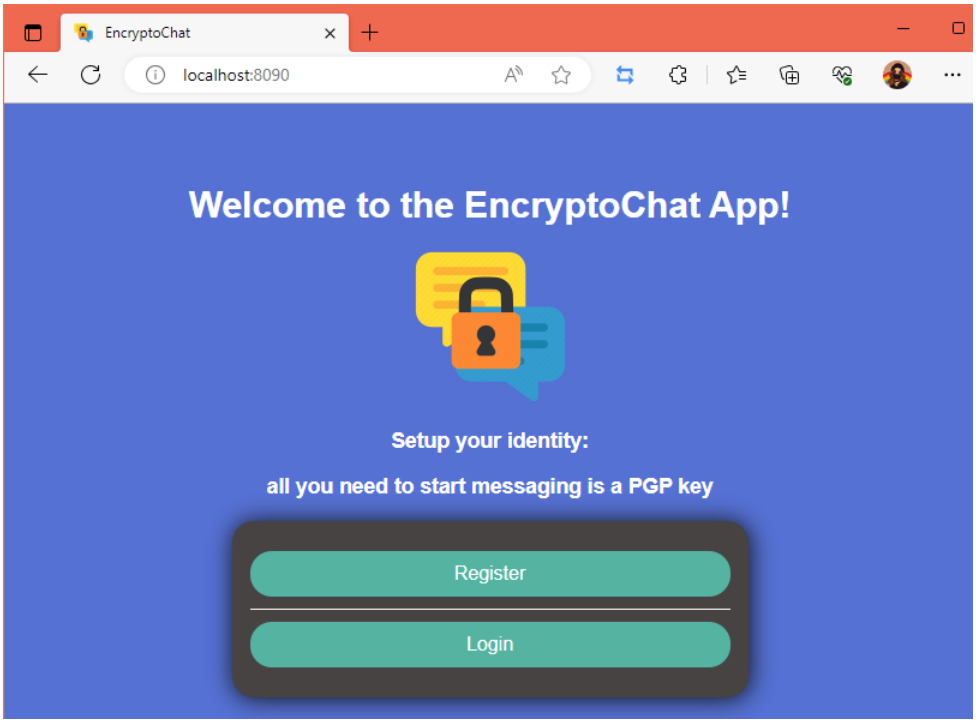

Рисунок 3.4 – Початкова сторінка вебдодатку при взаємодії користувача з вебсервером

Важливо зазначити, що програмний засіб на поточному етапі в межах даної роботи реалізується як прототип на локальному сервері, оскільки розробка прототипу

дозволяє швидко створити, перевірити та представити основний функціонал та інтерфейс додатку перед повноцінною розробкою. Прототип дозволяє зрозуміти, як програма працює на практиці та виявити можливі недоліки або покращення для подальшого процесу розробки. Крім того, розробка прототипу програмного засобу з обмеженим функціоналом дозволяє зосередитись на аспектах захищеності, забезпечення яких і є головною ціллю дослідження.

3.2 Архітектурна модель розробленого рішення

Архітектура розробленого модуля для захищеного обміну миттєвими повідомленнями зображена на рисунок 3.5:

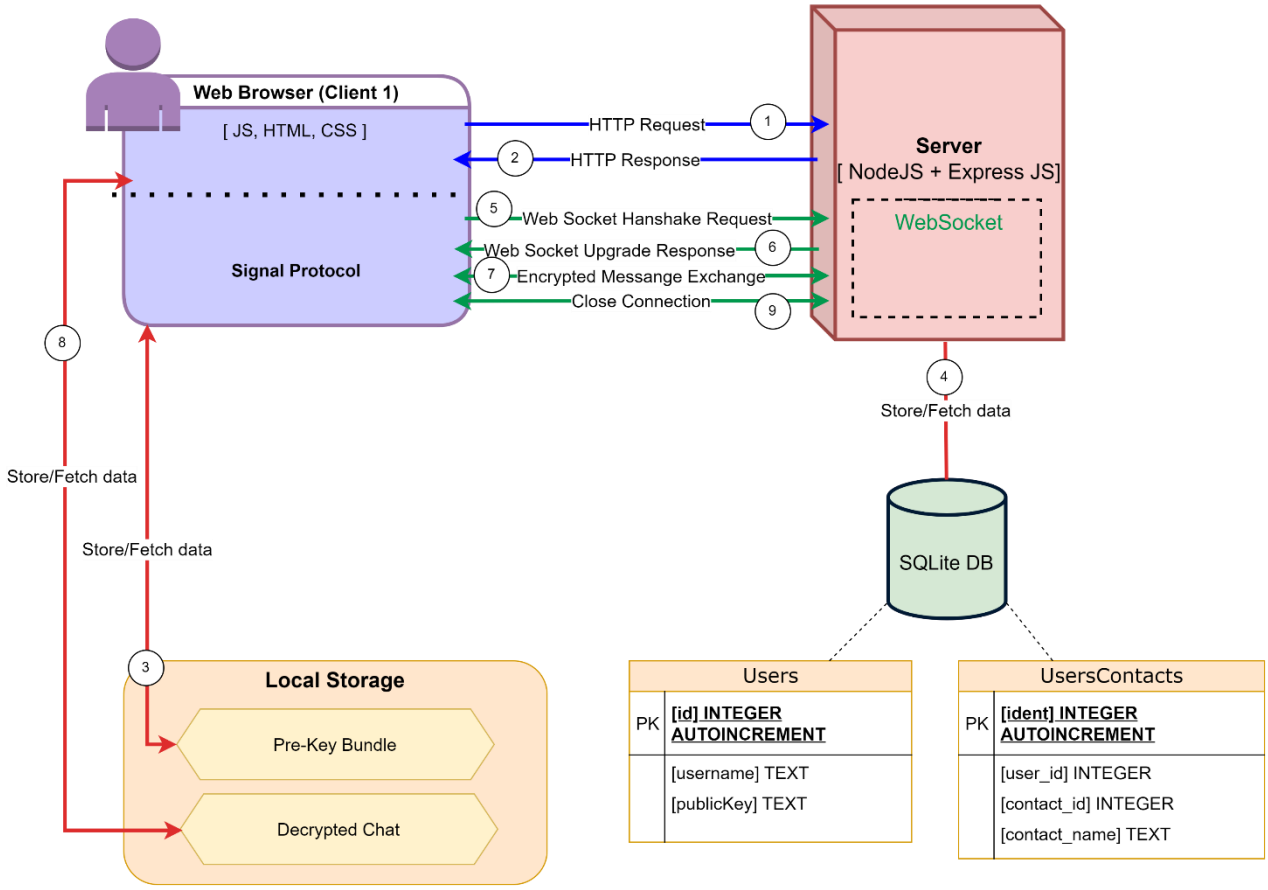

Рисунок 3.5 – Високорівнева діаграма архітектури програмного рішення для обміну миттєвими повідомленнями

Відповідно до наведеної HLD архітектури вище, послідовність дій обміну миттєвими повідомленнями можна описати наступним чином:

Кроки 1 і 2. HTTP-запит-відповідь для входу користувачів, також для додавання контактів.

Крок 3. збереження та отримання попереднього ключа в/з LocalStorage під час входу користувача.

Після підключення кожен клієнт генерує свою ідентичність (ключі реєстрації та ключі ідентичності) і надсилає їх на сервер. Клієнт також генерує ключі передпідписаного ключа (signed prekey) та попереднього ключа (prekey).

*Крок 4. П*еревірка користувачів з файлу бази даних SQLite, перевірка чи можуть розпочати чат, для чого потрібно бути доданими контактами, також виклик /fetch до сервера, щоб отримати всі контакти користувача і вивести на екран

Тобто, користувач може вибрати контакт для спілкування в чаті за обраним ідентифікатором користувача і перейти до приватного чату з ним у разі проходження всіх необхідних перевірок.

Кроки 5 і 6. Запит-відповідь на рукостискання вебсокета для перемикання протоколів для організація приватного чату.

Крок 7. Двосторонній обмін повідомленнями.

Сервер отримує ключі користувачів, після чого розсилає їх іншим клієнтам у кімнаті (крім самого клієнта, який згенерував ключі) Кожен клієнт отримує ключі від інших клієнтів і зберігає їх у своєму локальному сховищі (SignalProtocolStore). Клієнт будує сеанс шифрування (session cipher) для кожного клієнта, з яким він хоче обмінюватись повідомленнями. Для цього використовується сесійний побудовник (SessionBuilder), який отримує ключі іншого клієнта зі сховища і встановлює спільний секретний ключ для шифрування та дешифрування повідомлень.

Далі безпосередньо під час обміну повідомленнями, коли користувач надсилає повідомлення, воно спочатку шифрується за допомогою Signal, точніше за допомогою ключів, збережених у локальному сховищі (InMemorySignalProtocolStore.js), відправляючи зашифровані повідомлення через

сесійний шифр (session cipher), а потім надсилається на сервер за допомогою Web Socket. Користувач отримує зашифровані повідомлення від свого контакту через сокети і може розшифрувати їх за допомогою ключів.

Крок 8. Зберігання та отримання чатів до/з LocalStorage.

Чати у вікні повідомлень (розшифровані) і локальному сховищі (зашифровані) оновлюються новими повідомленнями. Наразі було створено синхронну систему обміну миттєвими повідомленнями, що потребує присутності учасників та не передбачає збереження повідомлень на сервері.

Крок 9. З’єднання Web Socket закривається з боку клієнта або сервера

Основною базою даних, яку використовує проект, є SQLite база даних з файлом mydbase.db. Створено 2 таблиці "Users" та "UserContacts", можна сформувати ER-діаграму, яка відображає зв'язок між цими таблицями. Нижче наведена описана ER-діаграма на рисунок 3.6:

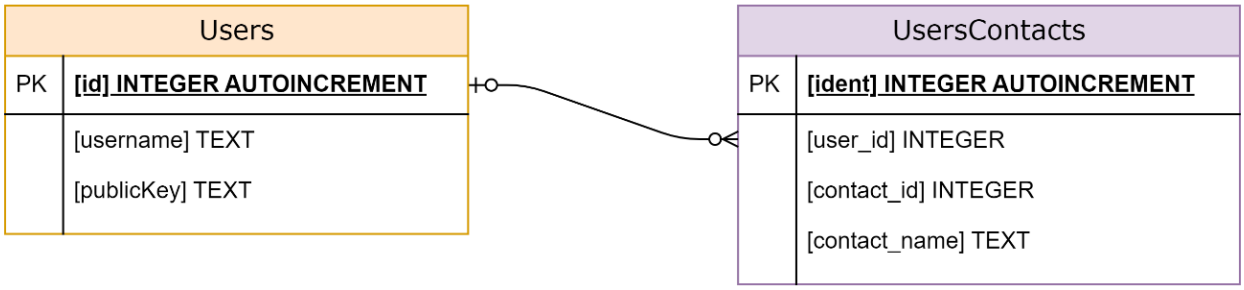

Рисунок 3.6 – ER-діаграма додатку

Таблиця "Users" містить такі поля:

- id: унікальний ідентифікатор користувача (автоматично згенерований);
- username: ім'я користувача, яке використовується під час реєстрації користувача;
- publicKey: публічний PGP ключ користувача.

Таблиця "UserContacts" містить такі поля:

- ident: унікальний ідентифікатор зв’язку користувачів;
- user_id: ідентифікатор користувача (зовнішній ключ, пов'язаний з полем id таблиці "Users");

- contact_id: ідентифікатор контакту, з яким користувач утримує зв'язок (зовнішній ключ, пов'язаний з полем id таблиці "Users");
- contact_name: ім'я контакту.

