КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «Програмна розробка захищеного Web-мессенджера для обміну повідомленнями в реальному часі»

ПОЛОТНЯК ЄВГЕНІЙ ГЕННАДІЙОВИЧ

(прізвище, ім’я та по-батькові Здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
д. т. н., професор Гончаренко Т. А.

_____
„___”_____202_ року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «Програмна розробка захищеного Web-месенджера для обміну повідомленнями в реальному часі»

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Полотняк Євгеній Геннадійович
(прізвище, ім’я та по-батькові повністю)
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21-1
Керівник Поплавський О. А.
(прізвище та ініціали)
доктор технічних наук, доцент
(вчене звання, науковий ступінь)
Рецензент Гуменний Д. О.
(Прізвище та ініціали)
кандидат технічних наук, доцент
(вчене звання, науковий ступінь)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій
Випускова кафедра інформаційних технологій
Ступінь вищої освіти: «бакалавр»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: Інформаційні управляючі системи і технології

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
д. т. н., професор Гончаренко Т. А.
_____
„___"_____202_ року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Полотняк Євгеній Геннадійовича
(прізвище, ім'я та по-батькові Здобувача)

1. Тема роботи: Програмна розробка захищеного Web-мессенджера для обміну повідомленнями в реальному часі
затверджена наказом ректора КНУБА № 235 від «14» лютого 2025 року
2. Керівник роботи Поплавський Олександр Анатолійович, д. т. н., доцент кафедри інформаційних технологій
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)
3. Термін подання Здобувачем роботи до захисту_____
4. Зміст пояснювальної записки за розділами:
Р.1 Аналіз сучасних засобів безпечного обміну повідомленнями
Р.2 Проєктування та архітектурні рішення захищеного Web-мессенджера
Р.3 Програмна реалізація та тестування Web-мессенджера
Р.4 Ергономічне забезпечення розробки Web-мессенджера

5. Графічний матеріал за розділами:

Р.1 Слайди №3 – №4

Р.2 Слайди №5 – №6

Р.3 Слайди №7 – №15

Р.4 Слайди №16 – №18

6. Консультанти розділів кваліфікаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Гончаренко Т. А.	13.02.2025	
Розділ 2	Поплавський О. А.	15.04.2025	
Розділ 3	Мацієвський О. О.	22.05.2025	
Розділ 4	Рябчун Ю. В.	23.05.2025	

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	13.02.2025
Розділ 2	15.04.2025
Розділ 3	15.05.2025
Розділ 4	23.05.2025
Остаточне оформлення роботи	26.05.2025
Направлення роботи для перевірки на плагіат	26.05.2025
Попередній захист роботи на випусковій кафедрі	27.05.2025
Направлення роботи на рецензування	28.05.2025

8. Дата видачі завдання ___14 лютого 2025 року___

Зав. кафедри _____ Гончаренко Т. А
(підпис) (прізвище та ініціали)

Керівник _____ Поплавський О. А.
(підпис) (прізвище та ініціали)

Здобувач _____ Полотняк Є. Г.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускної роботи Здобувача:*	Полотняк Євгеній Геннадійович Yevhenii Polotniak		
ЗВО	Київський національний університет будівництва і архітектури		
Тема (українською та англійською)	Програмна розробка захищеного Web-мессенджера для обміну повідомленнями в реальному часі Software development of a secure web messenger for real-time messaging		
Освітній ступінь	Бакалавр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Інформаційних технологій		
Спеціальність	122 «Комп'ютерні науки»		
Освітня програма	Інформаційні управляючі системи і технології		
Керівник	Поплавський Олександр Анатолійович		
Обсяг роботи:	пояснювальна записка, стор.	розділів	креслень формату А
	155	4	-
Розділ 1.	Аналіз сучасних засобів безпечного обміну повідомленнями		
Розділ 2.	Проєктування та архітектурні рішення захищеного Web-мессенджера		
Розділ 3.	Програмна реалізація та тестування Web-мессенджера		
Розділ 4.	Ергономічне забезпечення розробки Web-мессенджера		
Ключові слова: *Keywords:*	Web-мессенджер, шифрування, конфіденційність, користувач, груповий чат, повідомлення, сервер, клієнт-серверна архітектура, вебсокет, база даних. Web messenger, encryption, confidentiality, user, group chat, message, server, client-server architecture, websocket, database.		

Здобувач: _____ /Євгеній ПОЛОТНЯК /

Керівник: _____/ Олександр ПОПЛАВСЬКИЙ /

"___" _____ 202_р.

АНОТАЦІЯ

Полотняк Є. Г. Програмна розробка захищеного Web-мессенджера для обміну повідомленнями в реальному часі.

Кваліфікаційна випускна робота бакалавра за спеціальністю: 122 «Комп'ютерні науки», освітньо-професійна програма: «Інформаційні управляючі системи і технології». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Робота присвячена створенню захищеного Web-мессенджера для обміну повідомленнями в реальному часі з використанням наскрізного шифрування (E2EE) на основі алгоритму Double Ratchet. Система має клієнт-серверну архітектуру: клієнт шифрує повідомлення перед відправленням, сервер зберігає їх у зашифрованому вигляді. Користувачі можуть обмінюватися повідомленнями в групових чатах, а автори чатів змінювати їхні назви або видаляти. Тестування підтвердило ефективність програмного забезпечення.

Ключові слова: Web-мессенджер, шифрування, конфіденційність, користувач, груповий чат, повідомлення, сервер, клієнт-серверна архітектура, вебсокет, база даних.

SUMMARY

Polotniak Y. H. Software development of a secure web messenger for real-time messaging.

Bachelor's thesis for a bachelor's degree in specialty: 122 “Computer Science”, specialization: "Information Management Systems and Technologies" - Kyiv National University of Construction and Architecture - Kyiv, 2025.

The work focuses on creating a secure web messenger for real-time messaging using end-to-end encryption (E2EE) with the Double Ratchet algorithm. The system is based on a client-server architecture: the client encrypts messages before sending, and the server stores them in encrypted form. Users can exchange messages in group chats, and chat authors can change or delete them. Testing confirmed the software's effectiveness.

Keywords: web messenger, encryption, confidentiality, user, group chat, message, server, client-server architecture, websocket, database.

ЗМІСТ

ВСТУП

У сучасному інформаційному середовищі обмін повідомленнями став не лише засобом повсякденної комунікації, а й критично важливим компонентом бізнес-процесів, управління, освіти, волонтерських та військових ініціатив. Однак зростання ролі цифрових каналів супроводжується відповідним зростанням кіберзагроз, витоків даних та атак на приватність. Традиційні мессенджери, хоч і забезпечують базовий захист, не завжди гарантують цілісність та конфіденційність повідомлень у повному обсязі, особливо у web-only версіях.

З огляду на це, **актуальність теми дипломної роботи обумовлена потребою у створенні сучасного, безпечного, зручного у використанні Web-мессенджера**, який не вимагав би встановлення стороннього ПЗ, проте забезпечував високий рівень шифрування та повну незалежність від зовнішніх платформ. У критичних умовах — зокрема в контексті кібербезпеки України — такі інструменти можуть мати стратегічне значення як у громадянському секторі, так і в державному управлінні, освіті чи обороні.

Метою дипломної роботи є *створення захищеного Web-мессенджера для обміну повідомленнями в реальному часі*, який би поєднував високу безпеку з інтуїтивною ергономікою та доступністю з будь-якого пристрою через браузер.

Необхідність розробки обумовлена низкою факторів:

- Наявність компромісів у безпеці популярних мессенджерів (WhatsApp, Telegram, Viber тощо);
- Відсутність дійсно *web-only* рішення з підтримкою *end-to-end* шифрування для групових чатів;
- Зростаюча актуальність інформаційної безпеки в умовах воєнного часу в Україні.

Об'єктом дослідження є *інформаційна система обміну повідомленнями в реальному часі*.

Предмет дослідження — *методи захисту даних у клієнт-серверній архітектурі Web-мессенджера*.

Основні методи дослідження: критичний аналіз існуючих рішень, об'єктно-орієнтоване моделювання, прототипування, модульна реалізація, тестування, застосування принципів Double Ratchet шифрування, а також використання технологій Docker та WebSocket.

Короткий зміст основних розділів

У першому розділі проаналізовано сучасні засоби безпечного обміну повідомленнями, розглянуто функціональні та криптографічні особливості популярних мессенджерів, визначено їх недоліки, що обґрунтовують доцільність створення власного рішення.

У другому розділі виконано повне архітектурне проєктування системи: визначено методологію розробки, інформаційну модель, архітектуру, принципи шифрування та побудовано DFD-діаграми. Визначено шляхи забезпечення конфіденційності, масштабованості та зручності використання.

У третьому розділі реалізовано клієнтську частину на React + TypeScript та серверну частину на Node.js з WebSocket та PostgreSQL. Розробку виконано з урахуванням Docker-контейнеризації, розгортання на AWS, забезпечення HTTPS-захисту та наскрізного шифрування повідомлень.

У четвертому розділі акцент зроблено на ергономічному забезпеченні: реалізовано адаптивний інтерфейс, протестовано зручність користування на різних пристроях, сформовано логіку візуальних компонентів. Також описано основні технічні, інтерфейсні та хостингові проблеми, що виникали під час розробки, та шляхи їх усунення.

Галузі застосування результатів

- У малих та середніх підприємствах для внутрішньої комунікації;
- В освітніх закладах для безпечного спілкування студентів та викладачів;
- У громадських організаціях і волонтерських ініціативах, де особливо важлива приватність;
- У проєктах, які потребують web-only рішення без встановлення ПЗ.

1. АНАЛІЗ СУЧАСНИХ ЗАСОБІВ БЕЗПЕЧНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ

1.1 Загальна характеристика Web-мессенджерів

Спілкування є фундаментальною потребою людини та основою соціальної взаємодії. З часом засоби передачі інформації еволюціонували: від усної мови та жестів до сучасних цифрових технологій.

Початково люди передавали повідомлення голосом, жестами, димовими сигналами чи наскельними малюнками. Із розвитком суспільств з'явилися піктографія та писемність, що дозволило передавати інформацію на відстані. Значним проривом стали поштові системи у стародавніх цивілізаціях. Згодом друкарство, телеграф, телефон, а потім радіо та телебачення суттєво прискорили й розширили комунікацію. Інтернет відкрив нову епоху — цифрову [1].

Соціальні платформи — це онлайн-простори для спілкування та обміну інформацією. Термін *мессенджер* походить від англ. *Instant Messenger* — «миттєвий кур'єр» [4].

З появою інтернету з'явилися перші сервіси миттєвих повідомлень — ICQ (1996) та AOL Instant Messenger (1997). У 2000-х роках із розвитком мобільних технологій з'явився Skype (2003), а пізніше — WhatsApp (2009), Telegram (2013) та Signal (2014) [3]. Вони поєднують текстові чати, передачу файлів, дзвінки та інше.

Сучасні мессенджери вирізняються такими характеристиками:

- Миттєвість — обмін повідомленнями без затримки;
- Доступність — безкоштовне або умовно безкоштовне використання;
- Мультиплатформеність — підтримка різних пристроїв;
- Безпека — наскрізне шифрування, двоетапна автентифікація;
- Мультимедійність — підтримка дзвінків, файлів, відео;
- Групове спілкування — чати, канали, спільноти;
- Інтеграція — зв'язок із банками, ботами, сервісами [17].

Огляд підготовлено на основі дослідження Digital 2025 Ukraine від платформи DataReportal, що щорічно публікує глобальні й національні цифрові звіти.

Станом на лютий 2025 року у світі налічується 5,56 млрд інтернет-користувачів (67,9%) та 5,24 млрд користувачів соцмереж (63,9%). Мобільних підключень — 8,78 млрд, що на 7% перевищує чисельність населення (рис. 1.1)[2].

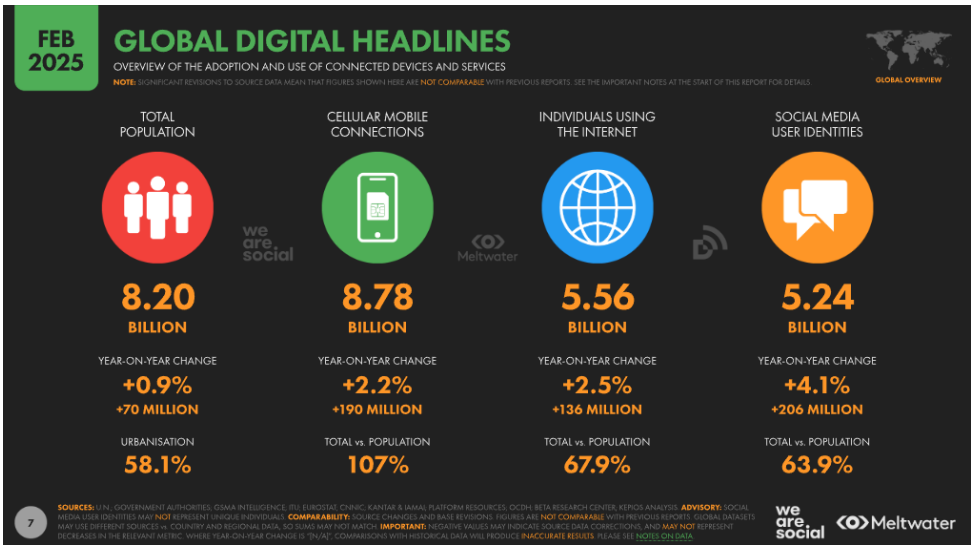


Рис. 1.1. Глобальні цифрові показники станом на лютий 2025 року

В Україні — 31,5 млн інтернет-користувачів (82,4%) і 21,6 млн користувачів соцмереж (56,4%). Мобільних підключень — 56,4 млн, або 147% до кількості населення. Спостерігається незначне скорочення мобільних зв'язків порівняно з минулим роком (рис. 1.2)[2].

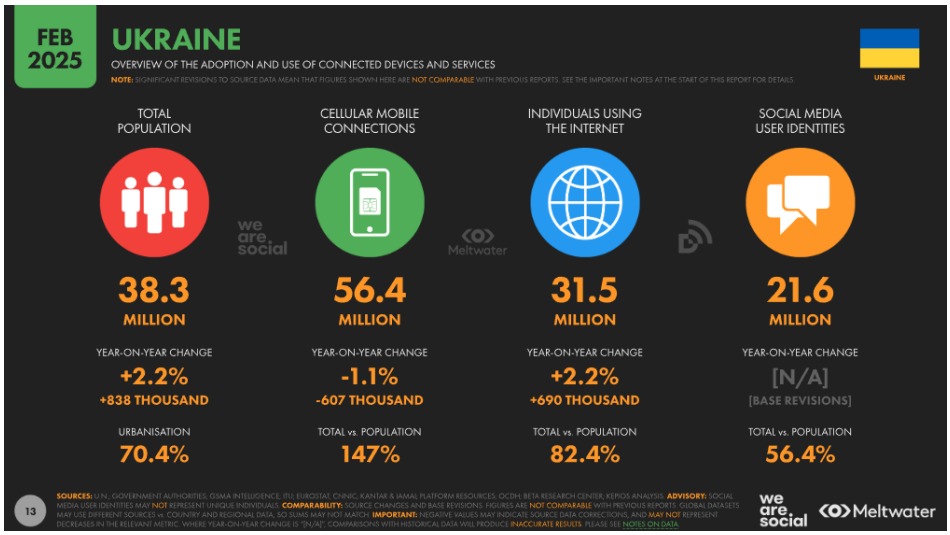


Рис. 1.2. Цифрові показники в Україні станом на лютий 2025 року

Соцмережами користується 68,5% інтернет-користувачів та 61,5% дорослого населення. Жінки становлять 51,9% користувачів соцмереж, чоловіки — 48,1%. Кількість обліковок залишається стабільною (рис. 1.3)[2].

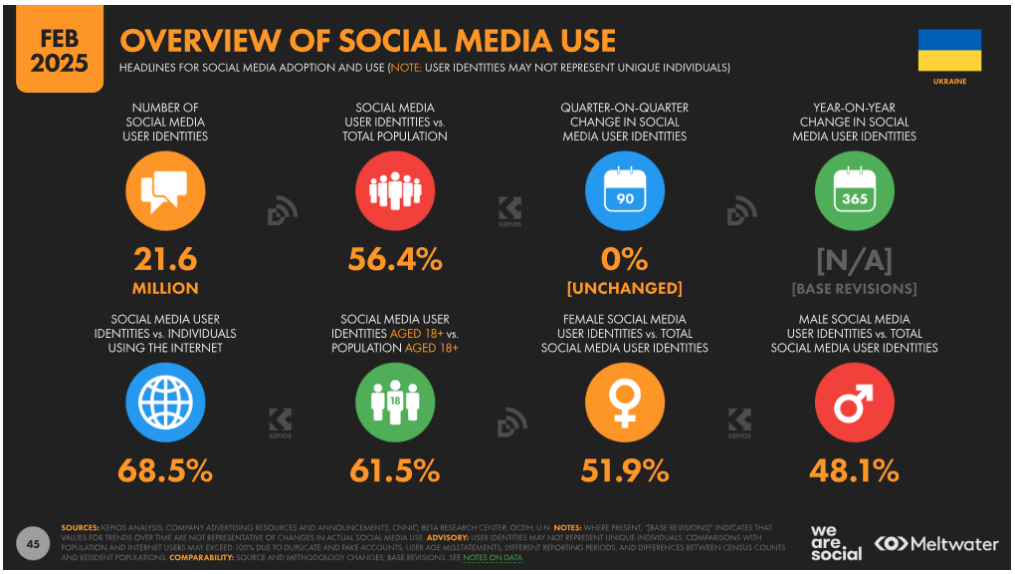


Рис. 1.3. Статистика використання соціальних мереж в Україні

Станом на 2025 рік мессенджери продовжують набирати популярності (рис. 1.4). Найшвидше зростає Telegram, кількість його завантажень перевищує 50 млн щомісяця [3].