3.3 Опис користувацького інтерфейсу

Інтерфейс користувача програми – це те, що дозволяє користувачеві взаємодіяти з усією системою. На основі описаних вимог та функціональних можливостей користувачів було розроблено додаток, графічний інтерфейс якого складається з таких компонентів:

1) Компонент «Welcome»

Це перший елемент графічного інтерфейсу користувача, в цілому, представляє собою головну сторінку додатку EncryptoChat, який бачить користувач під час запуску програми. Користувач зайшовши до вебдодатку отримує доступ до home.html. Відповідно до рисунок 3.7, користувачеві пропонується можливість створення облікового запису або увійти до вже існуючого. Залежно від вибору, наступним екраном GUI буде або «Реєстрація», або «Вхід до системи».

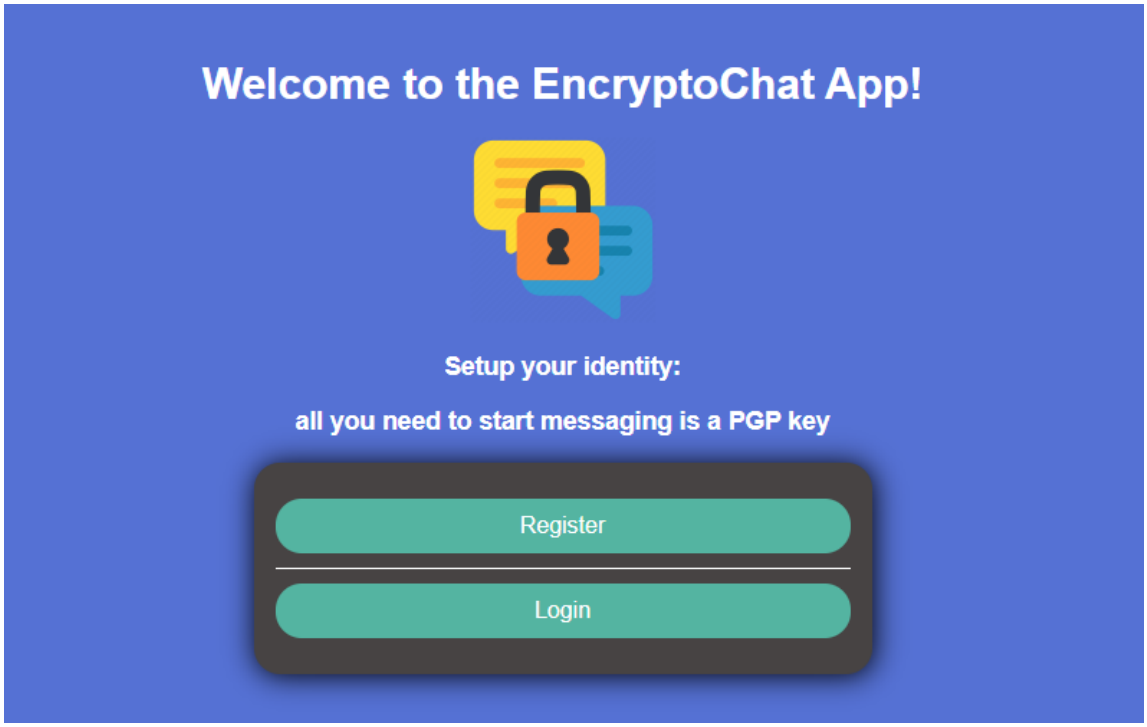

Рисунок 3.7 – Компонент GUI “Welcome”

Якщо користувач новий, він переходить до сторінки реєстрації (register.html), де вводить необхідні облікові дані (логін, пароль і т.д.) і надсилає їх для створення нового облікового запису. Якщо користувач вже має обліковий запис, він вводить свої облікові дані на сторінці авторизації (authenter.html) і надсилає їх для перевірки.

2) Компонент «Реєстрація»

Даний компонент графічного інтерфейсу (рисунок 3.8) відображається, коли користувач вибирає на початковому екрані варіант зі створенням облікового запису. На екрані користувачу пропонується вказати всі деталі, необхідні для створення ключа PGP:

- *Ім'я:* їх ім'я (ім'я, за яким користувачі, з якими вони хочуть спілкуватися, впізнають їх).
- *Парольна фраза:* секретна фраза, яка використовується для захисту секретного ключа PGP-ключа, діє як пароль.

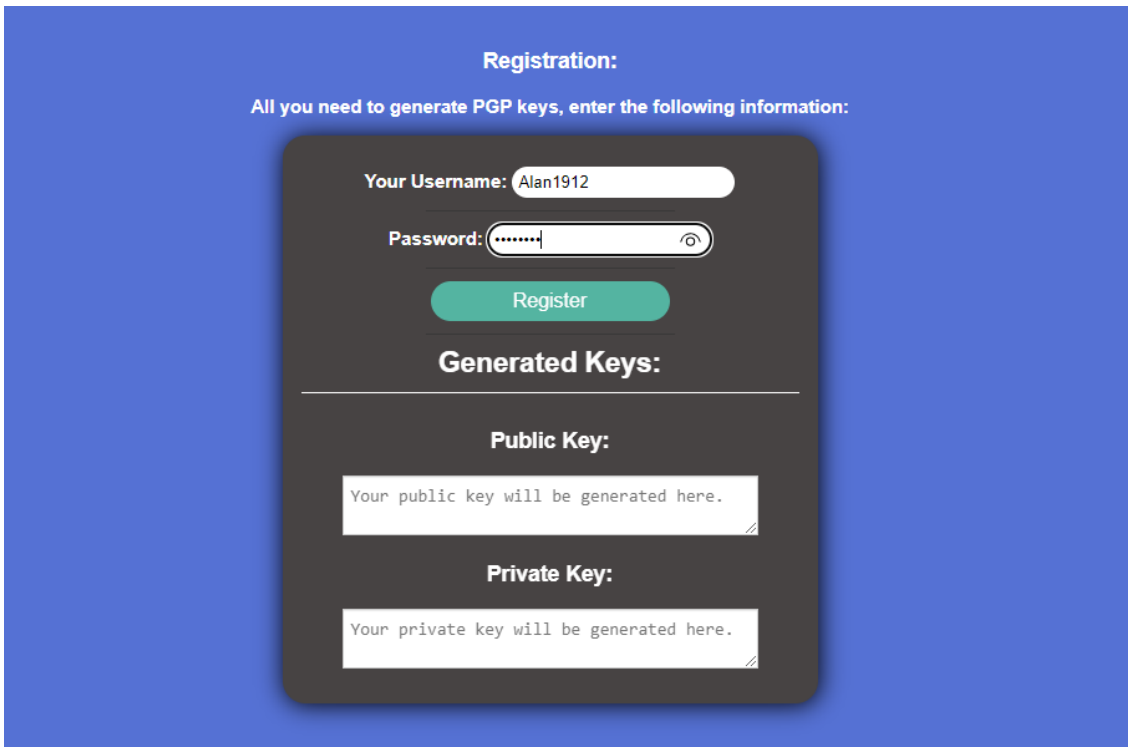

Рисунок 3.8 – Клієнтський інтерфейс для створення нового облікового запису користувача з прикладом заповнення даних

Коли користувач надсилає ці вхідні дані, вони проходять валідацію. Якщо вони недійсні, відображається відповідне повідомлення про помилку. Якщо вони валідні, ключі PGP генеруються з використанням деталей на клієнтській стороні, здійснюється їхній вивід у відповідні поля, після чого робиться відповідний запит для внесення даних користувача до бази даних.

За результатами реєстрації показується відповідне сповіщення як на рисунок 3.9:

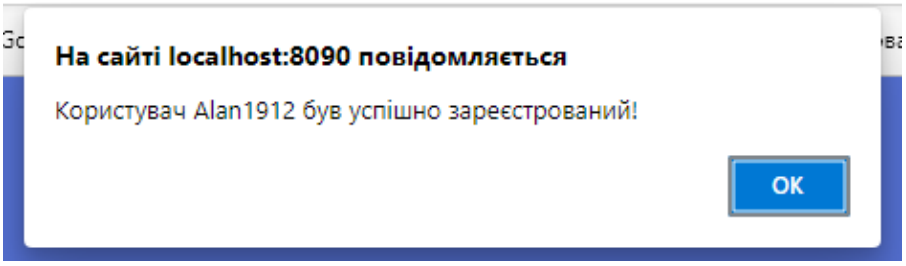

Рисунок 3.9 – Повідомлення про результат реєстрації

Після реєстрації користувач повертається на початковий компонент та має перейти до входу в обліковий запис.

Файл register.html представляє собою HTML-сторінку, яка використовується для реєстрації користувачів у вебдодатку EncryptoChat. На сторінці також підключений JavaScript-файл pgpkeygen.js, який містить скрипти для генерації ключів PGP.

Обробка маршруту для обробки реєстрації та використана функція реєстрації користувача в системі наведено на рисунок 3.10 та рисунок 3.11 відповідно.

```
101  const userModule = require('./chat/user');
102  // Маршрут для обробки реєстрації
103  router.post('/register', (req, res) => {
104    const {username, publicKey} = req.body;
105    // Виклик функції реєстрації з модуля user.js
106    console.log("Router post register.....")
107    userModule.registerUser(username, publicKey);
108    // Відповідь з успішним статусом
109    res.sendStatus(200);
110    //return res.redirect('/chat');
111  });
```

Рисунок 3.10 – Лістинг обробки маршруту для реєстрації користувачів у файлі routes.js

```
function registerUser(username, publicKey) {
    const sql = 'INSERT INTO Users (username, publicKey) VALUES (?, ?)';
    db.run(sql, [username, publicKey], (err) => {
      if (err) {
        console.error('Error registering user:', err);
      } else {
        console.log('User registered successfully');
      }
    });
}
```

Рисунок 3.11 – Лістинг функції реєстрації в файлі user.js

Лістинг частини функції генерації PGP ключів з використанням бібліотеки OpenPGP для нового користувача показано на рисунок 3.12:

```
const {name, password} = req.body;
console.log(name);
console.log(password);
(async () => {
    const keys = await openpgp.generateKey({
        type: 'ecc', // Type of the key, defaults to ECC
        curve: 'curve25519', // ECC curve name, defaults to curve25519
        userIDs: [{ name: name,  }], // you can pass multiple user IDs
        passphrase: password, // protects the private key
        format: 'armored' // output key format, defaults to 'armored' (other options: 'binary'
    });

    console.log(keys.privateKey);     // '-----BEGIN PGP PRIVATE KEY BLOCK ... '
    console.log(keys.publicKey);      // '-----BEGIN PGP PUBLIC KEY BLOCK ... '
    const publicKey1 = await openpgp.readKey({ armoredKey: keys.publicKey});
    console.log("publicKey:", publicKey1);
```

Рисунок 3.12 – Лістинг генерації PGP ключів

Під час генерації закритих і відкритих ключів PGP за допомогою OpenPGP обрано використання еліптичної кривої Ed25519 через продуктивність і малий розмір ключа. Також визначається використання парольної фрази для закритого ключа. На практиці це має бути надійний рандомізований секрет, створений для одноразового використання, проте в даному випадку це визначений користувачем пароль.

3) Компонент «Вхід до системи»

Даний компонент GUI (рисунок 3.13) відображається, коли користувач вибирає увійти до системи на початковому екрані. На екрані буде запропоновано вказати ім’я, існуючі відкритий і закритий PGP-ключі разом із його парольною фразою. Коли користувач надсилає ці деталі, деталі перевіряються. Зокрема здійснюється автентифікація, алгоритм якої було описано в §2.5. Якщо вони недійсні,

відображається відповідне повідомлення про помилку. Якщо деталі правильні, ключі надійно зберігаються всередині, і користувачеві відображається головний екран Messenger.

У цілому, ця сторінка представляє собою форму входу користувача, де він може ввести свої дані та ключі PGP для аутентифікації у вебдодатку EncryptoChat та доступу до функцій чату.

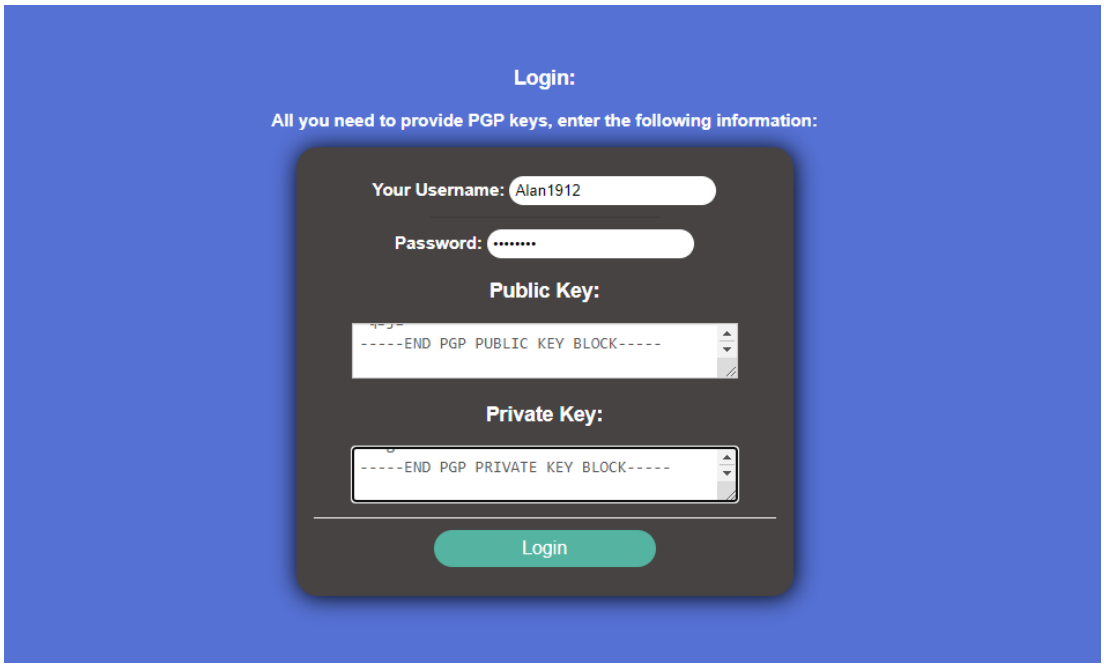

Рисунок 3.13 – Компонент «Вхід до системи»

Перевірка підпису для автентифікації здійснюється з використанням функції pgpVerify(), лістинг якої наведено на рисунок 3.14:

```
149  const openpgp = require('openpgp'),
150    { SIGNATURE_EXPIRE_TIME } = require('../../config')
151
152  // Verifies a pgp signature is valid and not expired and returns the userId
153  exports.pgpVerify = async (publicKeyArmored, timestamp, signature) => {
154    // Fail verification if all params are not supplied
155    if (!publicKeyArmored || !timestamp || !signature) return false
156
157    // Parse public key
158    const {
159      keys: [publicKey]
160    } = await openpgp.key.readArmored(publicKeyArmored)
161
162    // Validate signature
163    const verified = await openpgp.verify({
164      message: openpgp.cleartext.fromText(timestamp),
165      signature: await openpgp.signature.readArmored(signature),
166      publicKeys: [publicKey]
167    })
168
169    // Check if signature has expired (is older than SIGNATURE_EXPIRE_TIME)
170    const signatureTime = new Date(timestamp)
171    const isExpired = new Date() - signatureTime > SIGNATURE_EXPIRE_TIME
172
173    const valid = verified.signatures[0].valid && !isExpired
174    const userId = publicKey.getFingerprint()
175    return { valid, userId }
```

Рисунок 3.14 – Функція верифікації підпису

Файл authenter.html представляє собою HTML-сторінку, яка використовується для входу користувачів у вебдодаток EncryptoChat. Після успішної автентифікації або реєстрації, користувач переходить до сторінки чату (chat.html).

4) Головний компонент «Messenger»

Це основний графічний інтерфейс програми (рисунок 3.15), оскільки він показує всі чати та дозволяє користувачеві створювати нові чати та повідомлення. Використовується інтуїтивно зрозумілий дизайн чату, який використовують сучасні програми чату.

На сторінці чату користувач може бачити свій список контактів (завантажений з бази даних) і вибрати одного з контактів для спілкування в компоненті «Start chat with»

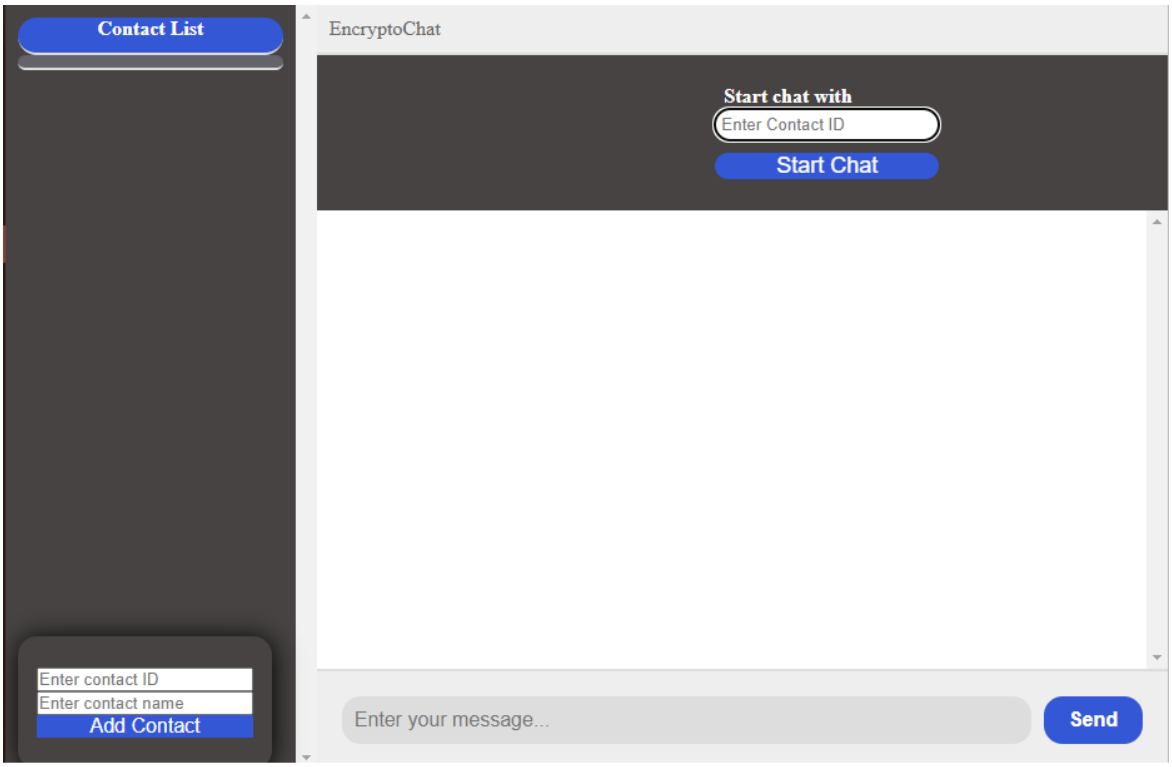

Рисунок 3.15 – Компонент додатку для додавання контактів та початку чатів з ними

Коли користувач вибирає контакт, додаток встановлює захищену сесію з цим контактом, і переходить до приватного чату (рисунок 3.16).

Файл chatting.html представляє собою HTML-сторінку для вебдодатку EncryptoChat, яка відображає інтерфейс чату. Ця сторінка створена для забезпечення функціональності приватного чату в EncryptoChat, де користувачі можуть обмінюватися зашифрованими повідомленнями. Вона відображає список учасників чату, відображає та оновлює повідомлення та надає можливість введення та надсилання повідомлень. У файлі також підключені різні JavaScript-файли, такі як socket.io.js, libsignal-protocol.js, InMemorySignalProtocolStore.js та client.js, які містять логіку для роботи з сокетами, шифруванням повідомлень та обробки подій на сторінці приватного чату.

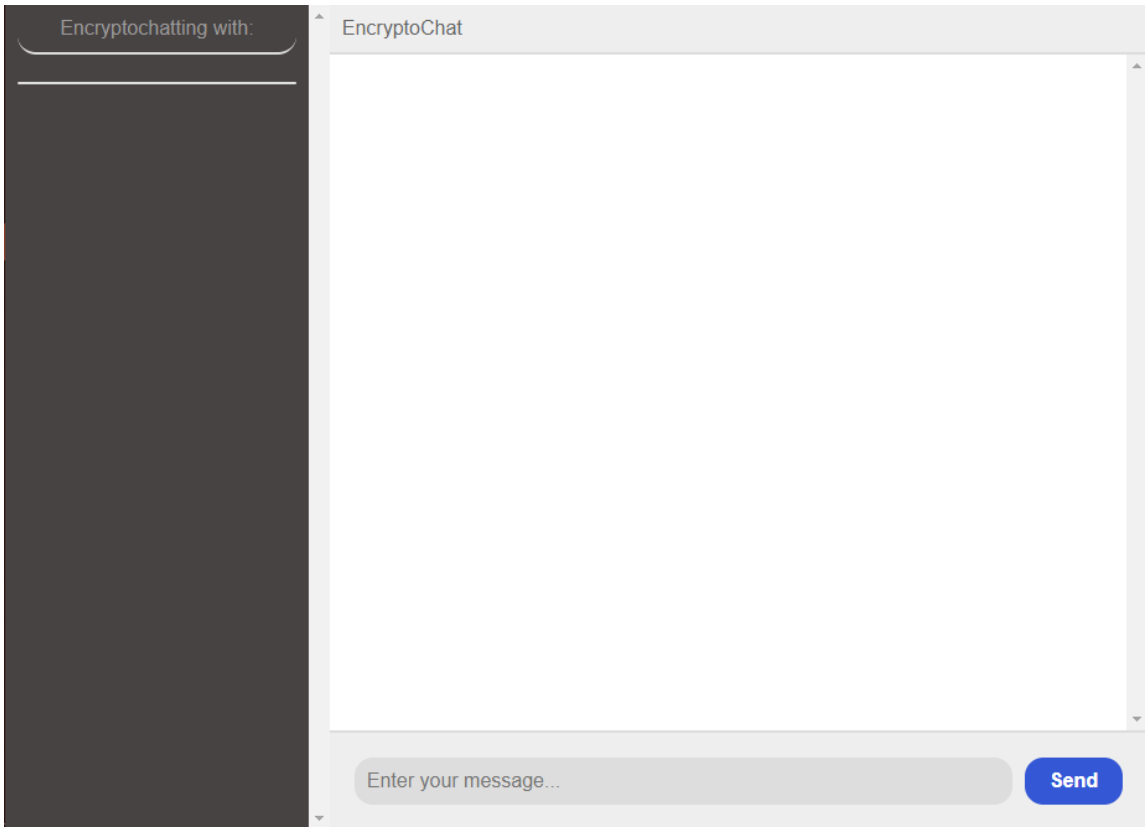

Рисунок 3.16 – Компонент для приватного чату з контактом

На сторінці є бічна панель (sidebar), яка містить заголовок "Encryptochatting with:" та список користувачів, які беруть участь у приватному чаті.