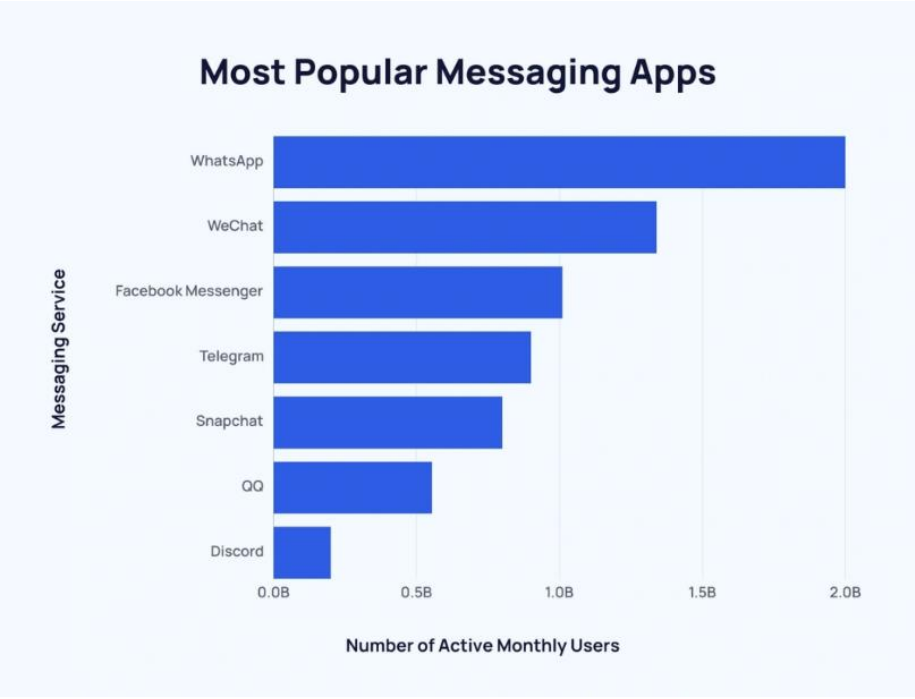


Рис. 1.4. Рейтинг найпопулярніших мессенджерів станом на 2025 рік

Станом на червень 2020 року найпопулярнішими мессенджерами в Україні були Viber, WhatsApp, Facebook Messenger і Telegram, якими щодня користувалися близько 76% українців (рис. 1.5). Особливо помітне зростання показав Telegram, який з 2017 року обігнав WhatsApp і майже зрівнявся з FB Messenger, досягнувши 5,6 млн користувачів [4].

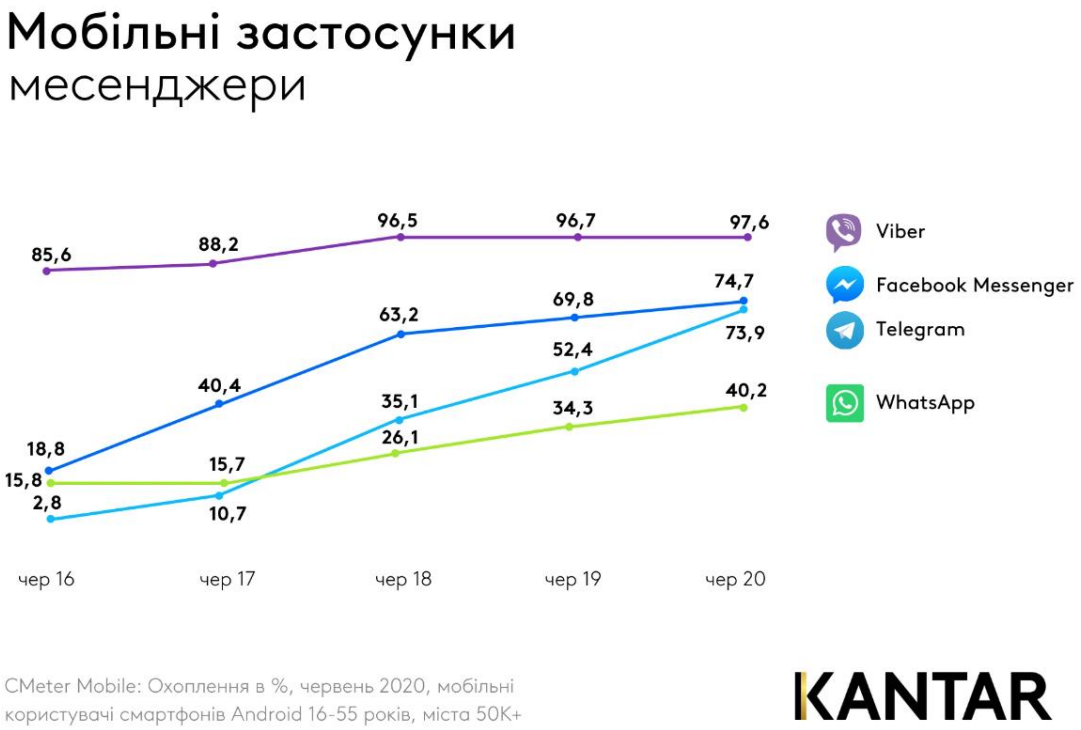


Рис. 1.5. Популярність мобільних мессенджерів в Україні (2016–2020), KANTAR

Еволюція засобів комунікації — від усної мови до високотехнологічних Web-мессенджерів — докорінно змінила взаємодію між людьми. Сучасні мессенджери вже давно вийшли за межі особистого спілкування і стали ключовим інструментом у бізнесі, освіті, держуправлінні та медицині. Вони забезпечують можливість віддаленої роботи, онлайн-зустрічей, передачі файлів, здійснення оплат та швидкого доступу до інформації.

Водночас масове використання Web-мессенджерів супроводжується новими ризиками: зростає потреба в захисті даних, боротьбі з кіберзагрозами, дезінформацією та збереженні конфіденційності. Щоб протистояти цим викликам, розробники постійно впроваджують сучасні алгоритми шифрування, автентифікації та безпечного зберігання даних.

1.2 Огляд сучасних мессенджерів

Розглянемо популярні мессенджери, такі як **WhatsApp, Telegram, Viber, Session** та **Signal**. Додатки типу WeChat і QQ Messenger, хоч і мають величезну аудиторію в Китаї, є локальними ринками і не відповідають вимогам для аналізу у контексті Європи чи Америки, тому їх розглядати не будемо.

Майже всі сучасні мессенджери мають власну Web-версію, що дозволяє користувачам використовувати їхній програмний продукт безпосередньо через браузер, без необхідності завантаження додатку на пристрій. Це значно спрощує доступ до сервісів, оскільки користувачі можуть залишатися на зв'язку як через смартфони, так і через комп'ютери, не встановлюючи додаткове програмне забезпечення.

WhatsApp відомий своїм простим інтерфейсом і високим рівнем захисту завдяки наскрізному шифруванню, що гарантує конфіденційність спілкування.

Telegram позиціонується як сервіс для тих, хто цінує швидкість та розширені можливості: він пропонує створення великих групових чатів, канали для масового поширення інформації, інтегрованих ботів та секретних чатів із додатковими шарами захисту.

Viber здобув популярність завдяки можливості здійснювати якісні голосові та відеодзвінки, що робить його зручним інструментом як для особистого, так і для міжнародного спілкування.

Session представляє собою мессенджер, орієнтований на забезпечення анонімності та високого рівня конфіденційності.

Нарешті, **Signal** займає особливе місце завдяки своїй орієнтації на безпеку. Використовуючи відкритий вихідний код і передові алгоритми шифрування, Signal гарантує, що дані користувачів залишаються захищеними від стороннього доступу, що робить його вибором номер один для користувачів із високими вимогами до конфіденційності.

1.2.1 Telegram

Telegram — швидкий і безпечний мессенджер із простим інтерфейсом та повною безкоштовністю. Сервіс підтримує одночасну роботу на кількох пристроях із автоматичною синхронізацією через Web-версію — без потреби встановлення додаткового ПЗ [5].

Користувачі можуть надсилати текстові повідомлення, фото, відео та файли будь-якого формату до 2 ГБ. Доступні групові чати до 200 тис. учасників і канали з необмеженою аудиторією. Зв'язок можливий як із контактами з телефонної книги, так і за унікальним ім'ям користувача, що поєднує функції SMS та електронної пошти [5, 16].

Telegram підтримує наскрізне шифрування для дзвінків та голосові чати з тисячами учасників, що підвищує безпеку й зручність [5].

Платформа також є відкритою: доступні API і вихідний код для розробки власних застосунків та інтеграції сервісів. Бот-API дозволяє автоматизувати дії, взаємодіяти зі сторонніми платформами та приймати платежі (рис. 1.5) [5, 16].

Рис. 1.5. Переваги та функціональна універсальність Telegram

У Telegram можна встановити публічне ім’я користувача, яке дозволяє знаходити вас через глобальний пошук. У цьому разі інші користувачі можуть надсилати вам повідомлення без знання номера телефону. Якщо це небажано — краще не задавати публічне ім’я [5].

Щоб видалити ім’я, потрібно зайти в «Налаштування» та очистити відповідне поле — після цього вас не буде видно в пошуку, хоча існуючі чати залишаться.

Приклад мобільного інтерфейсу Telegram: список чатів, відеодзвінки, можливість обмінюватися фото, відео, файлами та емоджи (рис. 1.7).

Рис. 1.7. Приклад мобільного інтерфейу Telegram

У Telegram для захисту повідомлень використовується протокол MTProto, який реалізує два рівні шифрування: клієнт-серверне (у хмарних чатах) та наскрізне (у секретних чатах). MTProto базується на поєднанні AES у режимі IGE, протоколу Діффі-Геллмана для обміну 2048-бітними RSA-ключами та хеш-функцій. Також підтримується end-to-end шифрування з можливістю перевірки ключів [5].

На відміну від WhatsApp, де наскрізне шифрування вмикається за замовчуванням у всіх чатах, Telegram використовує його лише в секретних чатах. Питання безпеки даних у хмарних чатах залишається дискусійним [18].

У розділі «Налаштування» також можна змінити номер телефону або фото профілю (рис. 1.6). У вкладці «Приватність та безпека» доступне автоматичне видалення акаунта — якщо користувач неактивний протягом обраного періоду (1–24 місяці), обліковий запис буде повністю видалено разом з усіма даними (рис. 1.6).

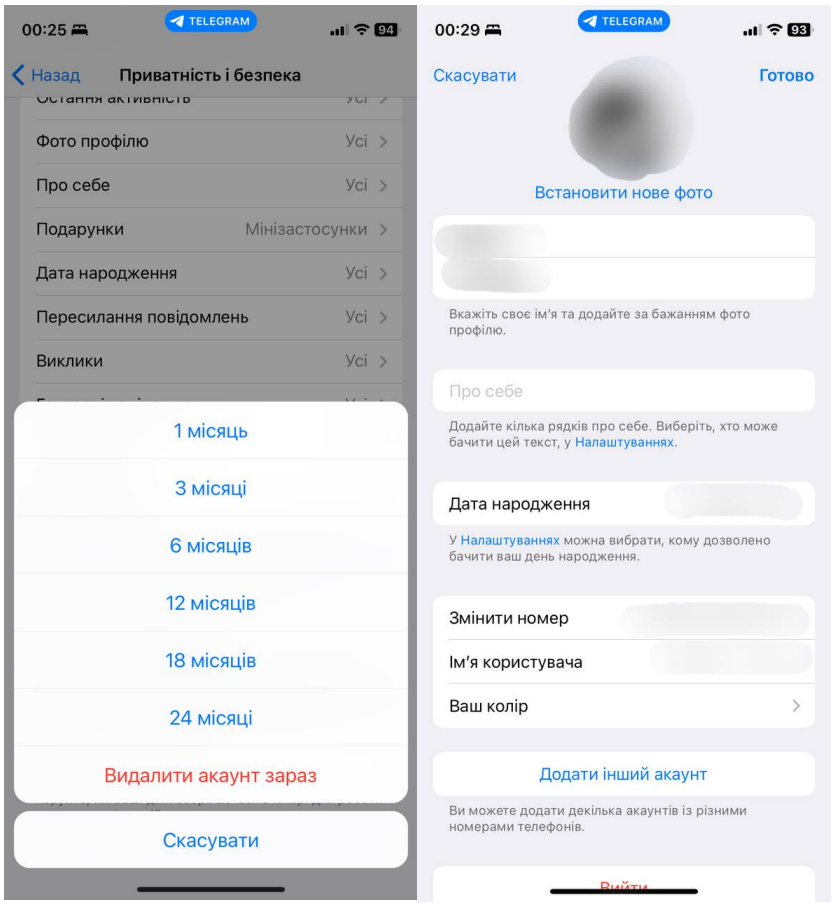


Рис. 1.6. Налаштування автовидалення профілю та особистої інформації

Окрім цього, існує Web-версія Telegram Web, яка дозволяє користуватись мессенджером без встановлення додатку [6] (рис. 1.8).

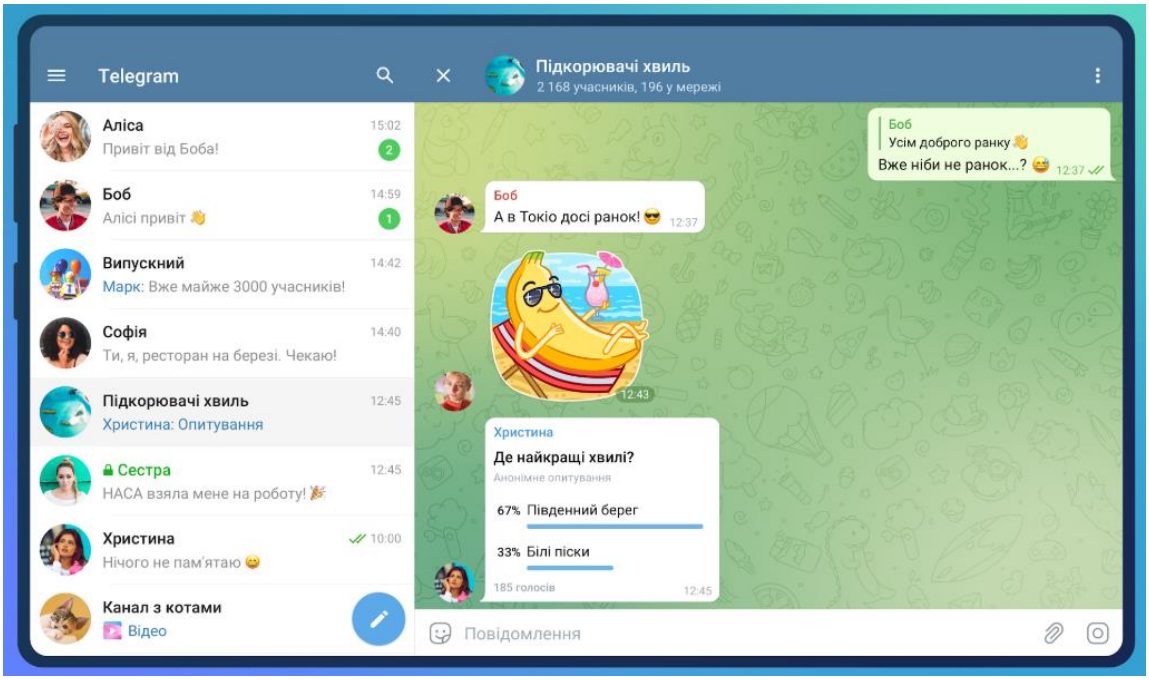


Рис. 1.8. Приклад інтерфейсу Telegram Web

У таблиці нижче наведено основні переваги Telegram Web, які роблять цю версію корисною для повсякденного використання (табл. 1.1):

Таблиця 1.1

Переваги Telegram Web

№	Перевага
1	Можна користуватись без встановлення додатку — доступ через будь-який браузер.
2	Не займає місце на комп'ютері — всі дані зберігаються в хмарі.
3	Зручний у використанні — інтерфейс схожий на десктопну та мобільну версії.

У наступній таблиці подано ключові недоліки Web-версії Telegram, які варто враховувати при її використанні (табл. 1.2):

Таблиця 1.2

Недоліки Telegram Web

№	Недолік
1	Потребує стабільного та швидкого інтернет-з'єднання.
2	Оптимізований переважно для використання на комп'ютерах, не для мобільних браузерів.

Нижче в таблиці подано основні переваги Telegram, що забезпечують його популярність серед мільйонів користувачів у всьому світі (табл. 1.3):

Таблиця 1.3

Переваги Telegram:

№	Перевага
1	Можливість надсилати файли розміром до 2 ГБ.
2	Секретні чати з наскрізним шифруванням і самознищенням повідомлень.
3	Інтуїтивно зрозумілий інтерфейс.
4	Висока швидкість передачі даних завдяки оптимізованій інфраструктурі.
5	Хмарне зберігання — доступ до даних із будь-якого пристрою.
6	Велика кількість безкоштовних стікерів.

Продовження таблиці 1.3

7	Підтримка голосових і відеодзвінків.
8	Гнучке управління чатами (папки, закріплення, сортування).
9	Web-версія, яка не потребує інсталяції.