В основній частині сторінки розташована заголовок "EncryptoChat" та блок повідомлень (messages), який відображає отримані та надіслані повідомлення. Внизу сторінки є форма для введення повідомлення, яку користувач може використовувати для надсилання повідомлень у чат.

Коли користувач вибирає контакт, додаток встановлює захищену сесію з цим контактом, використовуючи Signal Protocol для обміну зашифрованими повідомленнями. Користувач може надсилати текстові повідомлення своєму обраному контакту через вікно чату. Повідомлення шифруються за допомогою ключів, збережених у локальному сховищі (InMemorySignalProtocolStore.js). Користувач отримує зашифровані повідомлення від свого контакту і може розшифрувати їх за допомогою своїх ключів. Користувач може завершити сесію чату та вийти з додатку, виконавши відповідну дію повернення.

Наочний приклад обміну миттєвими повідомленнями між двома користувачами наведено на рисунок 3.17:

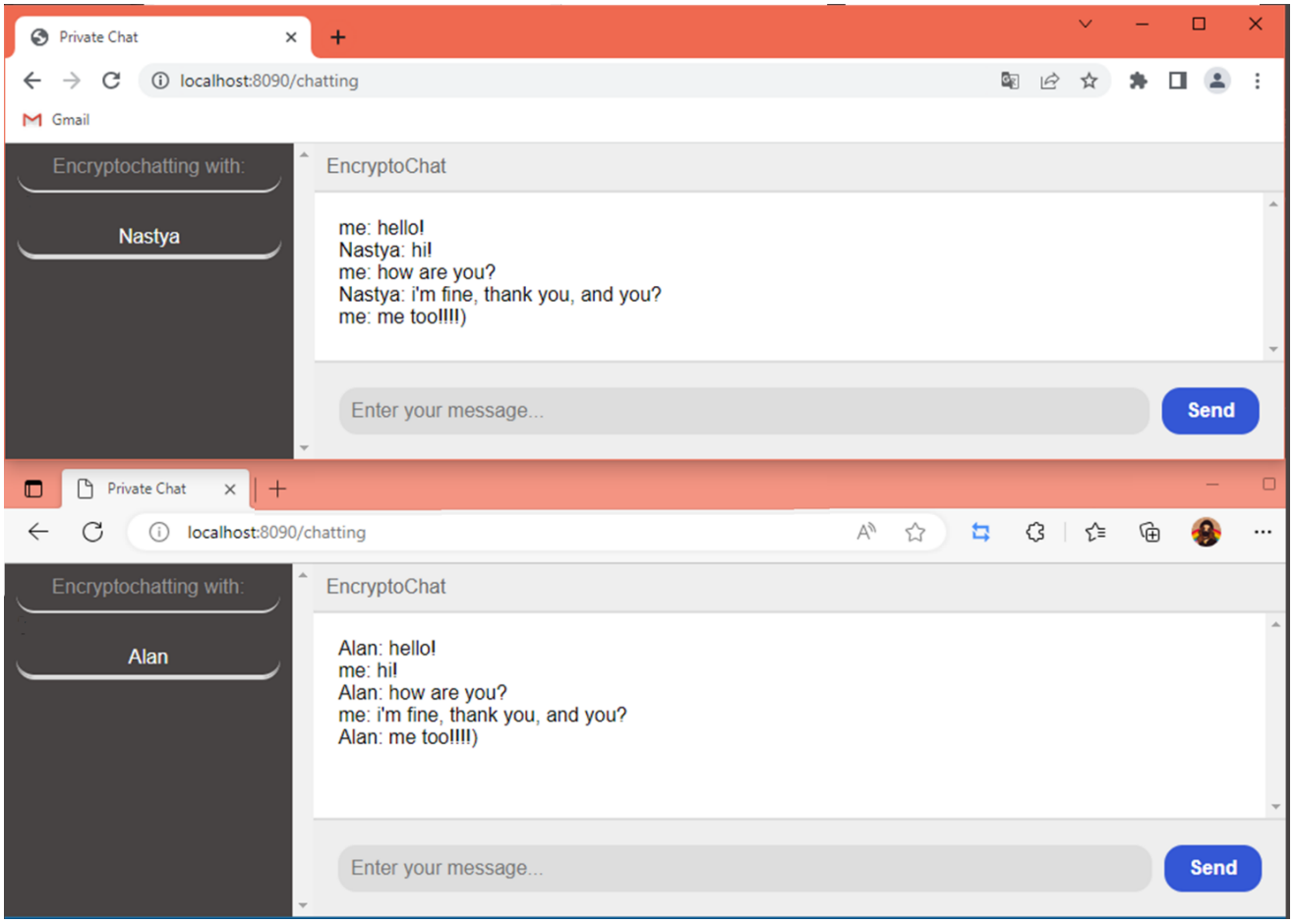

Рисунок 3.17 – Приклад використання додатку двома користувачами

3.4 **Переваги та недоліки**

На основі порівняння з існуючими рішеннями та відповідності попередньо визначених вимог і цілей, можна визначити переваги та недоліки розробленого програмного модуля захищеного обміну миттєвими повідомленнями.

Переваги:

- Швидкий обмін повідомленнями у реальному часі завдяки використанню WebSocket для взаємодії клієнтської та серверної частини. WebSocket забезпечує більшу ефективність та пропускну здатність порівняно з використанням протоколу HTTP.

- Наскрізне шифрування всіх повідомлень, що забезпечує захищений обмін повідомленнями між сторонами. Розроблене рішення використовує для цього сучасний протокол Signal, відповідно усі його властивості безпеки (конфіденційність, цілісність, аутентифікація даних, forward secrecy, post-compromise security) надаються для всіх повідомлень

- Перевірка ідентичності, використовуючи ключі PGP. Користувачі ідентифікуються за своїм публічним ключем PGP, а не за номером телефону чи будь-якою іншою особистою інформацією, як це впроваджено в більшості існуючих рішеннях, тому для використання системи потрібен лише ключ PGP, який можна згенерувати в системі локально. Це забезпечує анонімність користувачів сервісу.

- Реалізовано автентифікацію, надійно перевіряючи особу кожного користувача за ключами PGP Аутентифікація користувачів з використанням ключа PGP гарантує ідентифікацію та перевірку особи для забезпечення безпеки. Використання ідентифікатора користувача з ключа PGP замість особистої інформації спрощує процес використання системи і зберігання даних.

- Система проста у використанні. Налаштування включає лише один крок: генерування ключа PGP шляхом введення імені користувача та парольної фрази або імпортування існуючих. Для забезпечення безпеки не потрібна особиста перевірка, на відміну від існуючих рішень.

- Реалізація програмного модуля у формі вебдодатку, що забезпечує підтримку різних платформ та необхідні універсальність та зручність використання, оскільки користувачі можуть отримати доступ до системи без необхідності встановлювати окремий додаток.

Недоліки:

- Розроблене рішення є синхронною системою обміну миттєвимип повідомленнями, дозволяє користувачам обмінюватися повідомленнями в режимі реального часу негайно, без помітної затримки. Проте недоліком синхронних систем є вимога до одночасної онлайн-присутності користувачів для успішного обміну повідомленнями.

- Впроваджений протокол Signal має потенційний недолік, пов’язаний з реалізацією загрози MITM, що може бути вирішено за допомогою розгортання PKI, що практично передбачає необхідність більш тісної інтеграції протоколу з вже частково використаної в розробці системи PGP.
- Обмін лише текстовими повідомленнями, тому це як і інший користувацький функціонал можна буде розширити далі.
- Невиконані вимоги безпеки вебзастосунків (тому, що на даному етапі дослідження увагу приділено загальній концепції та криптографічній складовій)

Висновки за розділом 3

У даному розділі було описано розроблене рішення для захищеного обміну миттєвими повідомленнями та процес його реалізації з використанням результатів попереднього розділу.

На початку розділу наведено проектну структуру додатку EncryptoChat, що включає клієнтську та серверну частини, використання бази даних. Високорівнева діаграма архітектури була надана для більшого розуміння системи.

Також було описано послідовність дій користувачів для обміну миттєвими повідомленнями, включаючи кроки входу користувачів, генерацію ключів, взаємодію з базою даних та обмін шифрованими повідомленнями. Для опису продемонстровано графічний інтерфейс та наведено лістинг коду. Компоненти користувацького інтерфейсу EncryptoChat необхідні для виконання функціональних можливостей системи та забезпечують зручну взаємодію користувачів з системою.

Результат розробки задовольняє сформульовані попередньо вимоги безпеки, конфіденційності та зручності використання. Враховуючи визначені переваги та недоліки, майбутні дослідження та розробка можуть спрямовуватися на розширення функціоналу, використання сучасних фреймворків для покращення графічного інтерфейсу, інтеграцію PGP з протоколом Signal, забезпечення безпечного

збереження повідомлень, впровадження WebRTC для безпосередньої комунікації клієнтів, поліпшення безпеки вебзастосунку.

ВИСНОВКИ

У дипломній роботі було розроблено програмний засіб для захищеного обміну миттєвими повідомленнями в режимі реального часу через веббраузер з використанням сучасного протоколу наскрізного шифрування Signal та впровадженням криптографічного стандарту OpenPGP для ідентифікації та аутентифікації користувачів в системі.

У першій частині дипломної роботи було проведено огляд існуючих рішень та досліджено їх рівень забезпечення захищеності обміну миттєвими повідомленнями. Для цього було розглянуто основні аспекти захищеності обміну миттєвими повідомленнями, на основі яких сформовано критерії оцінювання безпеки рішень для обміну миттєвими повідомленнями. Критерії розглядають криптографічну складову, збір та передачу даних та метаданих, а також транспарентність та позицію компанії. Порівняльний аналіз месенджерів за критеріями безпеки дав можливість оцінити їх відповідність вимогам безпеки, визначити переваги, недоліки та можливі покращення з точки зору захищеності обміну миттєвими повідомленнями. Додатково розглянуто історію становлення систем обміну миттєвими повідомленнями.

Таким чином, у першій частині було виконано завдання щодо аналізу основних аспектів захищеності систем обміну миттєвими повідомленнями на основі існуючих рішень.

У другій частині дипломної роботи було розглянуто ключові аспекти проектування програмного модуля для захищеного миттєвого обміну повідомленнями. Була поставлено цілі та визначено основні вимоги до програмного модуля, зокрема функціональні та нефункціональні. Спроектовано основні компоненти програмного рішення, визначено процес взаємодії клієнтської та серверної частини з використанням протоколів HTTP та WebSocket, описано використання наскрізного шифрування, ідентифікацію та аутентифікацію користувачів за допомогою PGP. Додаток спроектовано таким чином, аби

забезпечувалась безпека даних та конфіденційність користувачів, а при реалізації програмного модуля у формі вебдодатку забезпечуються необхідні універсальність та зручність використання. Також вибрано технологічний стек для реалізації додатку.

Таким чином, у другій частині було виконано завдання щодо визначення вимог, яким має відповідати програмний модуль та проектування архітектури програмного модуля, що відповідає специфікації вимог.