У таблиці нижче наведено недоліки Telegram, які пов’язані переважно з аспектами безпеки та конфіденційності (табл. 1.4):

Таблиця 1.4

Недоліки Telegram

№	Недолік
1	Велика кількість чатів може спричинити плутанину.
2	Збір метаданих користувачів.
3	Зберігання повідомлень на серверах (крім секретних чатів).
4	Зафіксовано випадки зламів акаунтів та витоків даних.
5	End-to-end шифрування використовується лише в секретних чатах.
6	MTProto — закритий протокол, що викликає сумніви в експертів з кібербезпеки.
7	Дані користувача зберігаються до 12 місяців після видалення акаунту.
8	Можливість несанкціонованого доступу через функції, як-от «Люди поруч».
9	Telegram не рекомендується для передавання чутливої або службової інформації.

Telegram — це не мессенджер, що гарантує повну конфіденційність, а радше онлайн-сервіс і соціальна платформа з певними елементами анонімності, яка має як свої переваги, так і недоліки. Зафіксовано численні випадки несанкціонованого доступу до акаунтів та компрометації пристроїв користувачів через Telegram. Крім того, статистика використання та метадані пристроїв синхронізуються з серверами і зберігаються у базі даних протягом тривалого часу [18]. Тому не рекомендується пересилати через Telegram особисті, конфіденційні або службові дані. Для приватного, особистого листування існують більш захищені канали зв’язку.

1.2.2 WhatsApp

WhatsApp – це американський безкоштовний додаток, створений у 2009 році програмістом українського походження Яном Кумом та Браяном Ектоном, який був придбаний компанією Facebook, Inc. у 2014 році. Програма дозволяє користувачам надсилати текстові та голосові повідомлення, здійснювати голосові та відеодзвінки, а також обмінюватися зображеннями, документами, інформацією про місцезнаходження та іншими медіа [7].

Користувачі можуть створювати групові чати до 256 учасників для спілкування, організації подій чи обговорень, із можливістю налаштування сповіщень. Вбудована камера дозволяє робити фото та надсилати їх прямо з додатку. Максимальний розмір файлів для пересилання — 100 МБ. У деяких країнах доступна функція грошових переказів [7].

WhatsApp забезпечує конфіденційний обмін повідомленнями, підтримує ведення особистого бізнесу та пропонує широкий набір функцій (рис. 1.9) [10].

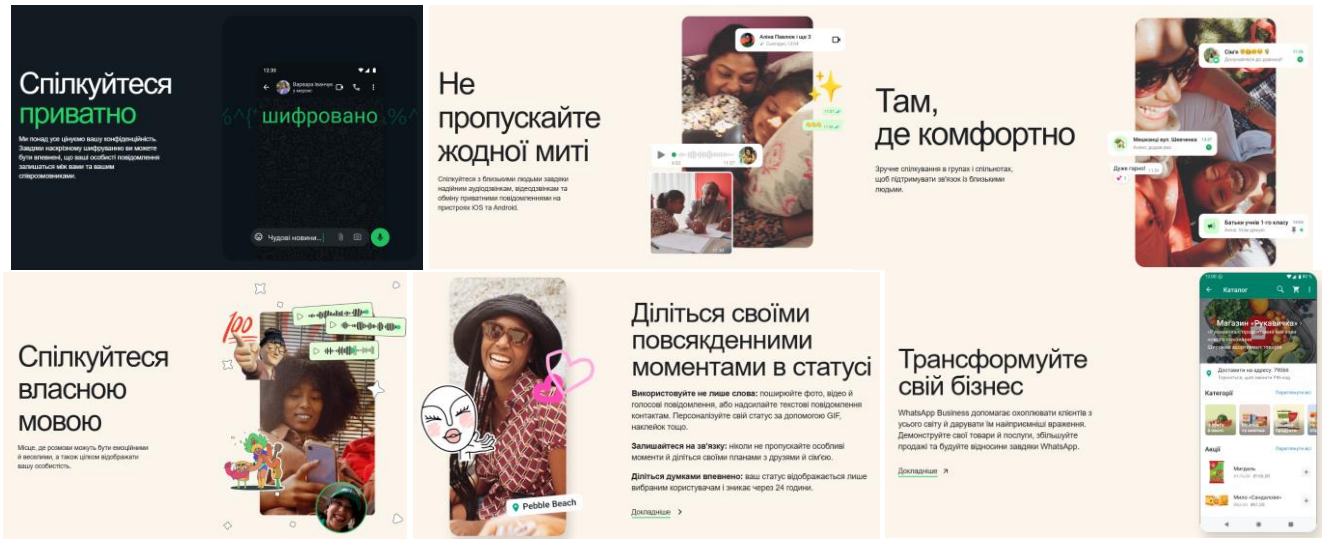


Рис. 1.9. Перелік основних можливостей WhatsApp

WhatsApp розроблений насамперед для мобільних пристроїв, однак доступний і на ПК за умови активного підключення телефону до Інтернету. Реєстрація відбувається за номером телефону з підтвердженням через SMS — як і в більшості подібних сервісів [8].

Інтерфейс мессенджера інтуїтивно зрозумілий. Користувач може переглядати чати, здійснювати аудіо- та відеодзвінки, надсилати фото, відео, голосові повідомлення, реакції, стікери, геолокацію, а також створювати відеонататки (рис. 1.10) [7].

Рис. 1.10. Приклад мобільного інтерфейсу WhatsApp

WhatsApp використовує наскрізне шифрування на базі протоколу Signal, який також застосовується в однойменному захищеному мессенджері. Протокол ґрунтується на алгоритмі Double Ratchet, що комбінує AES-256, Діффі-Геллмана та систему оновлення ключів, забезпечуючи пряму і зворотну секретність [13, 14].

Ключі постійно змінюються й не пов'язані між собою, тому історія переписки не підлягає відновленню навіть у разі компрометації. Крім того, у WhatsApp реалізовано додаткові функції конфіденційності (рис. 1.11) [7].

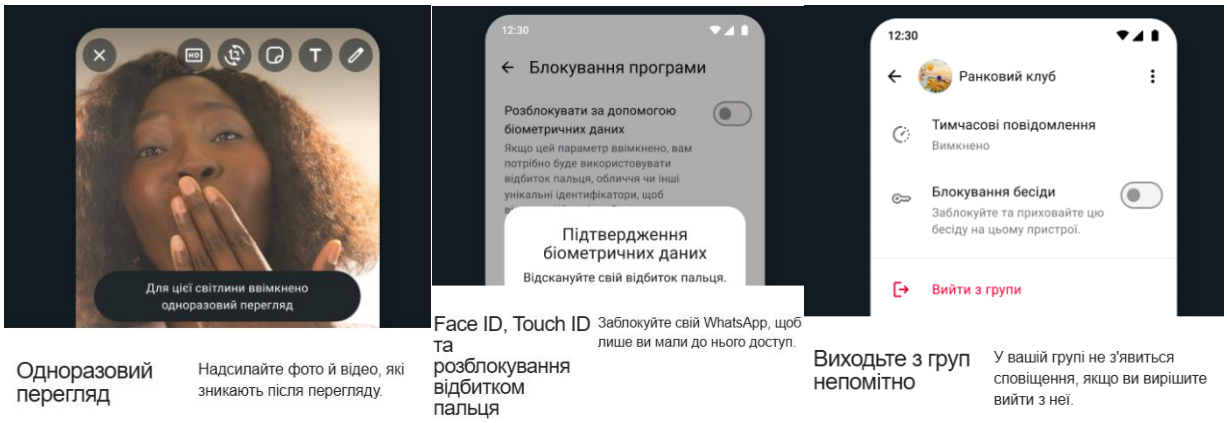


Рис. 1.11. Спеціалізовані функції конфіденційності WhatsApp

Мессенджер також пропонує базові функції конфіденційності: захист чатів паролем, тимчасові повідомлення, блокування викликів від невідомих контактів та керування статусами онлайн-активності (рис. 1.12) [7].

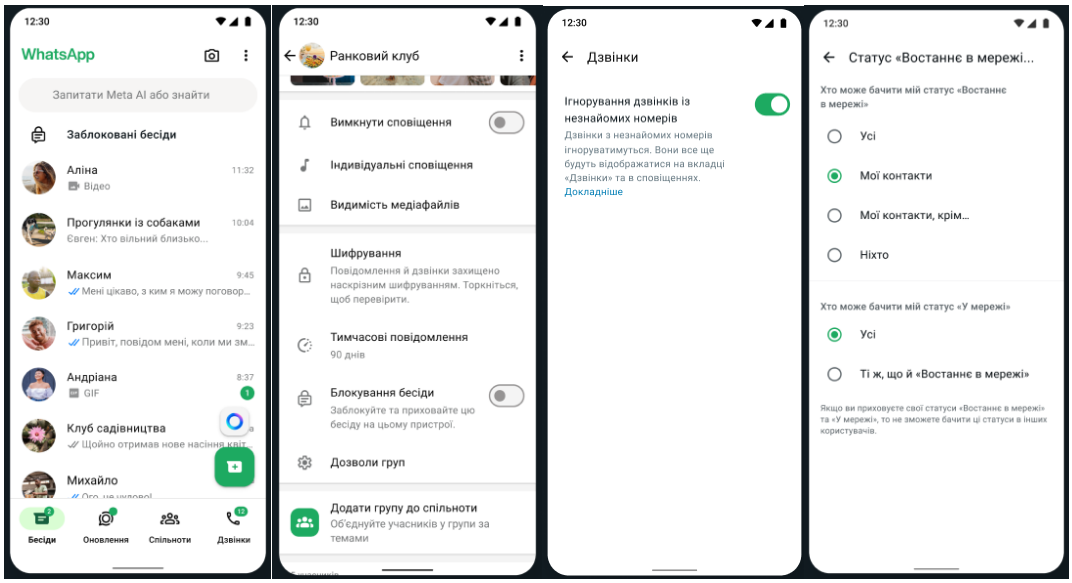


Рис. 1.12. Перелік основних базових функцій WhatsApp

WhatsApp Web дає змогу обмінюватися повідомленнями через браузер без встановлення програми. Великий екран забезпечує зручність роботи, а інтерфейс схожий на мобільний: ліворуч список чатів, праворуч — вміст повідомлень (рис. 1.13) [8, 9].

Попри зручність, Web-версія не має інструментів для ведення бізнесу, зокрема управління проєктами чи задачами (наприклад, Канбан-дошки) [10].

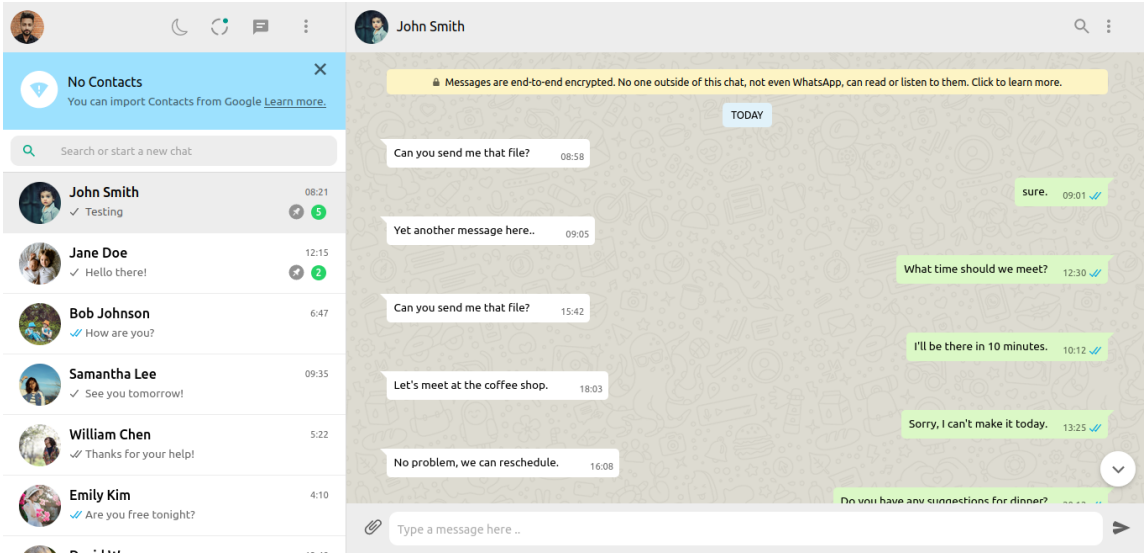


Рис. 1.13. Приклад інтерфейсу WhatsApp Web

Нижче в таблиці наведено переваги Web-версії WhatsApp (табл. 1.5):

Таблиця 1.5

Переваги WhatsApp Web

№	Перевага
1	Доступ через будь-який браузер без встановлення додатку.
2	Зручний інтерфейс із підтримкою комбінацій клавіш.
3	Можливість обміну файлами та мультимедіа.
4	Групові чати для командної співпраці.
5	Безкоштовне використання без реклами.

Наступна таблиця показує основні недоліки WhatsApp Web, які обмежують її використання (табл. 1.6):

Таблиця 1.6

Недоліки WhatsApp Web

№	Недолік
1	Залежність від смартфона — потрібне активне з’єднання.
2	Неможливість змінювати фото профілю чи статус.
3	Відсутність голосових та відеодзвінків.
4	Не підтримує інструментів для управління завданнями чи проектами.

У таблиці нижче наведено головні переваги WhatsApp, які забезпечили його популярність у всьому світі (табл. 1.7):

Таблиця 1.7

Переваги WhatsApp

№	Перевага
1	Підтримка текстових, голосових та відеоповідомлень.
2	Наскрізне шифрування за замовчуванням.
3	Інтуїтивний інтерфейс та проста реєстрація.
4	Автоматична синхронізація контактів.
5	Web-версія без інсталяції на ПК.
6	Групові чати та можливість створення спільнот.

У наступній таблиці подано основні недоліки WhatsApp, які викликають занепокоєння щодо приватності та безпеки (табл. 1.8):

Таблиця 1.8

Недоліки WhatsApp

№	Недолік
1	Залежність від смартфона для роботи на ПК.
2	Збір метаданих користувачів.
3	Закритий вихідний код.
4	Реєстрація лише за номером телефону.
5	Інтеграція з рекламною екосистемою Meta
6	Зафіксовані інциденти безпеки.

WhatsApp – це піонер у сфері інтернет-телекомунікацій, який спочатку робив ставку на захист даних завдяки наскрізному шифруванню. Він швидко здобув популярність і започаткував еру мобільних комунікацій. Однак після придбання компанією Meta акцент на кібербезпеці послаб, що спричинило низку інцидентів (зокрема, пов’язаних із Pegasus) через збір метаданих та рекламну інтеграцію. Попри це, додаток залишається популярним завдяки зручному інтерфейсу і численним корисним функціям, таким як двофакторна авторизація, наскрізне шифрування, автовидалення повідомлень, резервне копіювання з підтримкою E2EE і мультиакаунти [7]. Проте, з огляду на сучасні питання конфіденційності, не рекомендується використовувати WhatsApp для передачі особистої або конфіденційної інформації [18].

1.2.3 Rakuten Viber

Японська компанії Rakuten у 2017 році викупила Viber, він як і інші мессенджери, він дозволяє обмінюватися миттєвими повідомленнями, здійснювати дзвінки всередині мережі та створювати чати [11]. Реєстрація та ідентифікація користувачів здійснюється за номером телефону, хоча сервіс доступний і на десктопних платформах без потреби мобільного зв'язку [11].

Особливістю Viber є його сервіс Viber Out — платна функція, яка дозволяє користувачам здійснювати міжнародні дзвінки як на стаціонарні, так і на мобільні телефони. Крім того, Viber займає лідируючу позицію як найпопулярніший мессенджер в Україні та надає можливість проводити платежі [12]. Загалом, функціонал додатку є надзвичайно різноманітним і зручним (рис. 1.14).