У третій частині дипломної роботи на основі проаналізованих у першій та другій частині даних було розроблено програмний засіб EncryptoChat відповідно до поставлених цілей та вимог. Наведено проектну структуру додатку, що включає клієнтську та серверну частини, передбачає використання бази даних. Також було розглянуто основні програмні особливості розробленого рішення та представлено користувацький інтерфейс з описом необхідних дій користувачів для захищеного обміну миттєвими повідомленнями.

Таким чином, у третій частині було виконано завдання щодо розробки програмного модуля з урахуванням встановлених вимог та архітектури та проведено оцінку результатів у формі визначення переваг та недоліків розроблюваного рішення.

Отже, всі поставлені завдання було успішно виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Number of mobile phone messaging app users worldwide from 2018 to 2025 [Електронний ресурс]. – Режим доступу до ресурсу: https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/.

2. Bogos C. A security analysis comparison between Signal, WhatsApp and Telegram. / C. Bogos, R. Mocanu, E. Simion. // Cryptology ePrint Archive, Paper 2023/071. – 2023. – С. https://eprint.iacr.org/2023/071.

3. Paterson K. Three Lessons from Threema: Analysis of a Secure Messenger / K. Paterson, M. Scarlata, K. Truong. // USENIX Security 2023. – 2023. – С. https://eprint.iacr.org/2023/071.

4. Secure Messaging Apps Comparison [Електронний ресурс]. – Режим доступу до ресурсу: https://www.securemessagingapps.com/about/.

5. EFF Secure Messaging Scorecard [Електронний ресурс]. – Режим доступу до ресурсу: https://www.eff.org/pages/secure-messaging-scorecard

6. Larson G. Instant Messaging [Електронний ресурс] / Gary Larson // Encyclopedia Britannica. – 2023. – Режим доступу до ресурсу: https://www.britannica.com/topic/instant-messaging.

7. Bamford J. The NSA Is Building the Country's Biggest Spy Center (Watch What You Say) [Електронний ресурс] / James Bamford // Wired Magazine. – 2012. – Режим доступу до ресурсу: https://www.wired.com/2012/03/ff-nsadatacenter/.

8. Greenwald G. NSA Prism program taps in to user data of Apple, Google and others / G. Greenwald, E. MacAskill. // The Guardian. – 2013. – №7. – С. 1–43.

9. Green M. Looking back at the Snowden revelations [Електронний ресурс] / Matthew Green // A few thoughts on cryptographic engineering. – 2019. – Режим доступу до ресурсу: https://blog.cryptographyengineering.com/2019/09/24/looking-back-at-the-snowden-revelations/.

10. Фесенко А.О., Хоменко О. В. Дослідження критеріїв безпеки у сучасних системах обміну миттєвими повідомленнями // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: V Міжнародна науково-практична конференція – Київ, 27-28 жовтня 2022.

11. Фесенко А.О., Хоменко О. В. Огляд криптографічних протоколів наскрізного шифрування використаних у системах обміну миттєвими повідомленнями // Прикладні системи та технології в інформаційному суспільстві: VI Міжнародна науково-практична конференція – Київ, 30 вересня 2022.

12. Фесенко А.О., Хоменко О. В. Вплив мережевої архітектури додатків для обміну миттєвими повідомленнями на їх безпеку // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: VI Міжнародна науково-практична конференція – Київ, 27 квітня 2023.

13. Most popular global mobile messaging apps 2023 [Електронний ресурс]. – Режим доступу до ресурсу: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/

14. WhatsApp Encryption Overview Technical Whitepaper [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.

15. WhatsApp Privacy Policy [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.whatsapp.com/legal/privacy-policy

16. WhatsApp’s New Privacy Policy Just Kicked In. Here’s What You Need to Know [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.wired.com/story/whatsapp-privacy-policy-facebook-data-sharing/

17. Офіційний сайт Telegram [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/.

18. Статистика використання месенджерів в світі 2023: [Електронний ресурс] – 2023. – Режим доступу до ресурсу: https://www.similarweb.com/blog/research/market-research/worldwide-messaging-apps/

19. Telegram MTProto Documentation [Електронний ресурс]. – Режим доступу: https://core.telegram.org/mtproto.

20. Jakobsen J. On the CCA (in)security of MTProto / J. Jakobsen, C. Orlandi. // Cryptology ePrint Archive, Paper 2015/1177. – 2015. – С. https://eprint.iacr.org/2015/1177.

21. Офіційна політика конфіденційності Telegram. [Електронний ресурс]. – Режим доступу: https://telegram.org/privacy.

22. Репозитарій Signal [Електронний ресурс]. – Режим доступу: https://github.com/signalapp

23. Офіційний сайт організації Signal Foundation [Електронний ресурс]. – Режим доступу: https://signalfoundation.org/en/

24. Офіційна політика конфіденційності Signal [Електронний ресурс]. – Режим доступу:https://signal.org/legal/#privacy-policy

25. Recommendations for real-time communication [Електронний ресурс] – Режим доступу до ресурсу: https://www.privacyguides.org/en/real-time-communication/.

26. Faife C. Swiss Army drops WhatsApp for homegrown messaging service, citing privacy concerns [Електронний ресурс] / Corin Faife. – 2022. – Режим доступу до ресурсу: https://www.theverge.com/2022/1/7/22871881/swiss-army-whatsapp-messaging-threema-privacy-concerns-us-jurisdiction.

27. Schweizer Armee verbietet Whatsapp und Co [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.srf.ch/news/schweiz/threema-setzt-sich-durch-schweizer-armee-verbietet-whatsapp-und-co.

28. Threema. Privacy Policy [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://threema.ch/en/privacy-policy

29. The Privacy Pledge: An Alternative Vision to Big Tech’s Internet [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://threema.ch/en/blog/posts/privacy-pledge.

30. Офіційний вебсайт месенджеру Wire [Електронний ресурс]. – Режим доступу до ресурсу: https://wire.com/en/about/

31. Wire Security Technical Whitepaper [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://wire-docs.wire.com/download/Wire+Security+Whitepaper.pdf

32. Офіційна політика конфіденційності Wire [Електронний ресурс]. – Режим доступу: https://wire.com/en/privacy/

33. Session Protocol: Technical implementation details [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://getsession.org/blog/session-protocol-technical-information

34. Kee J. A Model for End-To-End Encrypted Conversations With Minimal Metadata Leakage [Електронний ресурс] / J. Kee, M. Shishmarev, S. Harman // arXiv. – Режим доступу до ресурсу: https://arxiv.org/pdf/2002.04609.pdf.

35. Офіційна політика конфіденційності Session [Електронний ресурс]. – Режим доступу: [https://getsession.org/privacy-policy]

36. Perrin T. The noise protocol framework [Електронний ресурс] / Trevor Perrin. – 2017. – Режим доступу до ресурсу: http://noiseprotocol.org/noise.html.

37. WhatsApp FAQ. How to back up to iCloud [Електронний ресурс]. – Режим доступу до ресурсу: https://faq.whatsapp.com/902477924463699/?helpref=uf_share

38. Can Snowden finally kill the 'harmless metadata' myth? [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://www.zdnet.com/article/can-snowden-finally-kill-the-harmless-metadata-myth/.

39. Rosenberg M. How Trump Consultants Exploited the Facebook Data of Millions [Електронний ресурс] / Matthew Rosenberg // The New York Times. – 2018. – Режим доступу до ресурсу: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html.

40. FBI document shows what data can be obtained from encrypted messaging apps [Електронний ресурс] // The Record. – 2021. – Режим доступу до ресурсу: https://therecord.media/fbi-document-shows-what-data-can-be-obtained-from-encrypted-messaging-apps.

41. Fette I. The WebSocket Protocol [Електронний ресурс] / I. Fette, A. Melnikov // RFC, 6455. – 2021. – Режим доступу до ресурсу: http://www.rfc-editor.org/rfc/rfc6455.

42. Cohn-Gordon K. A formal security analysis of the signal messaging protocol. / K. Cohn-Gordon, C. Cremers, B. Dowling. // IEEE European Symposium on Security and Privacy (EuroS&P),. – 2017. – С. 451–466.

43. Adams C. Understanding PKI: Concepts, Standards, and Deployment Considerations / C. Adams, L. Steve., 2002. – 352 с. – (2nd edition).

44. Бібліотека Socket.IO [Електронний ресурс]. – Режим доступу до ресурсу: https://socket.io/blog/socket-io-p2p/

45. JavaScript-бібліотека протоколу Signal [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/signalapp/libsignal-protocol-javascript

46. Модуль Sqlite [Електронний ресурс]. – Режим доступу до ресурсу: https://www.npmjs.com/package/sqlite3

47. Бібліотека протоколу OpenPGP.js [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/openpgpjs/openpgpjs

ДОДАТОК А
СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИПЛОМНОЇ РОБОТИ

Тези наукових доповідей

1. Фесенко А.О., Хоменко О. В. Дослідження критеріїв безпеки у сучасних системах обміну миттєвими повідомленнями // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: V Міжнародна науково-практична конференція – Київ, 27-28 жовтня 2022.
2. Фесенко А.О., Хоменко О. В. Огляд криптографічних протоколів наскрізного шифрування використаних у системах обміну миттєвими повідомленнями // Прикладні системи та технології в інформаційному суспільстві: VI Міжнародна науково-практична конференція – Київ, 30 вересня 2022.
3. Фесенко А.О., Хоменко О. В. Вплив мережевої архітектури додатків для обміну миттєвими повідомленнями на їх безпеку // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: VI Міжнародна науково-практична конференція – Київ, 27 квітня 2023.

ДОДАТОК Б
ТАБЛИЦЯ ОЦІНКИ ВІДПОВІДНОСТІ МЕСЕНДЖЕРІВ КРИТЕРІЯМ ВІДПОВІДНО ДО ВИЗНАЧЕНИХ РІВНІВ БЕЗПЕКИ

Таблиця Б.1

Оцінка відповідності критеріям відповідно до визначених рівнів безпеки месенджерів

Критерій	**Низький рівень безпеки**	**Середній рівень безпеки**	**Високий рівень безпеки**
І. Криптографічна складова			
Використання надійних криптографічних примітивів	Месенджер використовує криптографічні примітиви, які вважаються ненадійними, існують практичні атаки для злому.	Месенджер використовує криптографічні примітиви, які вважаються слабкими, однак відомих практичних атак проти них поки немає.	Месенджер використовує надійні, добре відомі безпечні криптографічні примітиви
Впровадження E2EE (за промовчанням)	Ні, додаток не використовує жодної форми наскрізного шифрування.	Додаток використовує певну форму шифрування E2E, але воно не використовується за промовчанням, потребує налаштування вручну або має відомі недоліки.	Так, використовує надійне шифрування E2E із надійним управлінням ключами та без відомих вразливостей
Створення та зберігання приватних ключів на кінцевому пристрої	Ні		Так
Підтримка PFS	Ні		Так
Попереднє шифрування повідомлень перед cloud backup	Ні		Так

продовження табл. Б.1

Шифрування метаданих	Ні	Більшість метаданих зашифровано. Однак деякі зберігаються компанією.	Так
Гешування персональних даних	Персональні дані не гешуються	Обмежена кількість ідентифікаційної інформації (номери мобільних телефонів) не гешується. Вся інша інформація, включаючи контакти, гешується	Усі персональні дані гешується.
Використання TLS/Noise для шифрування мережевого трафіку	Ні		Так
Використання certificate pinning	Ні		Так
Надійна система автентифікації/ механізми аутентифікації (2FA)	Ні		Так
II. Збір та передача даних та метаданих			
Збір даних користувачів компанією	Так, збирають більше, ніж потрібно для функціонування програми. Збирають інші дані клієнтів для інших частин свого бізнесу.	Збирають лише мінімальну кількість (наприклад, номер мобільного телефону чи адресу електронної пошти) даних про клієнтів, щоб забезпечити роботу програми	Не збирають дані користувачів.
Збір даних користувачів додатком	Так, збирає більше, ніж потрібно для функціонування програми. Збирають незахищені дані клієнтів або самі	Збирає лише мінімальну кількість (наприклад, номер мобільного телефону чи адресу електронної пошти) даних про клієнтів, щоб забезпечити програму	Не збирає дані користувачів.