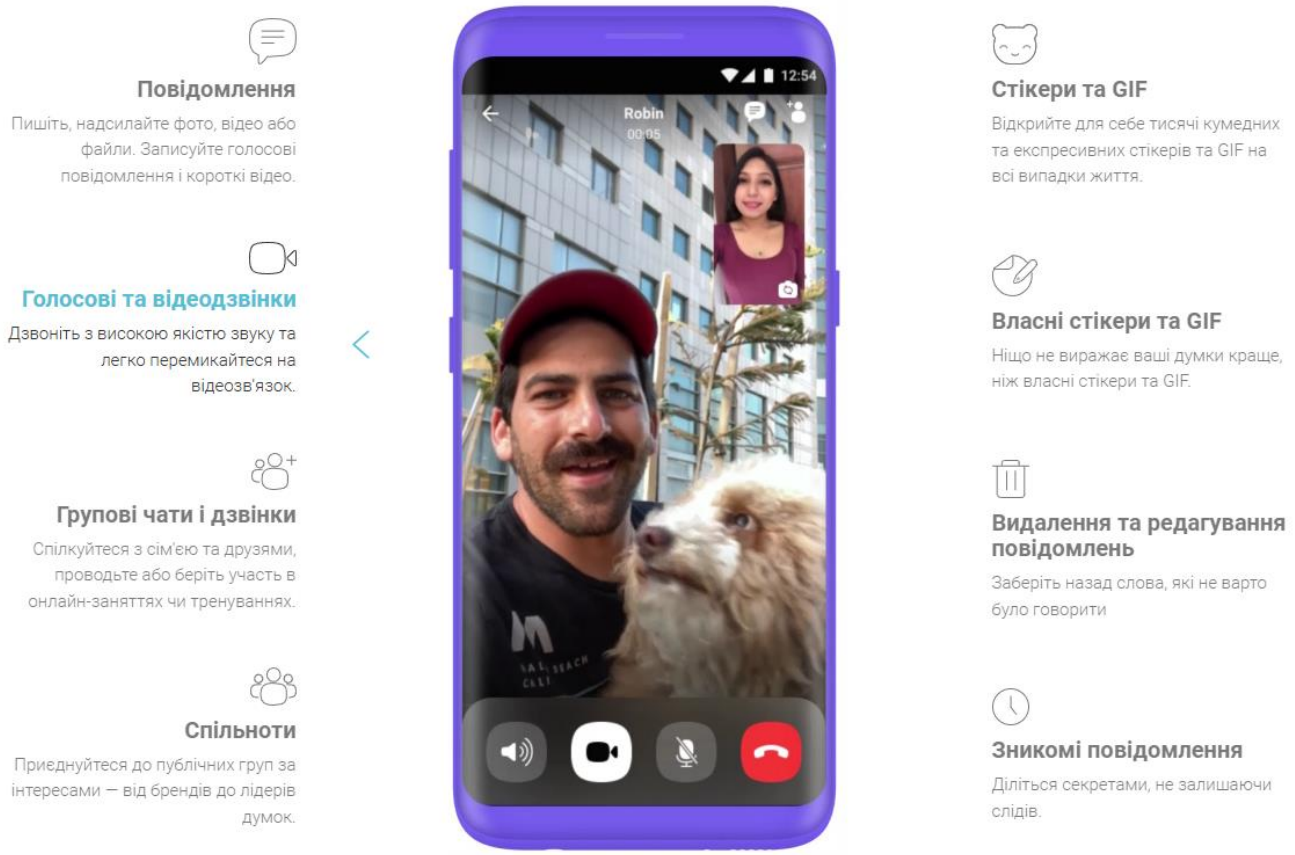


Рис. 1.14. Перелік основних можливостей Viber

Популярність мессенджера у світі, зокрема і в Україні, зумовлена його гнучким та потужним функціоналом у спільнотах. Адміністратор може налаштовувати ролі користувачів у чаті, призначати та змінювати модераторів, видаляти повідомлення. Додатково учасники спільноти можуть переглядати статистику, створювати повідомлення із автоматичним зникненням через заданий час (рис. 1.15) [11, 12].

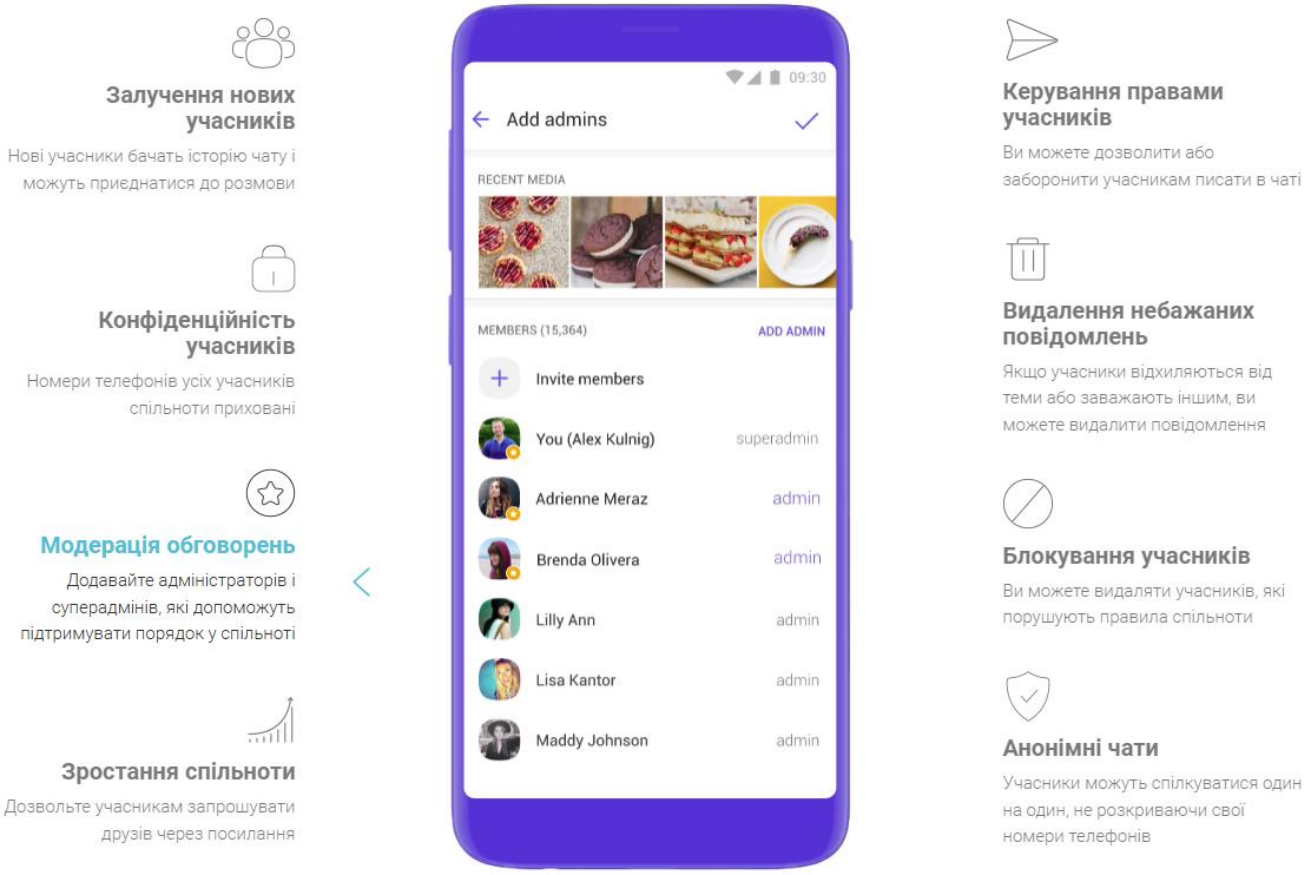


Рис. 1.15. Перелік гнучкого функціоналу спільнот Viber

Viber є безкоштовним сервісом, але містить рекламу та продає стікери. Через просту процедуру додавання в чат за номером телефону він схильний до спаму. Реєстрація відбувається аналогічно іншим мессенджерам — за номером. Меню навігації розташоване внизу та містить чати, виклики, новини, стікери, рекламу й розділ More, де знаходяться налаштування. Додаток має приємний та функціональний інтерфейс (рис. 1.16) [11, 12].

Viber забезпечує наскрізне шифрування для всіх чатів за замовчуванням, а секретні чати мають додаткові функції безпеки: автоматичне видалення повідомлень, захист від створення скріншотів, заборону копіювання та пересилання. Для резервного копіювання історії повідомлень використовується Google Drive. У своїй роботі мессенджер застосовує протокол Proteus, що є модифікацією Signal із власними криптографічними механізмами. Інтерфейс мессенджера є досить приємним та ергономічним [17, 18].

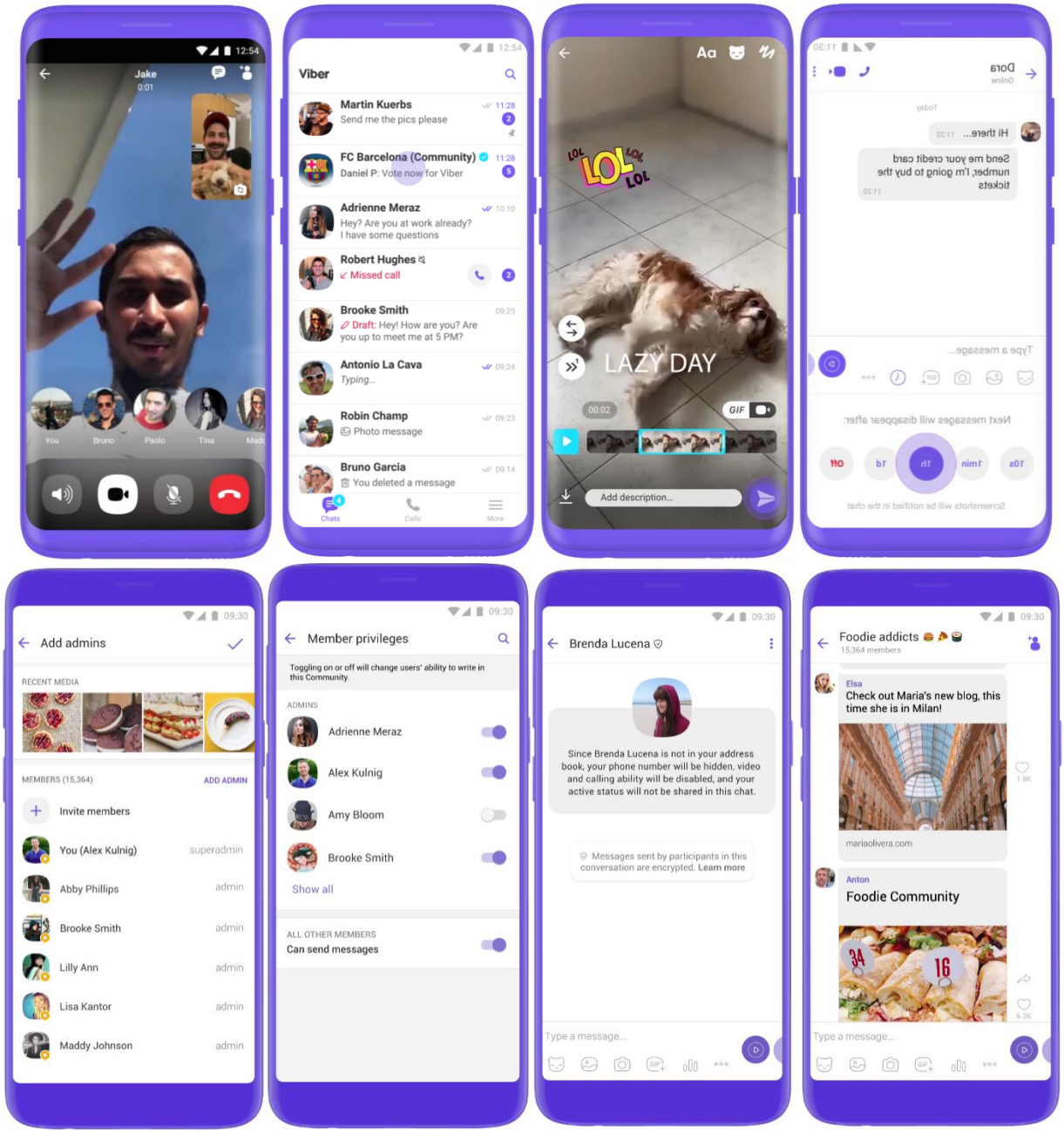


Рис. 1.16. Приклад мобільного інтерфейсу Viber

У таблиці нижче наведено ключові переваги мессенджера Viber, які роблять його привабливим для користувачів (табл. 1.9):

Таблиця 1.9

Переваги Viber

№	Перевага
1	Десктопна версія для Windows, macOS і Linux.
2	Групові чати та спільноти з розширеним управлінням.
3	Функція Viber Out для міжнародних дзвінків.
4	Індивідуальне оформлення чатів.
5	Можливість надсилання мультимедіа з підписом або малюнком.
6	Автоматичне зникнення повідомлень у секретних чатах.

Наступна таблиця містить основні недоліки Viber, які варто враховувати при виборі мессенджера для захищеного спілкування (табл. 1.10):

Таблиця 1.10

Недоліки Viber

№	Недолік
1	Наявність реклами та спаму.
2	Ліміт — до 250 учасників у груповому чаті.
3	Короткі голосові повідомлення (до 1 хвилини).
4	Обмеження на файли — до 200 МБ.
5	Відсутність Web-версії.
7	Збір метаданих і передача третім сторонам.
8	Обмежене застосування наскрізного шифрування.
9	Низький рейтинг безпеки (2/7 за оцінкою EFF).

Останні оновлення Viber додали функції зникання повідомлень, секретні чати, приховування номера, статусу онлайн, індикатора введення тексту та звітів про прочитання. Також з’явилась можливість блокування екрана. Для захисту повідомлень застосовується протокол на основі Double Ratchet, подібний до Signal [17, 18]. Водночас реклама та сторонні трекери залишаються в платформі [11].

1.2.4 Signal

Signal — це безкоштовний мессенджер із відкритим кодом (GPLv3/AGPLv3), розроблений американською некомерційною організацією Signal Foundation (раніше Open Whisper Systems). Його засновниками є криптограф Моксі Марлінспайк і Браян Ектон (співзасновник WhatsApp). Основою мессенджера є Signal Protocol, побудований на Axolotl Ratchet, який поєднує найкращі рішення OTR та OMEMO і став стандартом для безпечного спілкування. Signal також підтримує Perfect Forward Secrecy (PFS), що підвищує рівень конфіденційності та захисту повідомлень [13, 14].

Для реєстрації в Signal використовується номер телефону, який слугує ідентифікатором. Підтвердження відбувається через SMS або голосовий дзвінок, а віртуальні чи невалідні номери не приймаються. Для захисту від спаму застосовується reCaptcha, а кількість спроб реєстрації обмежена [13].

При першому запуску додаток запитує доступ до контактів. Якщо погодитися, вони можуть синхронізуватися між усіма прив’язаними пристроями (мобільними та ПК). Мессенджер має потужний функціонал (рис. 1.17).

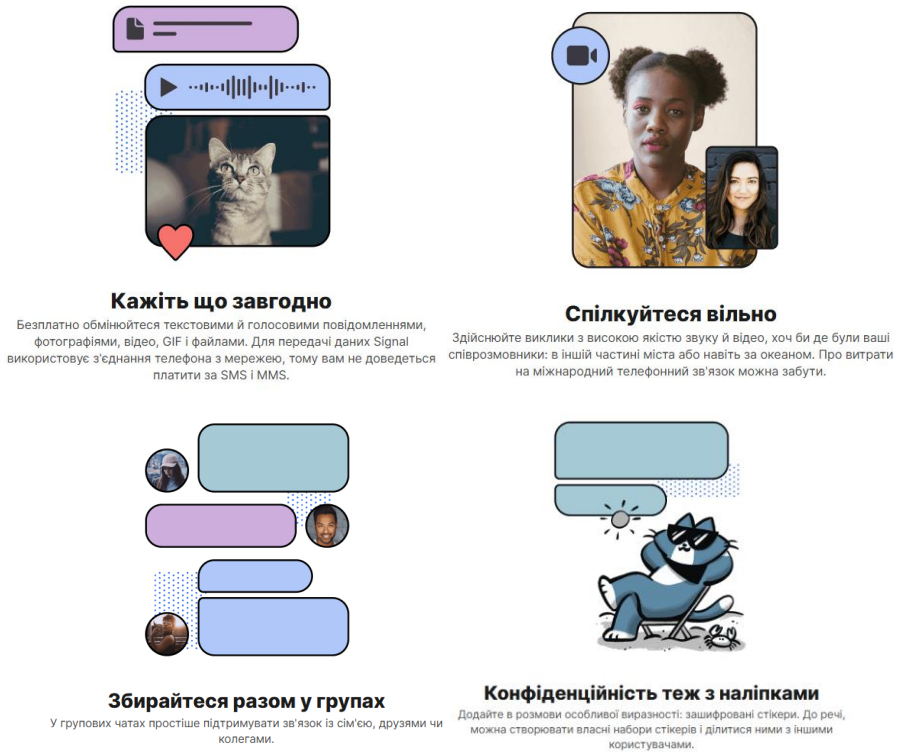


Рис. 1.17. Перелік базового функціоналу Signal

Signal має простий і функціональний інтерфейс, що дозволяє користувачам створювати зникаючі повідомлення, здійснювати аудіо та відеодзвінки, а також публікувати власні історії. Крім того, можна ділитися різними файлами, включаючи аудіо, відео, GIF-файли, емоджі та інші типи контенту. Користувачі можуть створювати групи для спілкування та обміну мультимедійними файлами (рис. 1.18) [13].

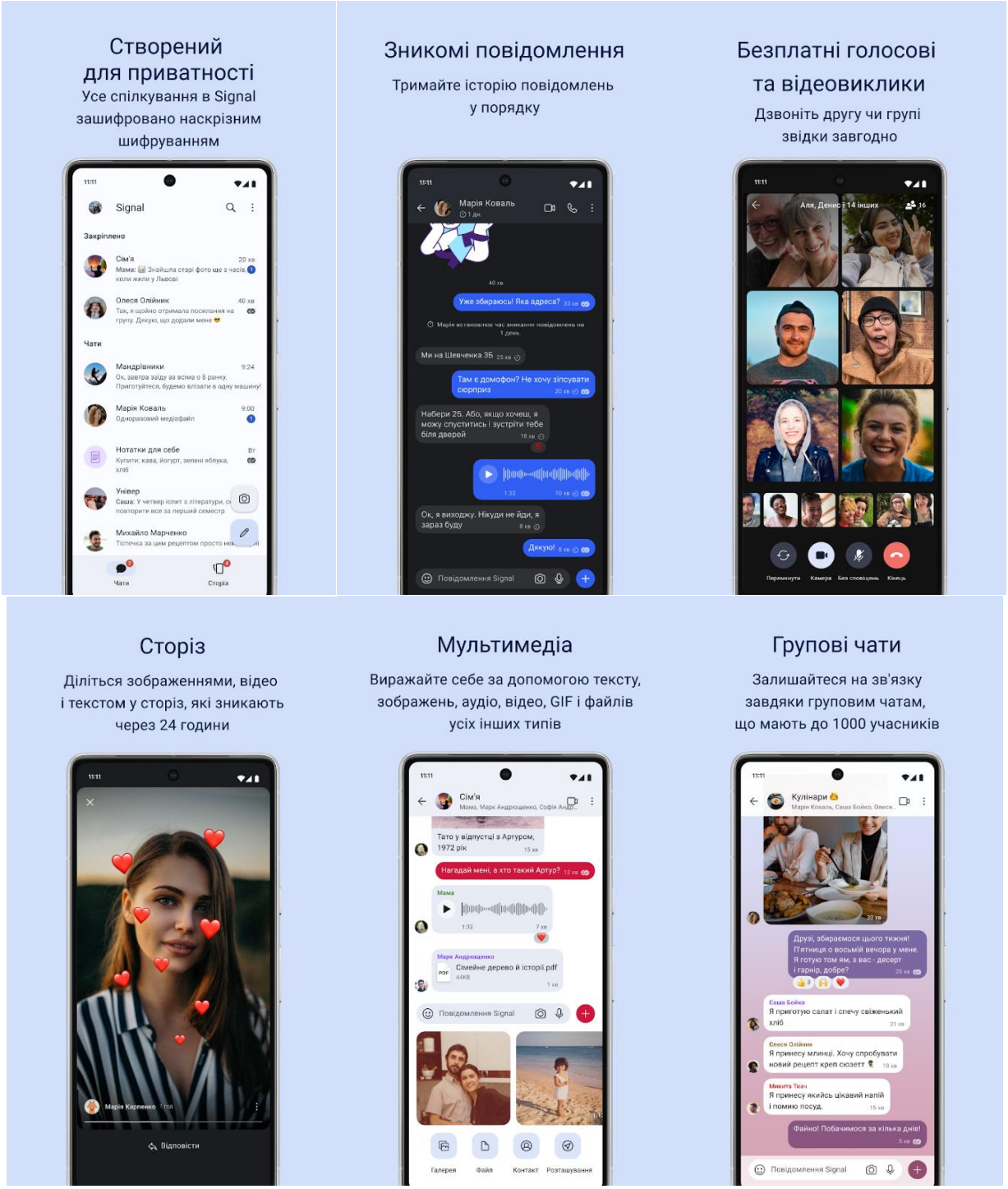


Рис. 1.18. Приклад мобільного інтерфейсу та функціоналу Signal

У таблиці нижче подано ключові переваги Signal, які роблять його одним із найнадійніших засобів захищеного спілкування (табл. 1.11):

Таблиця 1.11

Переваги Signal

№	Перевага
1	Високий рівень шифрування даних.
2	Працює стабільно навіть при повільному інтернет-з'єднанні.
3	Повністю без реклами — чистий інтерфейс.
4	Не збирає персональні дані користувачів.
5	Підтримка до 5 пристроїв одночасно.
6	Високий рейтинг безпеки — рекомендовано Electronic Frontier Foundation.

Нижче наведено основні недоліки мессенджера Signal, які обмежують його функціональність або зручність використання (табл. 1.12):

Таблиця 1.12

Недоліки Signal

№	Недолік
1	Втрата даних при зміні номера — чати не зберігаються на серверах.
2	Обмеження на пересилання файлів — максимум 100 МБ.
3	Ліміт на учасників у групах — до 1000 осіб.
4	Обов'язкова прив'язка до номера телефону для реєстрації.
5	Контакти зберігаються на серверах
6	ПК-додаток працює лише з підключеним мобільним пристроєм.
7	Відсутня Web-версія

Signal поєднує простоту, безкоштовність, стабільність, конфіденційність та зручний інтерфейс, що робить його одним із найкращих мессенджерів для безпечного спілкування. Водночас обов'язкова прив'язка до номера телефону обмежує анонімність. У разі впровадження альтернативної реєстрації, Signal має потенціал стати лідером ринку. Наразі це одна з найкращих альтернатив WhatsApp для особистого та професійного спілкування [14].

1.2.5 Session

Session — безкоштовний, кросплатформний мессенджер з відкритим кодом, створений для максимальної конфіденційності. Він використовує наскрізне шифрування (E2EE) за замовчуванням і працює на основі децентралізованої мережі Lokinet з маршрутизацією ONION. Розробником є Oxen Privacy Tech Foundation, а в основі лежить модифікований Signal Protocol [13, 15].

Реєстрація не потребує номера телефону чи email — достатньо вказати ім'я. Підтримуються Linux, macOS, Windows, Android та iOS, а завантажити додаток можна через Google Play, F-Droid або App Store. Мессенджер має низку функціональних можливостей (рис. 1.19) [15].

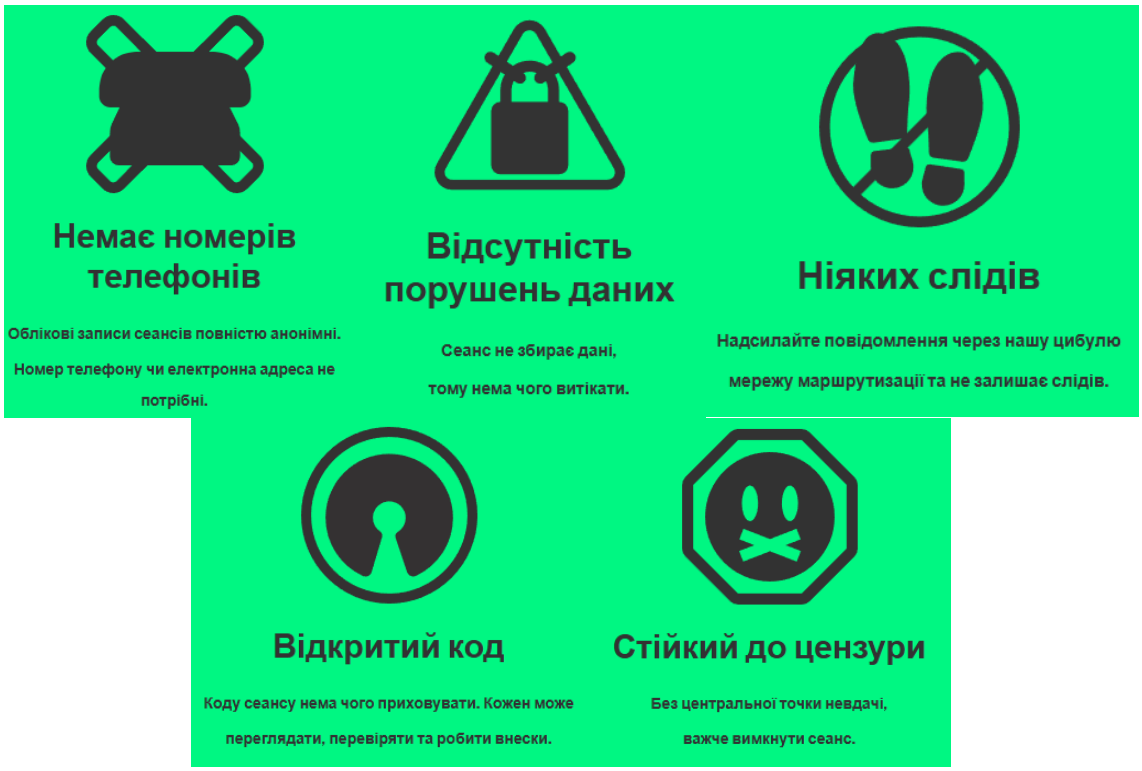


Рис. 1.19. Перелік функціональних переваг Session

Для початку спілкування в Session користувачі обмінюються Session ID або сканують QR-коди. Повідомлення проходять через "цибулеву" маршрутизацію, а ключі шифрування зберігаються лише на пристрої, що гарантує анонімність [15].

Мессенджер має зручний інтерфейс із вбудованими функціями приватності (рис. 1.20) [15].

Рис. 1.20. Приклад інтерфейсу та функціоналу Session

Нижче в таблиці наведено основні переваги Session (табл. 1.13):

Таблиця 1.13

Переваги Session

№	Перевага
1	Автознищення повідомлень (мінімальний час – 5 секунд).
2	Блокування скріншотів, індикаторів набору, перегляду URL і звітів про прочитання.
3	Підтримка реакцій, емодзі та гіфок.
4	Аудіо- та відеодзвінки (бета-версія, можлива передача IP-адреси).
5	Доступ через PIN-код.
6	Автоматичне видалення повідомлень у публічних чатах через 6 місяців.
7	Не потребує номера телефону чи email для реєстрації — лише Session ID.

Наступна таблиця демонструє обмеження та слабкі сторони Session, які можуть впливати на досвід користування (табл. 1.14):

Таблиця 1.14

Недоліки Session

№	Недолік
1	Обмеження на розмір файлів — до 10 МБ.
2	Групові чати та публічні спільноти не шифруються.
3	Відсутність групових викликів.
4	Можливість блокування користувачів адміністраторами (модерація).
5	Зберігання файлів на власних серверах
6	Проблеми синхронізації між мобільною та десктопною версіями.
7	Відсутність Web-версії — неможливість користування через браузер.

Session — це модифікована версія Signal з посиленим акцентом на конфіденційність, але поки що менш стабільна та зручна. Попри децентралізовану ONION-архітектуру, мессенджер використовує сервери OXEN, а деякі функції залишаються у бета-версії, що спричиняє помилки. Наразі не рекомендується для передачі конфіденційних файлів [15].

1.3 Порівняльний аналіз мессенджерів

За результатами дослідження кожного з мессенджерів проведемо порівняльну характеристику між ними (табл. 1.15):

Таблиця 1.15

Порівняльна характеристика мессенджерів

Параметр	Telegram	Viber	WhatsApp	Signal	Session
Шифрування	E2EE лише в секретних чатах	E2EE в особистих чатах	E2EE за замовчуванням	E2EE за замовчуванням	E2EE + децентралізація
Анонімність	Прив’язка до номера	Прив’язка до номера	Прив’язка до номера	Прив’язка до номера	Без номера, Session ID

Продовження таблиці 1.15

Доступність	Android, iOS, ПК, Web	Android, iOS, ПК	Android, iOS, ПК, Web	Android, iOS, ПК	Android, iOS, ПК
Груповий чат	До 200 000	До 250	До 256	До 100	До 100
Макс. розмір файлу	2 ГБ	200 МБ	100 МБ	100 МБ	10 МБ
Наявність Web-версії	Так	Ні	Так	Ні	Ні

Кожен із мессенджерів має свої переваги та недоліки. Telegram і WhatsApp пропонують широкий функціонал, але поступаються Signal і Session у рівні конфіденційності. Session виділяється децентралізацією, хоча частина функцій ще тестується. Viber зручний у використанні, але менш захищений. Остаточний вибір залежить від особистих пріоритетів користувача.

1.4 Висновки до розділу 1

У цьому розділі було розглянуто найпопулярніші мессенджери: Telegram, Viber, WhatsApp, Signal і Session. Кожен має свої сильні та слабкі сторони. Telegram і WhatsApp пропонують широкий функціонал — текстові й мультимедійні повідомлення, дзвінки, канали, боти. Однак рівень конфіденційності у них нижчий, ніж у Signal та Session, які орієнтовані на анонімність — зокрема завдяки децентралізованій мережі Session [14, 15, 18].

Водночас Session не має Web-версії й обмежений у мобільній підтримці, а Signal також не підтримує роботу через браузер [13, 15]. Viber зручний для дзвінків і спільнот, але поступається в питаннях захисту даних [11, 17].

Жоден із мессенджерів не поєднує одночасно розвинену функціональність, високий рівень конфіденційності та повноцінну Web-версію. Тому вибір залежить від пріоритетів користувача — між зручністю та безпекою.

2. ПРОЄКТУВАННЯ ТА АРХІТЕКТУРНІ РІШЕННЯ ЗАХИЩЕНОГО WEB-МЕССЕНДЖЕРА

2.1 Методологія та підхід до проєктування

Розробка захищеного Web-мессенджера вимагає застосування сучасних методів і технологій для забезпечення високого рівня безпеки, продуктивності та зручності використання. У цьому розділі розглянуто методологічний підхід до проєктування, який поєднує об’єктно-орієнтовану архітектуру з гнучкими методами розробки.

Зважаючи на динамічний розвиток Web-технологій та зростання вимог до захищеності даних, було прийнято рішення застосувати об’єктно-орієнтований підхід (ООП), що забезпечує модульність, масштабованість і повторне використання коду. Основною методологією обрано Agile, яка дозволяє адаптувати процес розробки до змінних вимог і швидко реагувати на виявлені недоліки. Поєднання ООП та Agile дозволяє створити систему, яка не лише задовольняє функціональні вимоги, а й забезпечує високий рівень безпеки та зручності користування.

Обґрунтування вибору підходу: об’єктно-орієнтоване програмування (ООП)

Об’єктно-орієнтований підхід (ООП) є ефективним методом проєктування програмних систем, що дозволяє структурувати код у вигляді класів та об'єктів. Це забезпечує підтримуваність та масштабованість системи, що важливо для розробки безпечного Web-мессенджера. Основними принципами ООП є інкапсуляція, наслідування, поліморфізм та абстракція.

Принципи ООП використовуються у Web-мессенджері для опису основних об'єктів системи. Це дозволяє легко розширювати функціонал та підтримувати код, оскільки кожен компонент ізольований і взаємодіє з іншими через визначені методи (табл. 2.1).

Таблиця 2.1

Основні принципи об’єктно-орієнтованого програмування (ООП)

Принцип ООП	Застосування в проєкті Web-мессенджера
Інкапсуляція	Дані кожного об’єкта (користувач, повідомлення, чат) захищені від прямого доступу ззовні, що підвищує безпеку системи.
Наслідування	Можливість створювати спеціалізовані інтерфейси для різних типів повідомлень або користувачів, що спрощує розширення системи.
Поліморфізм	Повідомлення можуть бути текстовими, зображеннями, файлами або відео, але обробляються через єдиний інтерфейс.
Абстракція	Всі сутності системи (користувач, чат, повідомлення) описані через інтерфейси з чітко визначеними властивостями та методами.

Для реалізації цього підходу було використано не класичні класи, а інтерфейси, що визначають структуру основних об'єктів системи. Такий підхід забезпечує легкість підтримки та модифікації, адже нові властивості чи методи можна додавати без змін основної архітектури (табл. 2.2). Це відповідає можливостям мови TypeScript, яка дозволяє використовувати ООП-принципи без створення повноцінних класів.

Таблиця 2.2

Основні інтерфейси проєкту

Інтерфейс	Призначення	Основні властивості
Message	Описує структуру повідомлення	текст, автор, час створення, ідентифікатор чату
Room	Визначає структуру кімнати для спілкування	назва, ідентифікатор творця, дата створення
User	Описує властивості користувача	унікальний ідентифікатор, ім'я, дата створення

Попри те, що структура об’єктів визначена через інтерфейси, система має реалізовані методи, які забезпечують повноцінну роботу цих сутностей у відповідних модулях. Основні методи системи описані у таблиці 2.3.

Таблиця 2.3

Основні методи об’єктів системи

Об'єкт	Методи
User	автентифікація
Message	шифрування, відправлення, форматування
Room	створення групових чатів, управління учасниками

Таке рішення забезпечує модульність та масштабованість системи, дозволяючи додавати нові властивості чи типи об’єктів без змін архітектури. Інтерфейси спрощують підтримку, адже нові методи можна додавати без впливу на існуючі компоненти [19].

Методологія розробки: Agile

Для розробки Web-мессенджера обрано методологію Agile, яка забезпечує гнучкий підхід до управління проєктом. Agile дозволяє поступово впроваджувати основні компоненти системи та адаптуватися до змінних вимог. Основні характеристики Agile у проєкті відображено у таблиці 2.4.

Таблиця 2.4

Основні характеристики Agile у проєкті

Характеристика	Застосування у проєкті Web-мессенджера
Гнучкість	Зміни можуть бути внесені на будь-якому етапі розробки, що важливо для підвищення безпеки.
Покрокова розробка	Реалізація основних функцій (авторизація, обмін повідомленнями) на початковому етапі.
Комунікація	Постійний зворотний зв'язок між розробниками та тестувальниками, що дозволяє оперативно реагувати.
Ітераційність	Функції додаються поступово, тестуються та оптимізуються, що дозволяє швидко виявляти помилки.

Agile дозволяє поступово вдосконалювати систему без значних витрат часу на перепроєктування. Кожна нова функція може бути протестована та оптимізована без порушення основної архітектури [20].

2.2 Формалізація задачі та постановка проблеми

Для успішної розробки захищеного Web-мессенджера необхідно чітко визначити основні вимоги та задачі, які система повинна виконувати. Основна мета цього проєкту — створити безпечний, масштабований та зручний Web-мессенджер, який забезпечує конфіденційність користувачів і захищений обмін повідомленнями. Важливим аспектом є забезпечення приватності користувачів без необхідності введення особистих даних, що робить систему більш безпечною та простою у використанні.

В умовах сучасного цифрового середовища конфіденційність комунікації та захист даних користувачів є пріоритетними завданнями. Багато популярних мессенджерів зберігають особисті дані користувачів, що може призводити до ризику їх компрометації. У розроблюваному мессенджері передбачається відмова від традиційної автентифікації (логін та пароль) на користь простого створення облікового запису за іменем користувача.