	надіслані повідомлення.	безпечного обміну повідомленнями.	

продовження табл. Б.1

Передача даних та/або метаданих холдинговій компанії та/або третім особам	Дані та/або метадані користувачів додатку відправляються до батьківської компанії або третіх сторін,	Мінімальний, номер мобільного телефону надсилається третій стороні лише для реєстрації та відновлення	Дані та/або метадані користувачів додатку не відправляються до батьківської компанії або третіх сторін,
Причетність до передачі даних користувачів спецслужбам	Компанія причетна до надання даних клієнтів спецслужбам, що доведено доказами.	Компанія причетна до надання даних клієнтів спец. службам, що недоведено доказами, але джерело авторитетне.	Компанія не була причетна до надання даних клієнтів спецслужбам.
Логування позначок часу (timestamp) та IP-адрес	Так	Деяка інформація про позначку часу/IP-адресу зберігається, хоча вона не зберігається для кожного надісланого повідомлення.	Ні
Можливість анонімної реєстрації	Ні, користувачі повинні надати певні контактні дані, наприклад адресу електронної пошти або номер мобільного телефону.	Потрібно вказати електронну адресу або номер мобільного телефону. Однак вони, як можна довести, гешуються, а отже, компанія їх не читає.	Так, не потрібно повідомляти жодних персональних даних, щоб використовувати додаток.
Повідомлення, що самознищуються	Ні		Так
IV. Транспарентність та позиція компанії			
Повна відкритість вихідного коду	Ні, вихідний код повністю закритий	Частково	Так, додаток і серверна частина мають відкритий вихідний код
Відтворювані збірки для перевірки	Ні		Так

Нещодавній публічний аудит безпеки	Ні		Так

продовження табл. Б.1

Звіт про прозорість	Компанія не надає періодичний звіт про прозорість (або це не дуже корисно).		Компанія періодично надає змістовний звіт про прозорість.
Джерела фінансування	Фінансується компанією/особою, яка/яка добре пов'язана або добре відома тим, що збирає дані клієнтів. Або вони відомі тим, що збирають дані клієнтів або співпрацюють з органами влади, коли справа доходить до запиту даних клієнтів.		Фінансується компаніями/людьми, які/які зацікавлені в шифруванні/захисті даних клієнтів або не мають очевидних причин проти цього. Вони не повинні бути відомі тим, що збирають дані клієнтів або співпрацюють з органами влади, коли справа доходить до запиту даних клієнтів.
Юрисдикція компанії	Компанія знаходиться під юрисдикцією відомого партнера Five Eyes. Або компанія знаходиться під юрисдикцією країни, яка добре відома масовим стеженням.	Компанія знаходиться під юрисдикцією, яка не відома як масове стеження або змушує компанії передавати чи розшифровувати дані. Або відомо, що країна співпрацює з округами Five Eyes.	Компанія знаходиться під юрисдикцією, якане відома як [масове] стеження або змушує компанії передавати чи розшифровувати дані. Немає відомих зв'язків із Five Eyes тощо.

Юрисдикція інфраструктури	Інфраструктура знаходиться під юрисдикцією відомого партнера Five Eyes. Або компанія знаходиться під юрисдикцією країни, яка добре відома своїм стеженням.	Інфраструктура знаходиться під юрисдикцією, яка не відома [масовим] стеженням або змушує компанії передавати чи розшифровувати дані. Або відомо, що країна співпрацює з округами Five Eyes.	Інфраструктура знаходиться під юрисдикцією, яка не відома [масовим] стеженням або змушує компанії передавати чи розшифровувати дані. Немає відомих зв'язків із Five Eyes тощо.

продовження табл. Б.1

Загальна позиція компанії щодо конфіденційності клієнтів	Погана Компанія не розробляє свої системи для збору мінімальної інформації про клієнтів або не має надійного контролю шифрування/безпеки; або не має простої, зрозумілої політики конфіденційності та умов. Або відомо, що компанія співпрацює з юридичними (чи неофіційними) запитами на інформацію про клієнтів. Або бізнес-модель компанії спирається на дані користувачів.		Хороша Компанія розробляє свої системи для збору мінімальної інформації про клієнтів; має потужні елементи керування шифруванням/безпекою; простий; зрозуміла політика конфіденційності та умови використання. Компанія не може передати дані користувачів урядам, навіть якщо їх попросять. так само Компанія, як відомо, бореться з судовими проблемами щодо розшифровки або іншим способом передачі даних клієнтів. Бізнес-модель компанії не відповідає на дані користувачів.

ДОДАТОК В
ЛІСТИНГ ОСНОВНОГО ВИХІДНОГО РОЗРОБЛЕНОГО РІШЕННЯ

src/index.js:

```
// index.js, який відповідає за створення сервера,
const http = require('http')
const express = require('express')
const cookieParser = require('cookie-parser')
const socketio = require('socket.io')
const config = require('./config')
const router = require('./routes')
const app = express()
const server = http.createServer(app)
const io = socketio(server)
require('./chat/chat')(io)
//chatgpt:
const userModule = require('./chat/user');
const contactModule = require('./chat/contacts');
const sqlite3 = require('sqlite3').verbose();

const DB_PATH = "D:\\!KSENNYA\\учеба\\4курс\\4.2\\!дипломне проектування\\РОЗРОБКА\\EncryptoChat\\data\\mybase.db";
const db = new sqlite3.Database(DB_PATH);
// Передача підключення до бази даних в модуль user.js
userModule.init(db);
//
app.use(express.static(config.pubDirPath))
app.use(cookieParser())
app.use((express.urlencoded({ extended: false })))
app.use(router)
server.listen(config.port, () => {
    console.log('Server is up on port ' + config.port)
})
```

***src/config.js*:**

```
path = require('path')
```

```
const sqlite3 = require('sqlite3').verbose();

const DB_PATH = "D:\\!KSENNYA\\учеба\\4курс\\4.2\\!дипломне проектування\\РОЗРОБКА\\EncryptoChat\\data\\mybase.db";
// Створення нового підключення до бази даних
const db = new sqlite3.Database(DB_PATH, (err) => {
  if (err) {
    console.error('Database connection error:', err);
  } else {
    console.log('Connected to SQLite database');

    const sql = 'CREATE TABLE IF NOT EXISTS UserContacts (ident INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER NOT NULL, contact_id INTEGER NOT NULL, contact_name TEXT)';
    db.run(sql, function (err) {
      if (err) {
        console.error('Error creating table:', err);
      } else {
        console.log('Table UserContacts created successfully.');
      }
    });
     const sql1 = 'CREATE TABLE IF NOT EXISTS Users (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, publicKey TEXT)';
    db.run(sql1, function (err) {
      if (err) {
        console.error('Error creating table:', err);
      } else {
        console.log('Table Users created successfully.');
      }
    });
    const sql3 = 'SELECT * FROM Users';
    db.all(sql3, function (err, rows) {
      if (err) {
        console.error('Error retrieving users:', err);
      } else {
        const users = rows.map(row => ({
          id: row.id,
          username: row.username,
          publicKey: row.publicKey
        }));
```

```
          console.log(users);
        }
      });
  const sql4 = 'SELECT * FROM UserContacts';
  db.all(sql4, function (err, rows) {
    if (err) {
      console.error('Error retrieving users:', err);

    } else {
      const contacts = rows.map(row => ({
        ident: row.ident,
        user_id: row.user_id,
        contact_id: row.contact_id,
        contact_name: row.contact_name
             }));
       console.log(contacts);
    }
  });
 db.all(sql4, function (err, rows) {
    if (err) {
      console.error('Error retrieving users:', err);

    } else {
      const contacts = rows.map(row => ({
        ident: row.ident,
        user_id: row.user_id,
        contact_id: row.contact_id,
        contact_name: row.contact_name
             }));
       console.log(contacts);
    }
  });
}
});
module.exports = {
    port: process.env.PORT || 8090,
    pubDirPath: path.join(__dirname, '../public'),
    viewsDirPath: path.join(__dirname, 'views'),
        db: db // Додано підключення до бази даних
```

```
}
```

src/routes.js:

```
const path = require('path')
const express = require('express')
const jwt = require('jsonwebtoken')
const config = require('./config')
const openpgp = require('openpgp')
//const userContacts = require('../public/js/userContacts.js');
const router = express.Router()
router.use(express.json()); //через цей рядок скільки проблем було мда

router.get('/', (req, res) => {
    res.sendFile(path.join(config.viewsDirPath, 'home.html'))
})

router.get('/register.html', (req, res) => {
  res.sendFile(path.join(config.viewsDirPath, 'register.html'));
});

router.get('/authenter.html', (req, res) => {
  res.sendFile(path.join(config.viewsDirPath, 'authenter.html'));
});

// Маршрут для обробки авторизації
router.post('/login', async (req, res) => {
  const { username, password } = req.body;
  try {
      const userId = await userModule.authenticateUser(username, password);
      if (userId !== null) {
        const token1 = jwt.sign(
          {
            username: req.body.username,
            userId: userId // Додали userId до пейлоаду токена
          },
          config.jwtKey
        );
        console.log("Authentication successful for user:", username);
```

```
        res.cookie(config.cookieName, token1);

        // Встановити req.user замість простого перенаправлення
        req.user = {
        id: userId,
        username: req.body.username
      };
        return res.redirect('/messenger');
        //return next(); // Перейти до наступного обробника маршруту
      } else {
        console.log('Authentication failed for user:', username);
        return res.status(401).json({ error: 'Invalid credentials' });
      }
    } catch (error) {
      console.error('Error authenticating user:', error);
      return res.status(500).send({ error: 'Internal server error' });
    }
  });

router.get('/messenger', async (req, res) => {
  console.log("Перейшло до /messenger:");
    //для налагодження res.sendFile(path.join(config.viewsDirPath, 'chat.html'))
//chatgpt
  const token1 = req.cookies[config.cookieName];
  console.log("token1:", token1);
  if (!token1) {
      console.log('No token found. Redirecting to /');
      return res.redirect('/');
  }
  try {
      const decodedToken1 = jwt.verify(token1, config.jwtKey);
      const username = decodedToken1.username;
      console.log('User authenticated:', username);
      return res.redirect('/messaging')

  } catch (error) {
      console.log('Error decoding token:', error);
      res.redirect('/');
  }
```

```
});

  router.get('/messaging', async (req, res) => {
      console.log("Перейшло до /messaging:");
           res.sendFile(path.join(config.viewsDirPath, 'chat.html'));
    })

const userModule = require('./chat/user');
// Маршрут для обробки реєстрації
router.post('/register', (req, res) => {
  const {username, publicKey} = req.body;
  // Виклик функції реєстрації з модуля user.js
  console.log("Router post register.....")
  userModule.registerUser(username, publicKey);
  // Відповідь з успішним статусом
  res.sendStatus(200);
  //return res.redirect('/chat');
});

const openpgp = require('openpgp'),
  { SIGNATURE_EXPIRE_TIME } = require('../../config')

// Verifies a pgp signature is valid and not expired and returns the userId
exports.pgpVerify = async (publicKeyArmored, timestamp, signature) => {
  // Fail verification if all params are not supplied
  if (!publicKeyArmored || !timestamp || !signature) return false

  // Parse public key
  const {
    keys: [publicKey]
  } = await openpgp.key.readArmored(publicKeyArmored)

  // Validate signature
  const verified = await openpgp.verify({
    message: openpgp.cleartext.fromText(timestamp),
    signature: await openpgp.signature.readArmored(signature),
    publicKeys: [publicKey]
  })
```

```
        // Check if signature has expired (is older than SIGNATURE_EXPIRE_TIME)
        const signatureTime = new Date(timestamp)
        const isExpired = new Date() - signatureTime > SIGNATURE_EXPIRE_TIME

        const valid = verified.signatures[0].valid && !isExpired
        const userId = publicKey.getFingerprint()
        return { valid, userId }
      }

router.post('/sign', (req, res) => {

         const { name, passphrase, publicKeyArmored, privateKeyArmored, message } =
req.body;
        (async () => {
          console.log("hmhmhmm");
        const publicKey = await openpgp.readKey({ armoredKey: publicKeyArmored });
        const privateKey = await openpgp.decryptKey({
          privateKey: await openpgp.readPrivateKey({ armoredKey: privateKeyArmored }),
          passphrase
        });

        const unsignedMessage = await openpgp.createCleartextMessage({ text: message });
        const cleartextMessage = await openpgp.sign({
          message: unsignedMessage,
          signingKeys: privateKey
        });
        console.log("clear: ", cleartextMessage); // '-----BEGIN PGP SIGNED MESSAGE ...
        res.json({
          cleartextMessage:cleartextMessage
      })
        //
      })();
      });

      const userModule1 = require('../src/chat/user');
      router.post('/verify', (req, res) => {
        temp = false;
        const { name, cleartextMessage } = req.body;
        (async () => {
```

```
  //дістати з БД публічний ключ за іменем
        const publicKeyArmored = await userModule1.getPublicKey(name);
        const publicKey = await openpgp.readKey({ armoredKey: publicKeyArmored });
//
      const signedMessage = await openpgp.readCleartextMessage({
        cleartextMessage // parse armored message
    });
      const verificationResult = await openpgp.verify({
        message: signedMessage,
        verificationKeys: publicKey
      });
      const { verified, keyID } = verificationResult.signatures[0];
      try {
        await verified; // throws on invalid signature
        console.log('Signed by key id ' + keyID.toHex());
        temp = true;
       // return res.redirect('/login-auth');
      } catch (e) {
        temp = false;
        console.log('Authentication failed for user:', );
        return res.status(401).json({ e: 'Invalid credentials' });
        //throw new Error('Signature could not be verified: ' + e.message);
             }
      res.json({
        verify_result: temp
    })
})();
});
// Маршрут для обробки авторизації
router.post('/login-auth', async (req, res) => {
  const { name, verify_result} = req.body;
  console.log("AUTHUN")
  console.log("login - verify_result", verify_result)
  console.log("login - username", name)
  try {
      if(verify_result)
      {
          const userId = await userModule1.getUserId(name);
```

```
        const token1 = jwt.sign(
          {
            username: req.body.name,
            userId: userId // Додали userId до пейлоаду токена
          },
          config.jwtKey
        );
        console.log("Authentication successful for user:");
        res.cookie(config.cookieName, token1);

        // Встановити req.user замість простого перенаправлення
        req.user = {
        id: userId,
        username: req.body.name
      };
        return res.redirect('/messenger');
        //return next(); // Перейти до наступного обробника маршруту
    } else {
      console.log('Authentication failed for user:', username);
      return res.status(401).json({ error: 'Invalid credentials' });
   }
  } catch (error) {
    console.error('Error authenticating user:', error);
    return res.status(500).send({ error: 'Internal server error' });
  }
});
  const contactModule = require('./chat/contacts');
  const authMiddleware = (req, res, next) => {
    const token1 = req.cookies[config.cookieName];
    if (!token1) {
      console.log('No token found. Redirecting to /');
      return res.redirect('/');
    }
    try {
      const decodedToken1 = jwt.verify(token1, config.jwtKey);
      req.user = {
        id: decodedToken1.userId,
        username: decodedToken1.username
      };
```

```
            next();
          } catch (error) {
            console.log('Error decoding token:', error);
            res.redirect('/');
          }    };
        // Маршрут для отримання списку контактів
    router.get('/contacts', authMiddleware, async (req, res) => {
        try {
            const userId = req.user.id; // Get userId from the authentication token
             const contacts = await contactModule.getContacts(userId); // Get the list
of contacts for the user
            res.status(200).json(contacts);
          } catch (error) {
            console.error('Error retrieving contacts:', error);
            res.status(500).json({ error: 'Internal server error' });
          }
      });
    //
    // Маршрут для додавання контакту
    router.post('/contacts', authMiddleware, async (req, res) => {
        try {
          const userId = req.user.id; // Отримати userId з аутентифікаційного токена
          const contactId = req.body.contactId;
          console.log('Contact ID:', contactId);
          const contactName = req.body.contactName; // Отримати ім'я контакту з запиту
          console.log('Contact contactName:', contactName);
          // Перевірка наявності значення contactId
        if (!contactId) {
          console.error('Contact ID is required', error);
          return res.status(400).json({ error: 'Contact ID is required' });
        }
         // Перевірити, чи існує користувач з вказаним ID або ім'ям
             const   userExists   =   await   contactModule.checkUserExists(contactId,
contactName);
          if (userExists) {

          // Виконати необхідні дії для додавання контакту
          const id = await contactModule.addContact(userId, contactId, contactName);
          res.status(200).json({ message: 'Contact added successfully, room: ', id });
```

```
          }
          else {
          res.status(400).json({ error: 'User does not exist' });
            }
          } catch (error) {
            console.error('Error adding contact:', error);
            res.status(500).json({ error: 'Internal server error' });
          }
        });

        router.post('/join', authMiddleware, async (req, res) => {
          const userId = req.user.id;
          const contactInput = req.body.contactinput.trim();
                    console.log("userid:contactId:username   ",   userId,   contactInput,
req.user.username);
           // Виконати необхідні дії для отримання ID за contactInput з бази даних
           const roomname = await contactModule.getRoomId(userId, contactInput);
           username = req.user.username;

          if (username && roomname) {
              const token = jwt.sign({
                  username: username,
                  roomname: roomname
              }, config.jwtKey)

              res.cookie(config.cookieName, token)
              return res.redirect('/chatting')
          }
          res.redirect('/')
      })
      router.get('/chatting', async (req, res) => {
        res.sendFile(path.join(config.viewsDirPath, 'chatting.html'))
      })
      module.exports = router
      // Маршрут для обробки авторизації
      router.post('/login', async (req, res) => {
        const { username, password } = req.body;
        try {
            const userId = await userModule.authenticateUser(username, password);
```

```
      if (userId !== null) {
        const token1 = jwt.sign(
          {
            username: req.body.username,
            userId: userId // Додали userId до пейлоаду токена
          },
          config.jwtKey
        );
        console.log("Authentication successful for user:", username);
        res.cookie(config.cookieName, token1);

        // Встановити req.user замість простого перенаправлення
        req.user = {
        id: userId,
        username: req.body.username
      };
        return res.redirect('/messenger');
        //return next(); // Перейти до наступного обробника маршруту
      } else {
        console.log('Authentication failed for user:', username);
        return res.status(401).json({ error: 'Invalid credentials' });
      }
    } catch (error) {
      console.error('Error authenticating user:', error);
      return res.status(500).send({ error: 'Internal server error' });
    }
  });
```

public/js/client.js:

```
(function () {
    // get cookie, remove cookie name
    const token = document.cookie.slice(document.cookie.indexOf('=') + 1)
    // init socket.io
    const socket = io({ query: 'token=' + token })
    // init store
    const store = new SignalProtocolStore()
    // init keyhelper
    const keyHelper = libsignal.KeyHelper
    // object that holds individual cyphers
```

```
        const sessionCipher ={}
        // no need for multi device support, so hardcoding 0
        const deviceId = 0
        // generate keyId
        // needs to be unique on all machines
        // needs to be number
        const generateKeyId = socketId => {
            let keyId = ''
            for (let i = 0; i < 2; i++) {
                        keyId   +=   socketId.charCodeAt(Math.floor(Math.random()   *
socketId.length))
            }
            const date = String(Date.now()).slice(-3)
            return Number(keyId + date)
        }


        // generate registration id & identity key pair
        const generateIdentity = async () => {

            // generate registration id
            const registrationId = keyHelper.generateRegistrationId();
            // store registration id
            store.put('registrationId', registrationId)

            // generate identity key pair
            const identityKeyPair = await keyHelper.generateIdentityKeyPair()
            // store identity key pair
            store.put('identityKey', identityKeyPair)
        }
        const generatePreKeys = async  socketId => {
            // generate keyId
            const keyId = generateKeyId(socketId)
            // get identity key pair
            const identityKeyPair = await store.getIdentityKeyPair()
            // generate signed pre key
            const signedPreKey = await keyHelper.generateSignedPreKey(identityKeyPair,
keyId)
            // save signed pre key
```

```
    store.storeSignedPreKey(signedPreKey.keyId, signedPreKey.keyPair)

    // generate pre key
    const preKey = await keyHelper.generatePreKey(keyId)
    // save pre key
    store.storePreKey(preKey.keyId, preKey.keyPair)
    // return key bundle to share with other clients
    return {
        registrationId: await store.getLocalRegistrationId(),
        identityKey: identityKeyPair.pubKey,
        signedPreKey: {
            keyId: signedPreKey.keyId,
            publicKey: signedPreKey.keyPair.pubKey,
            signature: signedPreKey.signature
        },
        preKey: {
            keyId: preKey.keyId,
            publicKey: preKey.keyPair.pubKey
        }
    }
}

const buildSession = async (username, keyBundle) => {
    // recipient address
    const address = new libsignal.SignalProtocolAddress(username, deviceId)
    // session builder for that address
    const sessionBuilder = new libsignal.SessionBuilder(store, address)
    // create session
    await sessionBuilder.processPreKey(keyBundle)
    // return session cipher for that address
    return new libsignal.SessionCipher(store, address)

}

const userList = new Set()
const $usersList = document.querySelector('#user-list')

const renderUserList = () => {
    $usersList.innerHTML = ''
```

```
    userList.forEach(user => {
        const $li = document.createElement('li')
        $li.innerHTML = user
        $usersList.append($li)
    })
}

const addUser = username => {
    userList.add(username)
    renderUserList()
}

const removeUser = username => {
    userList.delete(username)
    renderUserList()
}

const $roomName = document.querySelector('#room-name')
// when client connects
socket.on('join', async room => {
    $roomName.innerHTML = room
    // generate identity
    await generateIdentity()
    // generate pre keys
    const keyBundle = await generatePreKeys(socket.id)
    // send key request to other clients
    socket.emit('newSession', keyBundle)
})

// when client recives request for new session
socket.on('newSession', async data => {
    addUser(data.from)
    // generate pre keys
    const keyBundle = await generatePreKeys(socket.id)
    // create session
    sessionCipher[data.from] = await buildSession(data.from, data.body)
    // send own keys to requester
    socket.emit('newSessionReplay', {
        to: data.from,
```

```
        body: keyBundle
    })
})

// build session from recived keys
socket.on('newSessionReplay', async data => {
    addUser(data.from)
    sessionCipher[data.from] = await buildSession(data.from, data.body)

})


const $controlls = document.querySelector('#controlls')
const $message = $controlls.querySelector('input')
const $submit = $controlls.querySelector('button')
const $messages = document.querySelector('#messages')

const renderMessage = message => {
    const $p = document.createElement('p')
    $p.textContent = message
    $messages.append($p)
    $messages.scrollTo(0, $messages.scrollHeight)
    return $p
}

const renderError = message => {
    p = renderMessage(message)
    p.classList.add('error')
}

const sendMessage = async event => {
    event.preventDefault()
    const message = $message.value
    renderMessage('me: ' + message)
    for (let id in sessionCipher) {
        const encrypted = await sessionCipher[id].encrypt(message)
        socket.emit('message', {
            to: id,
            body: encrypted,
```

```
                })
            }
            $message.value = ''
        }

        $submit.addEventListener('click', sendMessage)
        $submit.addEventListener('ontouchstart', sendMessage)
        window.addEventListener('keydown', event => {
            if (event.keyCode === 13) {
                sendMessage(event)
            }
        })
        socket.on('message', async data => {
            let decrypted
            // if message is pre key whisper message
            if (data.body.type === 3) {
                                                    decrypted       =       await
sessionCipher[data.from].decryptPreKeyWhisperMessage(data.body.body, 'binary')
            } else {
                                                    decrypted       =       await
sessionCipher[data.from].decryptWhisperMessage(data.body.body, 'binary')
            }
                renderMessage(data.from + ': ' + String.fromCharCode.apply(null, new
Uint8Array(decrypted)))
        })
        socket.on('userDisc', async username => {
            removeUser(username)
            renderMessage(username + ' disconnected')
            await store.removeSession(username  + '.' + deviceId)
            delete sessionCipher[username]
        })

        socket.on('error', error => {
            renderError('error: ' + error)
        })
    })()
```