Формулювання задачі

Для досягнення мети проєкту визначено такі основні задачі (табл. 2.5):

Таблиця 2.5

Основні задачі проєкту

№	Назва задачі	Опис
1	Забезпечення безпечного обміну повідомленнями	Забезпечення обміну повідомленнями між користувачами за допомогою сучасних протоколів шифрування, що гарантують конфіденційність листування.
2	Проста авторизація	Користувач вводить бажане ім'я, після чого система створює унікальний профіль без необхідності введення пароля чи інших особистих даних, що нагадує підхід, який використовується у мессенджері Session [15].

Продовження таблиці 2.5

3	Обмін текстовими повідомленнями в реальному часі	Реалізація обміну текстовими повідомленнями без підтримки мультимедійного контенту (фото, відео, файли).
4	Підтримка групових чатів	Забезпечення можливості користувачам приєднуватися до групових чатів за допомогою унікальних ідентифікаторів, що дозволяє створювати групи без потреби у списку контактів.

Постановка проблеми

Розробка захищеного Web-мессенджера супроводжується низкою проблем, які необхідно вирішити (табл. 2.6):

Таблиця 2.6

Основні проблеми проєкту

№	Назва проблеми	Опис
1	Захист даних користувачів	Забезпечення надійного шифрування повідомлень та зберігання даних без можливості їх перехоплення третіми особами.
2	Конфіденційність комунікації	Забезпечення користувачам можливості спілкуватися без ризику доступу до їхніх облікових даних.
3	Продуктивність системи	Система повинна підтримувати миттєвий обмін повідомленнями навіть при великій кількості одночасних користувачів.
4	Масштабованість (необов'язкова)	Web-мессенджер повинен мати потенціал до розширення, підтримуючи більшу кількість користувачів у майбутньому.
5	Зручність використання	Створення інтуїтивно зрозумілого інтерфейсу, що забезпечує легкий доступ до основних функцій.

Моделювання потоків даних

Для коректної роботи Web-мессенджера важливо визначити основні потоки даних, які проходять через систему (табл. 2.7). Це дозволяє формалізувати процес обміну повідомленнями та забезпечити його безпеку.

Таблиця 2.7

Основні потоки даних у Web-мессенджері

Потік даних	Опис
Вхідні повідомлення	Текстові дані, які надходять від користувачів до сервера.
Вихідні повідомлення	Повідомлення, що пересилаються сервером іншим користувачам у режимі реального часу.
Дані користувача	Інформація про обліковий запис користувача (ім'я, унікальний ідентифікатор).
Дані про з’єднання	Відомості про IP-адресу, час з’єднання та використані протоколи захисту.
Дані шифрування	Ключі шифрування, що забезпечують безпечний обмін повідомленнями.

Ці потоки даних забезпечують базову функціональність системи. Вхідні повідомлення проходять через серверну частину, де вони можуть бути оброблені та зашифровані, після чого передаються іншим користувачам у захищеному вигляді. Інформація про користувача та його підключення дозволяє системі визначати учасників спілкування та забезпечувати їхню взаємодію.

Таким чином, визначення потоків даних є важливим етапом формалізації задачі. Це дозволяє не лише зрозуміти основні процеси системи, а й гарантувати їхню безпечну роботу.

2.3 Архітектура системи

Для створення захищеного Web-мессенджера обрано клієнт-серверну архітектуру, яка забезпечує чіткий розподіл функціональності між клієнтською та серверною частинами. Основна концепція полягає в тому, що клієнтська частина відповідає за взаємодію з користувачем, шифрування повідомлень перед їх відправленням, а також їх розшифрування після отримання. Серверна частина приймає зашифровані повідомлення, зберігає їх у зашифрованому вигляді у базі даних та передає їх іншим користувачам.

Клієнт-серверна архітектура є оптимальним вибором для реалізації Web-мессенджера через низку ключових переваг (табл. 2.8):

Таблиця 2.8

Основні переваги клієнт-серверної архітектури Web-мессенджера

Перевага	Опис
Централізоване управління	Всі дані користувачів і повідомлення зберігаються на сервері, що дозволяє контролювати їхню безпеку.
Захист даних	Шифрування повідомлень відбувається на стороні клієнта, а сервер зберігає їх у зашифрованому вигляді, не маючи доступу до змісту повідомлень.
Масштабованість	Система може обробляти великий обсяг одночасних підключень користувачів без значної втрати продуктивності.
Простота підтримки	Оновлення або вдосконалення серверної частини не потребують змін на стороні клієнта.
Розширюваність	До сервера можна легко додавати нові функції або компоненти без необхідності змінювати клієнтську частину.

Такий підхід дозволяє досягти високого рівня конфіденційності та забезпечити надійний захист даних користувачів, оскільки сервер не має доступу до відкритого тексту повідомлень. Сервер виконує лише функцію передачі та зберігання зашифрованих даних, що мінімізує ризик витоку інформації.

Основні компоненти архітектури

Основні компоненти архітектури розподілені таким чином, щоб кожен із них виконував свою конкретну функцію. Це дозволяє легко масштабувати систему, змінювати її функціональність або впроваджувати нові методи захисту. Далі наведено таблицю з описом ключових компонентів системи та їхніми завданнями (табл. 2.9).

Таблиця 2.9

Основні компоненти архітектури Web-мессенджера

Компонент	Призначення
Клієнтська частина	Забезпечує інтерфейс користувача, надсилання та отримання повідомлень.
Сервер	Приймає запити від клієнтів, обробляє їх, створює та зберігає профілі користувачів.
WebSocket-сервер	Забезпечує миттєвий обмін повідомленнями в режимі реального часу.
База даних	Зберігає інформацію про користувачів, групові чати та текстові повідомлення.
Система шифрування	Забезпечує захист повідомлень за допомогою протоколу наскрізного шифрування (E2EE).

Діаграма архітектури системи

Для забезпечення чіткого розуміння взаємодії компонентів системи розроблено архітектурну діаграму (рис. 2.7), яка демонструє основні елементи та їхній взаємозв'язок. Архітектура системи включає клієнтську частину (інтерфейс користувача), сервер для обробки запитів та зберігання даних, WebSocket-сервер для миттєвого обміну повідомленнями та базу даних для зберігання інформації.

Діаграма відображає, як користувачі підключаються до сервера через WebSocket-з’єднання, яке забезпечує постійний канал зв’язку між клієнтом і сервером. Сервер приймає зашифровані текстові повідомлення та зберігає у базу даних. При цьому повідомлення залишаються у зашифрованому вигляді, що гарантує їхню конфіденційність навіть у разі компрометації бази даних.

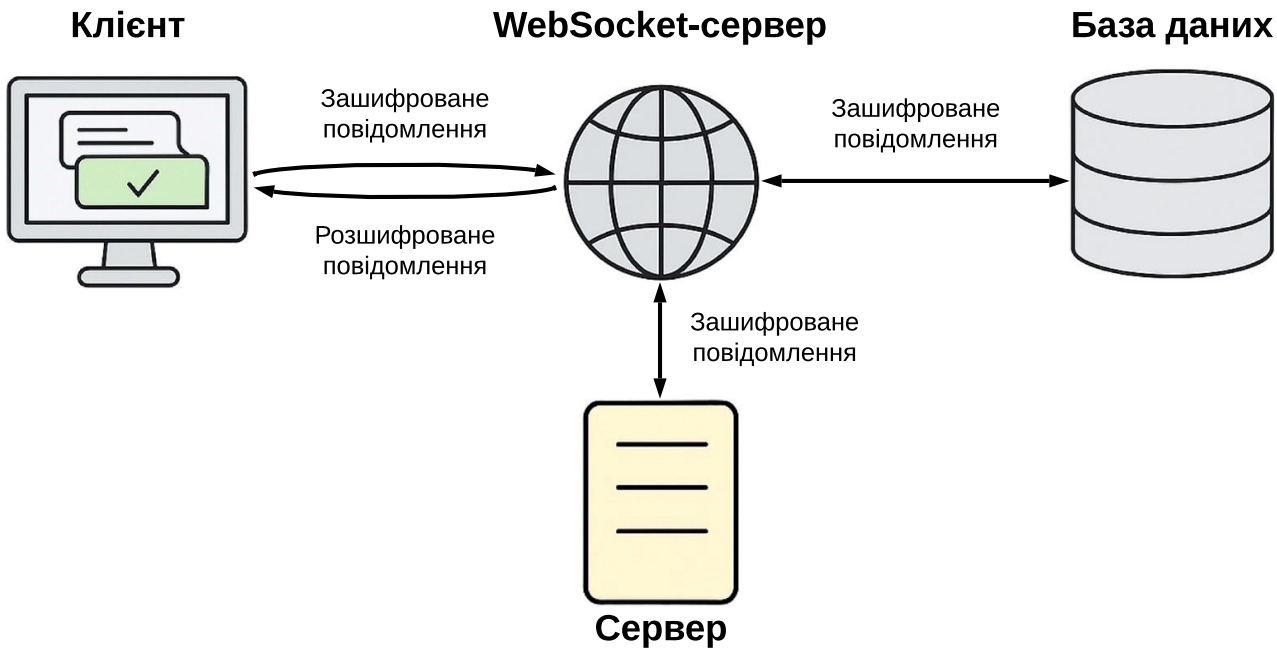


Рис. 2.1. Архітектурна схема Web-мессенджера

На діаграмі відображено основні архітектурні компоненти Web-мессенджера та їх зв’язок між собою. Розглянемо роль та призначення кожного (табл. 2.10):

Таблиця 2.10

Взаємодія компонентів Web-мессенджера

Компонент	Призначення
Клієнт	Web-додаток, який працює у браузері користувача. Дозволяє вводити текстові повідомлення, приєднуватися до групових чатів і отримувати повідомлення. Клієнт забезпечує зручний інтерфейс користувача та шифрує повідомлення перед відправленням.
WebSocket-сервер	Забезпечує постійне з'єднання між клієнтами та сервером, дозволяючи миттєво надсилати та отримувати повідомлення. Використовує зашифроване з'єднання (TLS), що гарантує безпечну передачу даних навіть через відкриті мережі.
Сервер	Виконує функції обробки повідомлень, управління профілями користувачів та шифрування даних. Сервер приймає зашифровані повідомлення від клієнта через WebSocket, розшифровує їх, зберігає у базі даних і пересилає їх іншим користувачам.
База даних	Зберігає інформацію про користувачів, групові чати та повідомлення. Повідомлення зберігаються у зашифрованому вигляді для забезпечення безпеки та конфіденційності.

Основні потоки даних на діаграмі

Діаграма на рис. 2.2 демонструє процес обміну повідомленнями між користувачами у Web-мессенджері. Повідомлення, створене користувачем, шифрується на стороні клієнта та надсилається через WebSocket-сервер на сервер. Сервер зберігає зашифроване повідомлення у базі даних, а при надсиланні іншим користувачам повідомлення розшифровується та доставляється їм у читабельному вигляді.

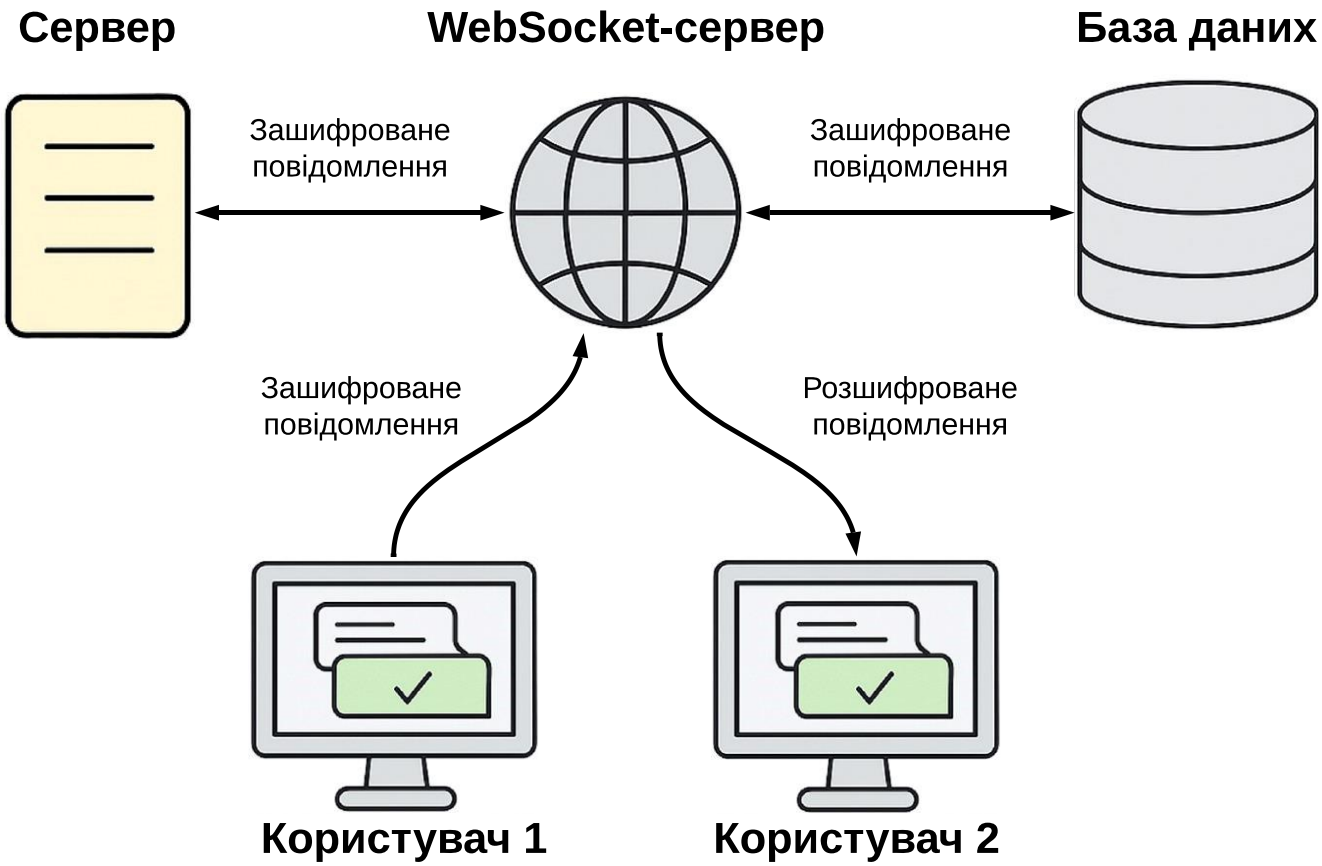


Рис. 2.2. Потік даних при обміні повідомленнями

Основні етапи обміну повідомленнями:

- Користувач створює та шифрує повідомлення.
- Повідомлення надсилається через WebSocket-сервер.
- Сервер зберігає зашифроване повідомлення у базі даних.
- Інші користувачі отримують розшифроване повідомлення.

Діаграма на рис. 2.3 відображає процес створення нового користувача та управління його даними. Користувач вводить нікнейм, сервер генерує унікальний ідентифікатор та зберігає інформацію про користувача у базі даних. Через WebSocket користувач отримує доступ до групових чатів та може приєднуватися до них.

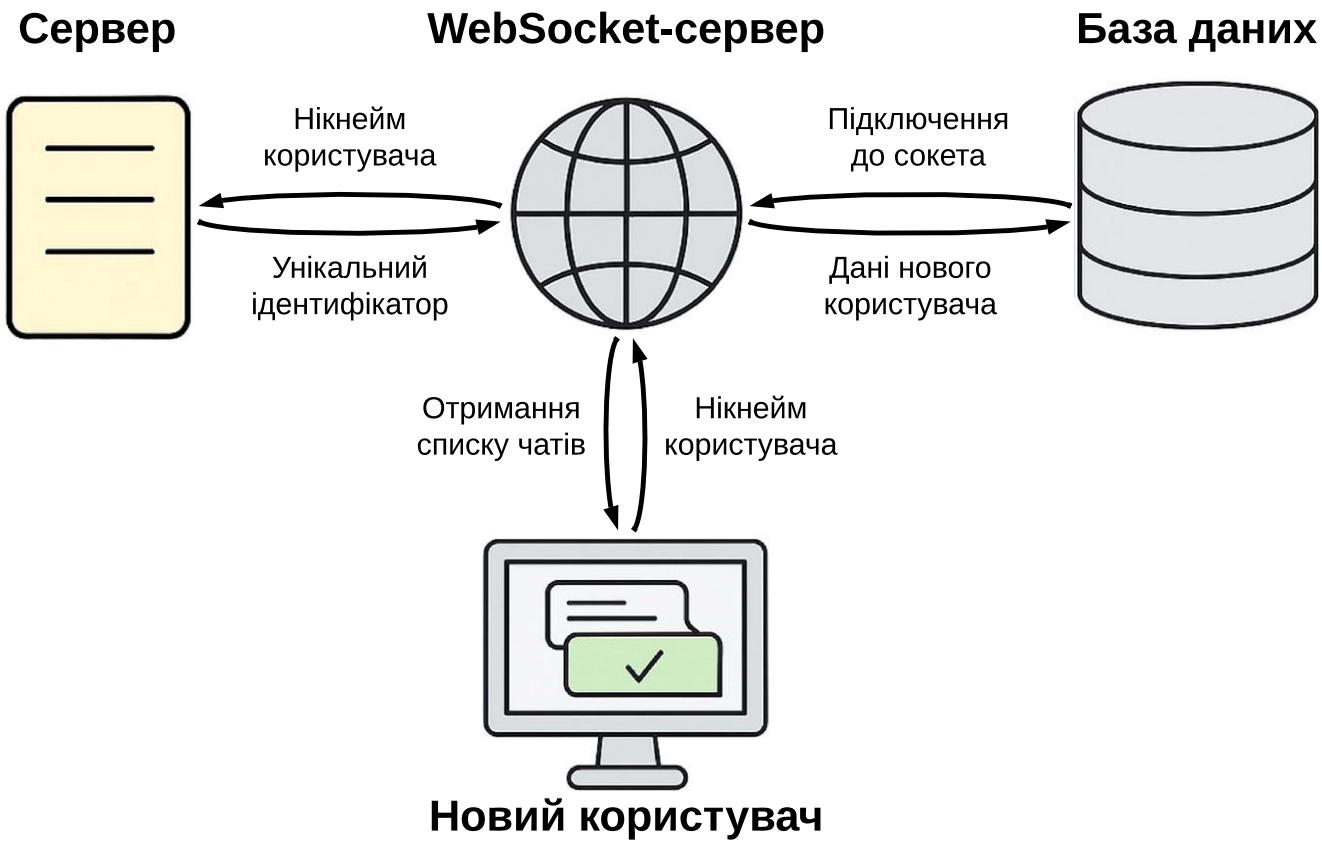


Рис. 2.3. Потік даних при авторизації нового користувача

Основні етапи управління даними користувачів:

- Створення нового користувача з унікальним ідентифікатором.
- Збереження даних користувача у базі даних.
- Підключення користувача до WebSocket для обміну повідомленнями.
- Управління груповими чатами та підключення до них.

Цей підхід забезпечує швидку реєстрацію, безпеку даних та миттєвий обмін повідомленнями між користувачами.

Принципи безпеки архітектури

Для забезпечення конфіденційності та безпеки обміну повідомленнями в архітектурі Web-мессенджера реалізовано наступні принципи (табл. 2.11):

Таблиця 2.11

Принципи безпеки архітектури Web-мессенджера

Принцип	Опис
Наскрізне шифрування (E2EE)	Повідомлення шифруються на стороні відправника та розшифровуються лише на стороні отримувача.
Захищене з'єднання (HTTPS, TLS)	Всі дані передаються через зашифроване з'єднання, що захищає їх від перехоплення.
Анонімність користувачів	Користувачі не вводять особисті дані при реєстрації, лише ім'я користувача.
Захист даних у базі даних	Повідомлення зберігаються у зашифрованому вигляді, що унеможливлює їх витік.
Контроль доступу	Лише користувачі групового чату мають доступ до повідомлень, що в ньому надсилаються.

Ця архітектура дозволяє створити систему, яка гарантує безпеку користувачів, зручність використання та стабільну роботу навіть при високому навантаженні. Розділення функцій між компонентами забезпечує надійність та масштабованість системи: серверна частина відповідає за обробку запитів та зберігання даних, а клієнтська частина — за взаємодію з користувачем. Використання WebSocket дозволяє забезпечити миттєвий обмін повідомленнями в реальному часі, що є ключовою вимогою для мессенджера.

2.4 Інформаційна модель системи

У процесі розробки захищеного Web-мессенджера важливим аспектом є правильне проєктування моделей даних, які забезпечують ефективне зберігання та обробку інформації. Моделі даних визначають структуру та взаємозв'язок основних сутностей системи, таких як користувачі, чати та повідомлення.

Правильне проєктування моделей даних гарантує стабільність та продуктивність системи, а також забезпечує її масштабованість. Відповідно, моделі даних мають бути розроблені таким чином, щоб мінімізувати ризики втрати даних, забезпечити їхню конфіденційність та спростити управління.

У межах цього підпункту буде розглянуто три основні моделі даних Web-мессенджера: концептуальну, логічну та фізичну. Це дозволить чітко визначити структуру бази даних, її компоненти та взаємозв'язки між ними. Окрему увагу буде приділено аналізу потоків даних у системі за допомогою DFD-діаграм, що забезпечує наочне розуміння функціонування системи.

Опис основних таблиць та їх атрибутів

Основні таблиці бази даних Web-мессенджера забезпечують зберігання інформації про користувачів, групові чати та текстові повідомлення. Кожна таблиця має свою чітко визначену функцію, що дозволяє ефективно організувати управління даними та забезпечити їхню безпеку (табл. 2.12).

Таблиця 2.12

Основні таблиці бази даних Web-мессенджера

Таблиця	Призначення	Основні атрибути
Користувачі	Зберігає інформацію про зареєстрованих користувачів.	Ідентифікатор (ID), нікнейм, дата створення.
Чати	Містить дані про групові чати та їх учасників.	Ідентифікатор (ID), назва чату, ID автора.
Повідомлення	Зберігає текстові повідомлення, надіслані користувачами.	Ідентифікатор (ID), текст повідомлення, час відправлення, ID чату.

Концептуальна модель даних

Концептуальна модель даних визначає основні сутності Web-мессенджера та їхні взаємозв'язки. Вона є абстрактним представленням структури даних системи, що дозволяє зрозуміти, як організовані дані та як вони взаємодіють між собою. Концептуальна модель фокусується на ключових об'єктах системи, не враховуючи технічні деталі їх фізичного зберігання.

Основні сутності системи та їхні атрибути наведено в таблицях 2.13 – 2.15.

Таблиця 2.13

Сутність "Користувач"

Атрибут	Опис
ID користувача	Унікальний ідентифікатор користувача (UUID).
Нікнейм	Ім'я користувача, яке він обирає під час реєстрації.
Дата створення	Час створення облікового запису користувача.
Дата редагування	Час останнього редагування профілю користувача.

Таблиця 2.14

Сутність "Чат (Кімната)"

Атрибут	Опис
ID кімнати	Унікальний ідентифікатор чату (UUID).
Назва кімнати	Назва групового чату.
ID власника	Ідентифікатор користувача, який створив чат.
Дата створення	Час створення чату.
Дата редагування	Час останнього редагування чату.

Таблиця 2.15

Сутність "Повідомлення"

Атрибут	Опис
ID повідомлення	Унікальний ідентифікатор повідомлення (UUID).
Нікнейм автора	Ім'я користувача, який надіслав повідомлення.
Текст повідомлення	Зміст текстового повідомлення.
Час відправки	Час, коли повідомлення було відправлене.
ID кімнати	Ідентифікатор чату, у якому надіслано повідомлення.
ID автора	Ідентифікатор користувача, який надіслав повідомлення.
Дата створення	Час створення повідомлення.
Дата редагування	Час останнього редагування повідомлення.

На рис. 2.7 представлено концептуальну модель даних Web-мессенджера, яка демонструє зв'язки між основними сутностями та їхніми атрибутами.

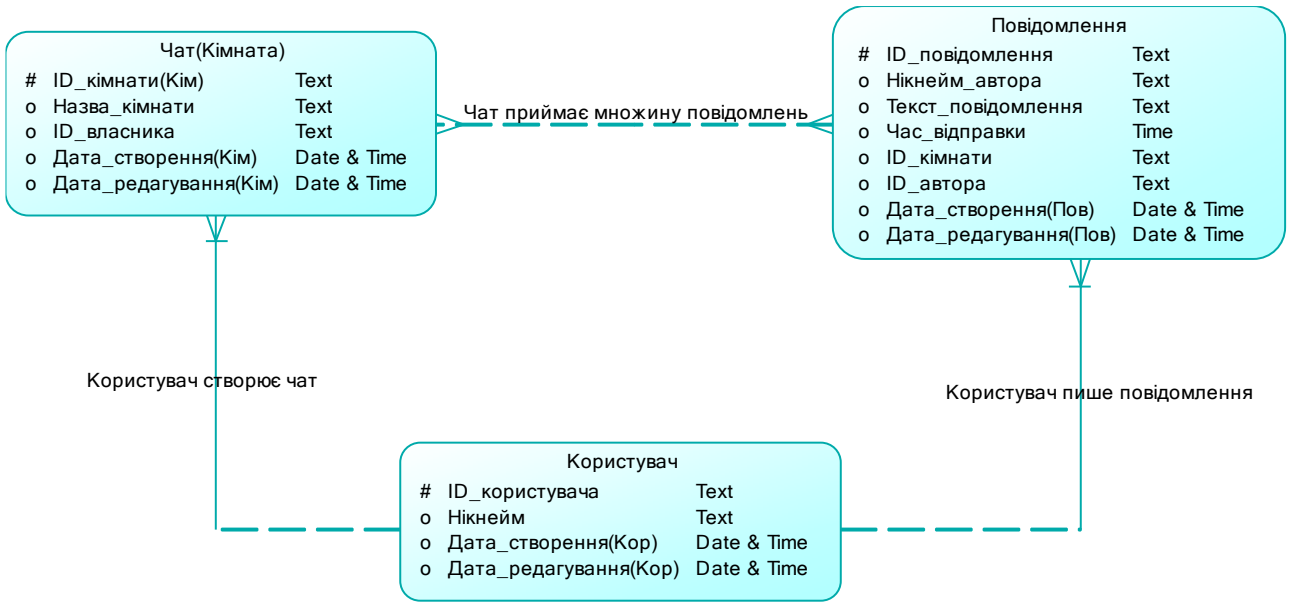


Рис. 2.7. Концептуальна модель даних Web-мессенджера

Взаємозв'язки між сутностями:

- Користувач може створити один або кілька чатів.
- Кожен чат має автора (власника), який створив його.
- Користувач може надсилати повідомлення у кількох чатах.
- Повідомлення належать до конкретного чату та мають автора.

Логічна модель даних

Логічна модель даних є деталізованим представленням концептуальної моделі, яке визначає структуру даних, їхні атрибути та взаємозв'язки між ними. Вона відображає, як дані організовані та взаємодіють між собою у межах системи, зокрема показує зв'язки між таблицями та їх ключами.

На рис. 2.8 зображено логічну модель даних Web-мессенджера, яка демонструє взаємозв'язки між основними сутностями системи:

- **Користувач:** може створювати чати та надсилати повідомлення.
- **Чат (Кімната):** містить повідомлення, які користувачі надсилають.
- **Повідомлення:** надсилаються користувачами та належать до конкретного чату.
- **Чат приймає повідомлення:** забезпечує зв'язок між користувачами, чатами та повідомленнями.

Сутність "Чат приймає повідомлення"

Ця сутність відображає зв'язок між користувачами, чатами та повідомленнями (табл. 2.16). Вона дозволяє організувати багато-в-багато зв'язок, що означає, що один користувач може надсилати або отримувати багато повідомлень у кількох чатах.

Таблиця 2.16

Сутність "Чат приймає повідомлення"

Атрибут	Опис
ID користувача	Ідентифікатор користувача, який є учасником чату.
ID кімнати	Ідентифікатор чату, до якого користувач має доступ.
ID повідомлення	Ідентифікатор повідомлення, яке було надіслано або отримано у чаті.

Рис. 2.8. Логічна модель даних Web-мессенджера

Дана модель дозволяє ефективно організувати структуру даних у системі, забезпечуючи їхню гнучкість та масштабованість. Користувачі можуть надсилати повідомлення у чати, створювати нові чати та приєднуватися до існуючих. Повідомлення відображаються лише тим користувачам, які є учасниками відповідного чату.

Фізична модель даних

Фізична модель даних є деталізованим представленням логічної моделі, яке показує, як дані будуть зберігатися у базі даних. Вона визначає типи даних для кожного атрибута, ключі (первинні та зовнішні), а також множинність зв'язків між сутностями.

На рис. 2.9 представлено фізичну модель даних Web-мессенджера. Вона показує структуру таблиць та зв'язки між ними, а також типи даних, які використовуються для зберігання інформації.

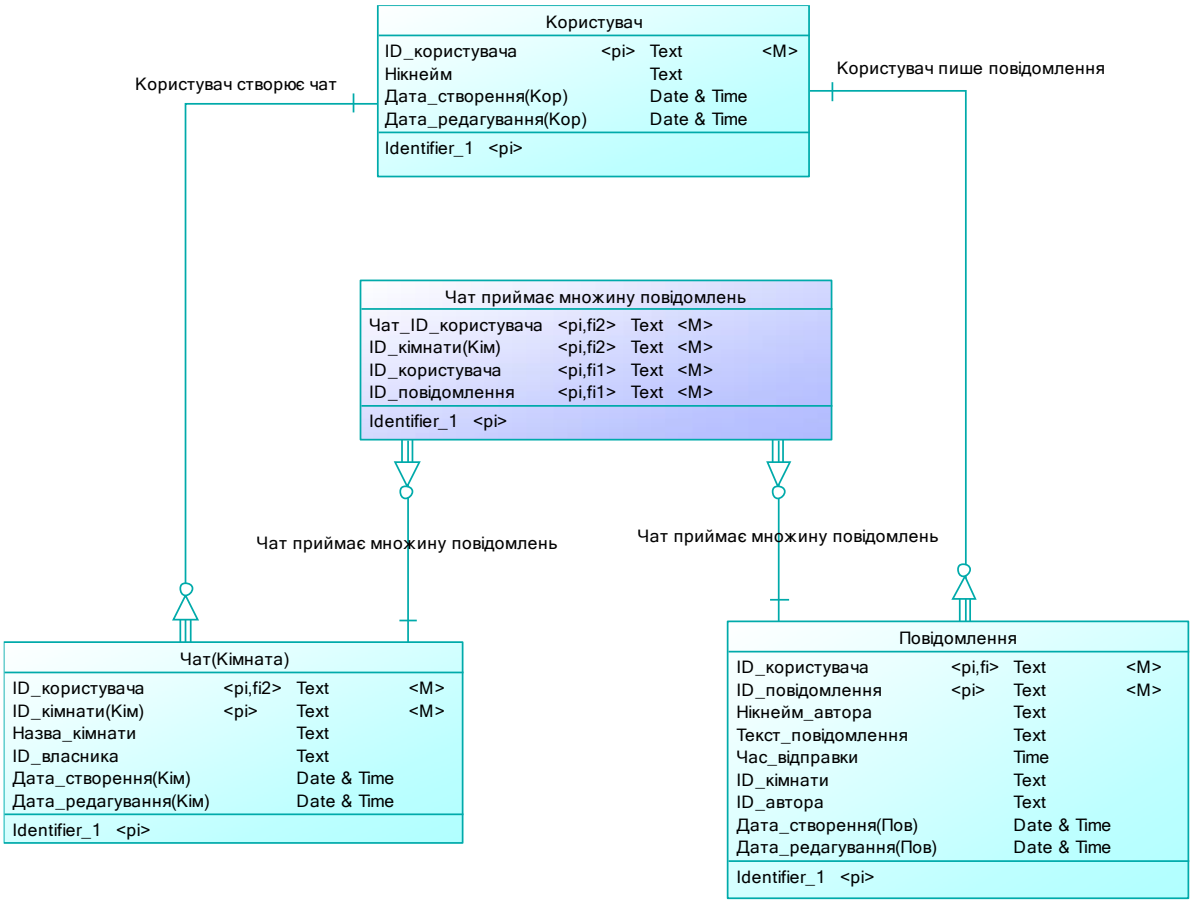


Рис. 2.9. Фізична модель даних Web-мессенджера

Дана модель відображає кардинальності між сутностями, що визначають, як дані пов'язані між собою (табл. 2.17):

Таблиця 2.17

Кардинальності та зв'язки

Зв'язок	Тип зв'язку	Опис
Користувач — Чат	Один-до-багатьох	Один користувач може створити кілька чатів.
Користувач — Повідомлення	Один-до-багатьох	Один користувач може надсилати багато повідомлень.
Чат — Повідомлення	Один-до-багатьох	Один чат може містити багато повідомлень.
Чат приймає повідомлення — Користувач	Багато-в-багато	Один користувач може брати участь у кількох чатах.
Чат приймає повідомлення — Повідомлення	Багато-в-багато	Одне повідомлення може бути надіслане в кілька чатів.

Фізична модель чітко визначає типи даних кожного атрибута, що забезпечує коректне зберігання інформації у базі даних (табл. 2.18):

Таблиця 2.18

Типи даних у фізичній моделі

Сутність	Атрибут	Тип даних
Користувачі	ID користувача	Text (UUID)
	Нікнейм	Text
	Дата створення	Date & Time
	Дата редагування	Date & Time
Чати	ID кімнати	Text (UUID)
	Назва кімнати	Text
	ID власника	Text (UUID)
	Дата створення	Date & Time
	Дата редагування	Date & Time
Повідомлення	ID повідомлення	Text (UUID)
	Нікнейм автора	Text
	Текст повідомлення	Text
	Час відправки	Time
	ID кімнати	Text (UUID)
	ID автора	Text (UUID)
	Дата створення	Date & Time
	Дата редагування	Date & Time
Чат приймає повідомлення	ID користувача	Text (UUID)
	ID кімнати	Text (UUID)
	ID повідомлення	Text (UUID)

DFD-діаграми системи

DFD (Data Flow Diagram) або діаграма потоків даних дозволяє візуалізувати, як інформація рухається у системі, які процеси відбуваються з даними та як вони взаємодіють між собою. У межах Web-мессенджера DFD-діаграми допомагають чітко зрозуміти процеси обробки повідомлень, управління користувачами та взаємодії між клієнтом і сервером.

DFD діаграма нульового рівня

На діаграмі нульового рівня (рис. 2.10) відображено загальну структуру системи Web-мессенджера та основні потоки даних між користувачем, сервером і базою даних:

- **Користувач:** надсилає дані авторизації або повідомлення до системи.
- **Сервер:** приймає вхідні запити, обробляє їх і передає захищені повідомлення.
- **База даних:** зберігає інформацію про користувачів, чати та повідомлення.