src/chat/chat.js:

```
const jwt = require('jsonwebtoken')
const config = require('../config')
const User = require('./user')
const userContacts = require('./contacts');

module.exports = (io) => {
    io.use((socket, next) => {
        try {
            const token = socket.handshake.query.token
            decoded = jwt.verify(token, config.jwtKey)
            User.add(socket, decoded)
            next()
        } catch (error) {
            next(new Error('Please login to continue'))
        }

    })
    .on('connection', socket => {

        const broadcastEvent = eventName => {
            socket.on(eventName, data => {
                const user = User.findById(socket.id)
                socket.broadcast.to(user.room).emit(eventName, {
                    from: user.name,
                    body: data
                })
            })
        }
        const sendToEvent = eventName=> {
            socket.on(eventName, data => {
                socket.to(User.findByUsername(data.to).id).emit(eventName, {
                    from: User.findById(socket.id).name,
                    body: data.body,
                })
            })
        }

        const user = User.findById(socket.id)
        socket.join(user.room, () => {
```

```
            socket.emit('join', user.room)
        })

        socket.on('disconnecting', () => {
            const user = User.remove(socket.id)
            io.to(user.room).emit('userDisc', user.name)
        })
        broadcastEvent('newSession')
        sendToEvent('newSessionReplay')
        sendToEvent('message')
            })
}
```

src/chat/contacts.js:

```
path = require('path')
const sqlite3 = require('sqlite3').verbose();

const DB_PATH = "D:\\!KSENNYA\\учеба\\4курс\\4.2\\!дипломне проектування\\РОЗРОБКА\\EncryptoChat\\data\\mybase.db";
//const DB_PATH = '../data/db.sqlite | sqlite.db';
// Створення нового підключення до бази даних
const db = new sqlite3.Database(DB_PATH);
function getUserName(userId)
{
  return new Promise((resolve, reject) => {
    const sql = 'SELECT * FROM Users WHERE id = ?';
    db.get(sql, [userId], (err, row) => {
      if (err) {
        console.error('Error checking user existence:', err);
        reject(err);
      } else {
        const username = row.username;
        resolve(username);
      }
    });
  });
}
```

```
function checkExistingRow(userId, contactId ) {
  return new Promise((resolve, reject) => {
    const sql = 'SELECT * FROM UserContacts WHERE user_id = ? AND contact_id = ?';

    db.get(sql, [contactId, userId], (err, row) => {
      if (err) {
        console.error('Error checking existing row:', err);
        reject(err);
      } else {
        if (row) {
          // Row exists
          resolve(true);
        } else {
          // Row does not exist
          resolve(false);
        }
      }
    });
  });
}
// Додавання контакту
function addContact(userId, contactId, contactName) {
  return new Promise(async (resolve, reject) => {
    try {
      const exists = await checkExistingRow(userId, contactId);
      if (exists) {
      console.log('Contact already exists');
      resolve(null);
    } else {
     const sql = 'INSERT INTO UserContacts (user_id, contact_id, contact_name) VALUES
(?, ?, ?)';

    db.run(sql, [userId, contactId, contactName], function (err) {
      if (err) {
        console.error('Error adding contact:', err);
        reject(err);
      } else {
        console.log('Contact added successfully: room:', this.lastID);
        resolve(this.lastID);
```

```
        console.log("ВИВОДИМО ЩЕ РАЗ ТАБЛИЦЮ ПІСЛЯ ДОДАВАННЯ КОНТАКТУ:");

        const sql4 = 'SELECT * FROM UserContacts';
        db.all(sql4, function (err, rows) {
          if (err) {
            console.error('Error retrieving users:', err);

          } else {
            const contacts = rows.map(row => ({
              ident: row.ident,
              user_id: row.user_id,
              contact_id: row.contact_id,
              contact_name: row.contact_name

             }));
             console.log(contacts);
          }   });
      }
    });
  }
}

catch (error) {
  console.error('Error checking existing row:', error);
  reject(error);
}
  });
}
// функція для перевірки наявності користувача з вказаним ID
function checkUserExists(userId) {
    // Нижче наведено лише приклад коду
    return new Promise((resolve, reject) => {
      const sql = 'SELECT * FROM Users WHERE id = ?';
      db.get(sql, [userId], (err, row) => {
        if (err) {
          console.error('Error checking user existence:', err);
          reject(err);
        } else {
```

```
            const userExists = row ? true : false;
            resolve(userExists);
          }
        });
      });
    }
    // Видалення контакту
    function removeContact(userId, contactId) {
      const sql = 'DELETE FROM UserContacts WHERE user_id = ? AND contact_id = ?';
      return new Promise((resolve, reject) => {
        db.run(sql, [userId, contactId], function (err) {
          if (err) {
            console.error('Error removing contact:', err);
            reject(err);
          } else {
            console.log('Contact removed successfully');
            resolve();
          }});
      });
    }

    // Отримання списку контактів для користувача
    function getContacts(userId) {
      const sql = 'SELECT contact_id, contact_name FROM UserContacts WHERE user_id =
?';
      return new Promise((resolve, reject) => {
        db.all(sql, [userId], function (err, rows) {
          if (err) {
            console.error('Error retrieving contacts:', err);
            reject(err);
          } else {
            const contacts = rows.map(row => ({
              contact_id: row.contact_id,
              contact_name: row.contact_name
            }));
            resolve(contacts);
          }    });  });}
        function getRoomId(userId, contactInput) {
         const sql4 = 'SELECT * FROM UserContacts';
```

```
            db.all(sql4, function (err, rows) {
              if (err) {
                console.error('Error retrieving users:', err);
                } else {
                const contacts = rows.map(row => ({
                  ident: row.ident,
                  user_id: row.user_id,
                  contact_id: row.contact_id,
                  contact_name: row.contact_name
                }));
                 console.log(contacts);
              }
            });
       //перевіряємо чи контакти вже знайомі якщо ні то додаємо запис

          const contactId = parseInt(contactInput);
          console.log("робимо 2-й запит в getRoomId:");
          return new Promise((resolve, reject) => {
            const sql = ('SELECT ident FROM UserContacts WHERE (user_id = ? AND contact_id
= ?) OR (user_id = ? AND contact_id = ?) ')
             db.get(sql, [userId, contactId, contactId, userId], function (err, row) {
              if (err) {
                console.error('Error retrieving room ID:', err);
                reject(err);
              } else {
                if (row) {
                  console.log("row.ident: ", row.ident);
                  resolve(row.ident);
                } else {
                  resolve(null); // Контакт не знайдений
                }        }      });    });  }
      module.exports = {
        getUserName,
        addContact,
        removeContact,
        getContacts,
        checkUserExists,
        getRoomId
      };
```

src/chat/user.js:

```
const users = []
const findByUsername = username => users.find(user => user.name === username)
const findById = socketId => users.find(user => user.id === socketId)
const findByRoom = room => users.filter(user => user.room === room)

const renameIfExists = username => {
    let uname = username
    let counter = 1
    while (findByUsername(uname)) {
        uname = `${username} (${counter})`
        counter++
    }
    return uname
}
const add = (socket, token) => {
    const user = {
        id: socket.id,
        name: token.username,
        room: token.roomname,
    }
    user.name = renameIfExists(user.name)
      users.push(user)
}
const remove = socketId => {
    const index = users.findIndex(user => user.id === socketId)
    return users.splice(index, 1)[0]}
let db;
function init(database) {
    db = database;
  }
function registerUser(username, publicKey) {
    const sql = 'INSERT INTO Users (username, publicKey) VALUES (?, ?)';
    db.run(sql, [username, publicKey], (err) => {
      if (err) {
        console.error('Error registering user:', err);
      } else {
```

```
            console.log('User registered successfully');
          }
        });
      }
      function getPublicKey(username) {
        return new Promise((resolve, reject) => {
            const sql = 'SELECT * FROM Users WHERE username = ?';
            db.get(sql, [username], (err, row) => {
                if (err) {
                    console.error('Error:', err);
                    reject(err);
                } else {
                    if (row) {
                       const publicKey = row.publicKey; // Отримання userId з результату
запиту
                        console.log('User publicKey:', username, publicKey);
                        resolve(publicKey); // Передача publicKey у виклик resolve
                    } else {
                        console.log(' failed for user:', username);
                        resolve(null);
                    }            }
            });
        });
    }function getUserId(username) {
        return new Promise((resolve, reject) => {
            const sql = 'SELECT * FROM Users WHERE username = ?';
            db.get(sql, [username], (err, row) => {
                if (err) {
                    console.error('Error:', err);
                    reject(err);
                } else {
                    if (row) {
                        const userId = row.id; // Отримання userId з результату запиту
                        resolve(userId); // Передача publicKey у виклик resolve
                    } else {
                        console.log(' failed for user:', username);
                        resolve(null);
                    }
                }
```

```
        });
    });
  function authenticateUser(username, password) {
    return new Promise((resolve, reject) => {
        const sql = 'SELECT * FROM Users WHERE username = ? AND password = ?';
        db.get(sql, [username, password], (err, row) => {
            if (err) {
                console.error('Error authenticating user:', err);
                reject(err);
            } else {
                if (row) {
                    const userId = row.id; // Отримання userId з результату запиту
                    console.log('User authenticated:', username, userId);
                    resolve(userId); // Передача userId у виклик resolve
                } else {
                    console.log('Authentication failed for user:', username);
                    resolve(null);
                }
            }});
    });
}

module.exports = {
    findByUsername,
    findById,
    findByRoom,
    add,
    remove,
    //
    init,
    registerUser,
    authenticateUser,
    getPublicKey,
    getUserId,
}
```