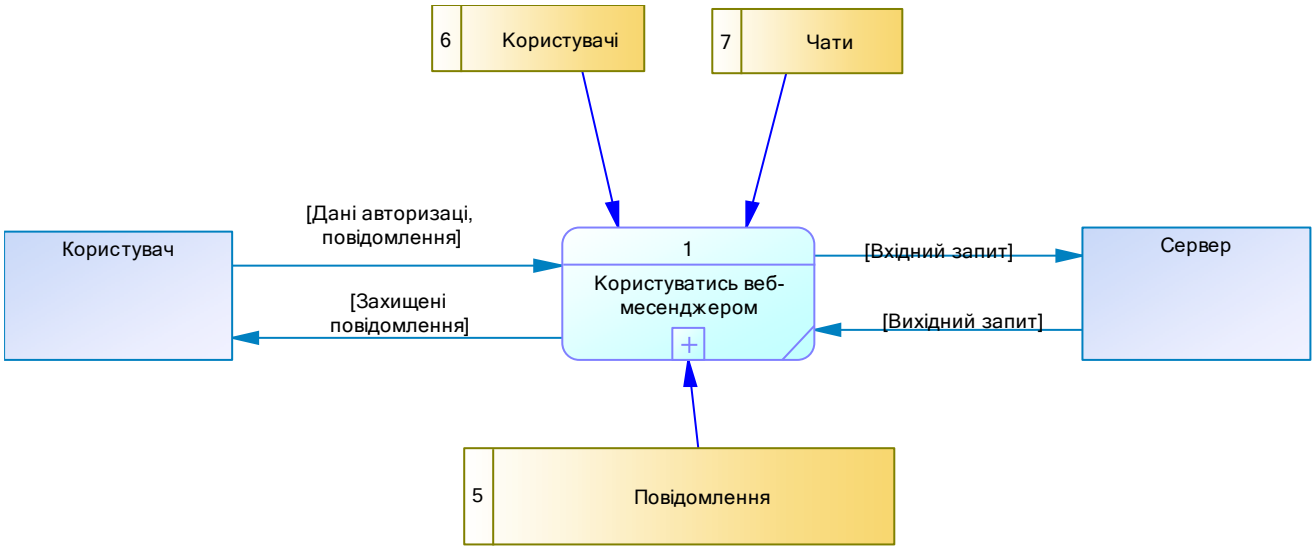


Рис. 2.10. DFD діаграма нульового рівня

Основні процеси нульового рівня:

- Надсилання повідомлень.
- Прийом та обробка повідомлень.
- Зберігання даних користувачів та повідомлень.

DFD діаграма першого рівня

Діаграма першого рівня (рис. 2.11) деталізує основні процеси, які відбуваються у системі Web-мессенджера. Вона відображає ключові функції, зокрема реєстрацію користувачів, управління чатами та обмін повідомленнями:

- **Реєстрація (1.1):** Користувач вводить дані реєстрації, які зберігаються у базі даних.
- **Вхід (1.2):** Користувач авторизується за допомогою свого нікнейму.
- **Створення кімнати (1.4):** Користувач може створити новий чат.
- **Перегляд списку чатів (1.5):** Відображає список доступних чатів.
- **Редагування назви кімнати (Автор) (1.6):** Користувач може змінити назву чата, якщо є його автором.
- **Видалення кімнати (Автор) (1.7):** Користувач може видалити чат, якщо є його автором.
- **Вхід в кімнату (1.8):** Користувач ввійти у чат.
- **Написання повідомлення (1.9):** Користувач надсилає повідомлення до чату.

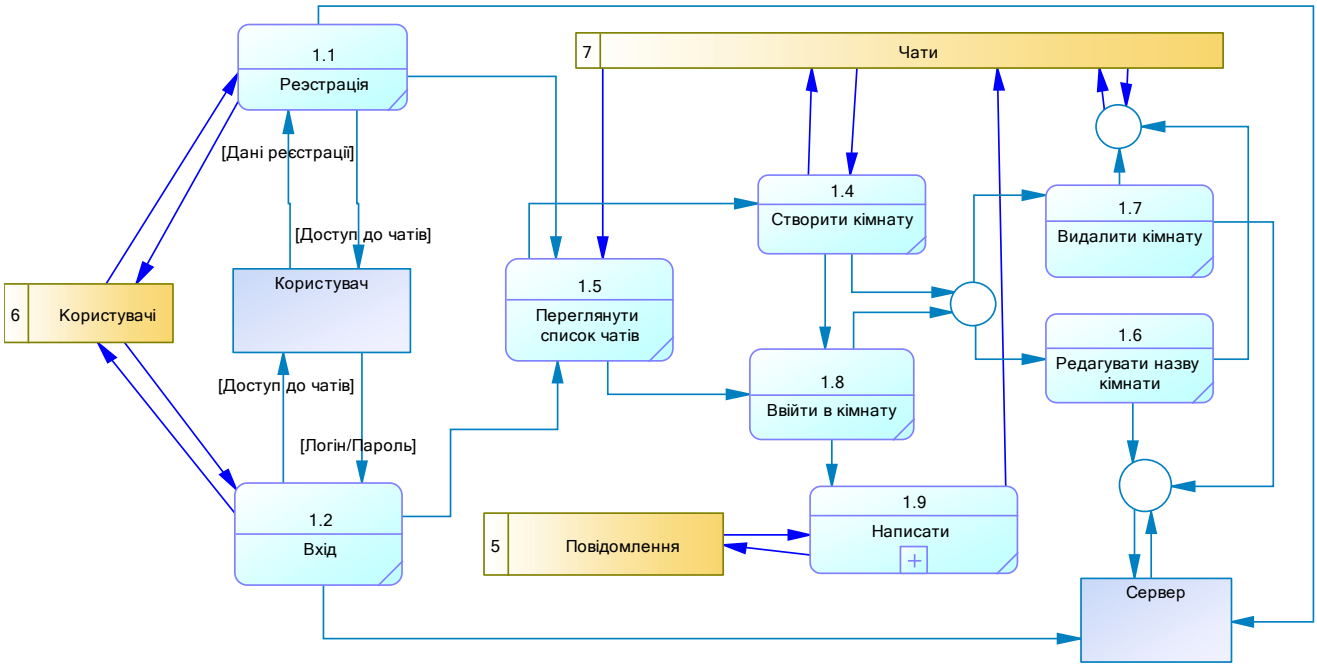


Рис. 2.11. DFD діаграма першого рівня

Ця діаграма показує, як користувачі взаємодіють із системою та яким чином сервер обробляє їхні запити.

DFD діаграма другого рівня

Діаграма другого рівня (рис. 2.12) деталізує процеси обробки повідомлень у чатах. Вона відображає, як система обробляє текстові повідомлення та забезпечує їх доставку до учасників чату.

Основні процеси другого рівня:

- **Написання тексту повідомлення (1.9.1):** Користувач створює текстове повідомлення.
- **Додавання повідомлення до чату (1.9.2):** Повідомлення додається до списку повідомлень чату.
- **Відправка повідомлень усім учасникам (1.9.3):** Система розсилає повідомлення учасникам чату.
- **Перегляд оновленого чату (1.9.4):** Користувачі бачать нові повідомлення у чаті.

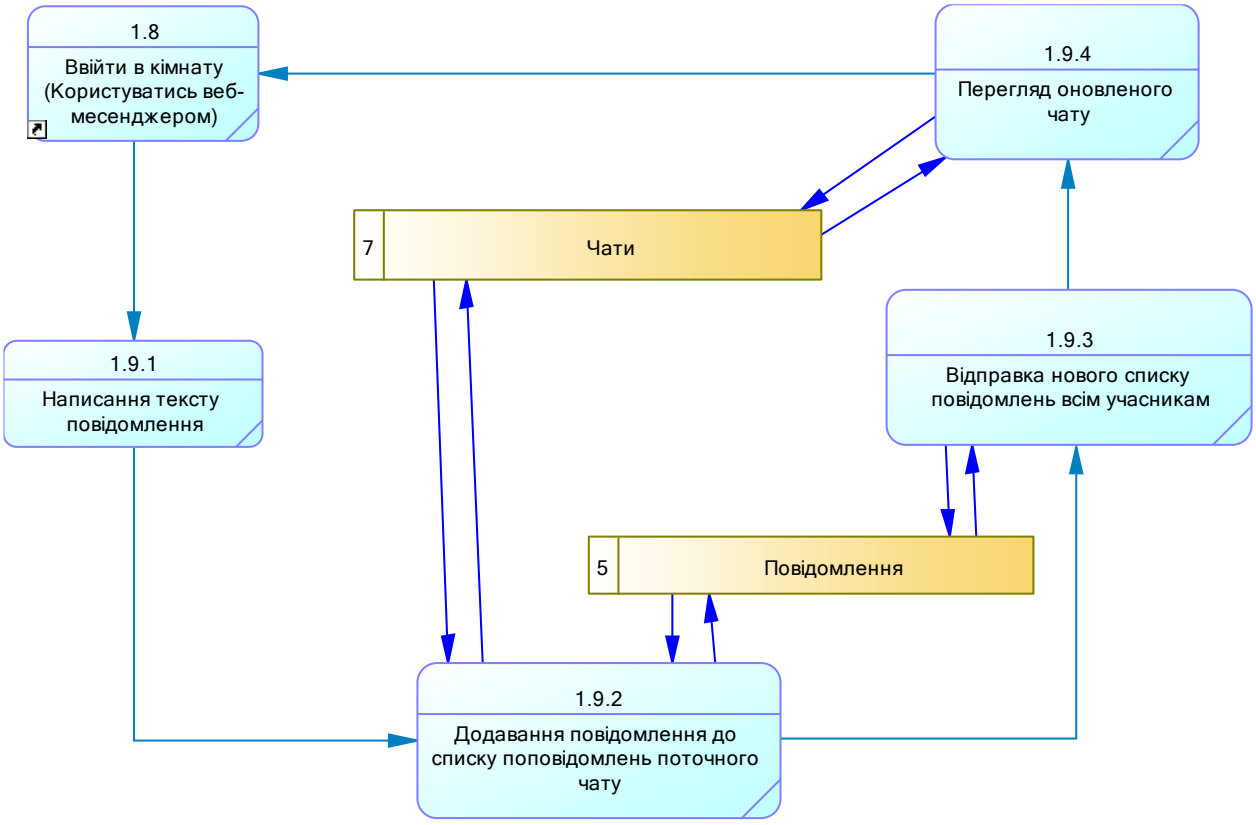


Рис. 2.12. DFD діаграма другого рівня

Ця діаграма дозволяє побачити, як повідомлення обробляються та передаються між користувачами через сервер.

2.5 Захищений обмін повідомленнями

Безпека даних є ключовим аспектом у Web-мессенджері, який забезпечує конфіденційність та захист особистих повідомлень користувачів. Для цього у системі реалізовано наскрізне шифрування (E2EE) із використанням алгоритму **Double Ratchet**, що забезпечує високий рівень захисту під час обміну повідомленнями у групових чатах. Важливо зазначити, що сервер зберігає отримані повідомлення у зашифрованому вигляді, що унеможливлює їх перегляд навіть адміністрацією сервера.

Принципи шифрування: E2EE, Double Ratchet

Наскрізне шифрування (E2EE, End-to-End Encryption) є методом захисту даних, за якого повідомлення шифрується на стороні відправника і розшифровується лише на стороні отримувачів у груповому чаті [21] (табл. 2.19).

Таблиця 2.19

Основні принципи наскрізного шифрування (E2EE)

Принцип	Опис
Прозорість	Користувачі бачать лише розшифровані повідомлення, тоді як сервер має справу із зашифрованими даними [21].
Захист від перехоплення	Повідомлення передаються у зашифрованому вигляді, що унеможливлює їх перехоплення третіми сторонами.
Генерація ключів	Повідомлення шифрується загальним ключем чату, а не індивідуальними ключами для кожної пари користувачів.
Зберігання повідомлень	Сервер зберігає повідомлення у базі даних вже у зашифрованій формі [22].
Доступність	Усі користувачі Web-мессенджера мають доступ до групових чатів, але лише учасники чату можуть переглядати повідомлення [22].

Алгоритм Double Ratchet забезпечує безпеку шифрування повідомлень навіть у багатокористувацьких чатах. Основні особливості його використання у групових чатах (табл. 2.20):

Таблиця 2.20

Основні принципи алгоритму Double Ratchet у групових чатах

Етап	Опис
Початкова генерація ключа	Сервер генерує початковий ключ для шифрування групового чату [21].
Шифрування повідомлення	Кожне повідомлення шифрується новим ключем, згенерованим на основі попереднього ключа.
Зберігання повідомлень	Сервер приймає вже зашифровані повідомлення та зберігає їх у зашифрованому вигляді.
Оновлення ключів	Для кожного нового повідомлення генерується новий ключ за допомогою KDF (Key Derivation Function).
Розшифрування повідомлень	Повідомлення розшифровуються на стороні клієнтів після їх отримання.

На рисунку 2.13 зображено процес оновлення ключів у алгоритмі Double Ratchet, який поєднує два механізми: симетричний та асиметричний [23].

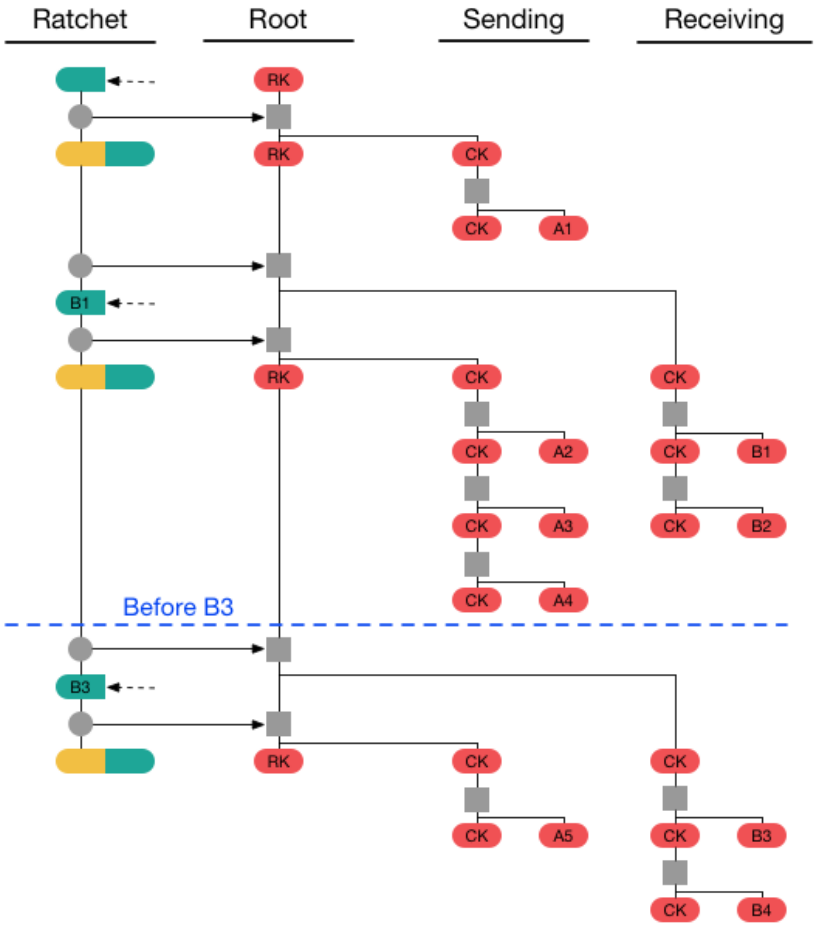


Рис. 2.13. Схема роботи алгоритму Double Ratchet

Детальний опис процесу шифрування повідомлень у чатах

Наведена нижче таблиця більш деатльно описує процес шифрування повідомлень та взаємодії користувачів із сервером під час обміну повідомленнями (табл. 2.21).

Таблиця 2.21

Детальний опис шифрування

Етап	Опис
Підключення нового користувача до чату	Сервер генерує для нового користувача загальний ключ шифрування чату та передає його користувачу у зашифрованому вигляді.
Надсилання повідомлення	Користувач вводить текст повідомлення, яке шифрується на стороні клієнта за допомогою поточного ключа чату.
Передача повідомлення	Зашифроване повідомлення надсилається через WebSocket-сервер до бази даних, де зберігається у зашифрованій формі.
Отримання повідомлення іншими учасниками	Користувачі отримують повідомлення та розшифровують його за допомогою загального ключа чату, який зберігається на їхніх пристроях.
Оновлення ключів після кожного повідомлення	Для кожного нового повідомлення автоматично генерується новий ключ шифрування за допомогою механізму KDF.
Забезпечення forward secrecy (прямої секретності)	Навіть у разі компрометації одного ключа, інші повідомлення залишаються захищеними завдяки унікальним ключам для кожного повідомлення.

Основні переваги цього підходу:

- **Конфіденційність:** Повідомлення зберігаються у зашифрованому вигляді та розшифровуються лише на пристроях користувачів.
- **Масштабованість:** Всі користувачі групового чату використовують один загальний ключ, що спрощує управління шифруванням.
- **Захист від компрометації:** Навіть якщо один із ключів буде скомпрометовано, це не вплине на захист інших повідомлень.

2.6 Висновки до розділу 2

У цьому розділі було здійснено аналіз та опис процесу проєктування архітектури захищеного Web-мессенджера для обміну повідомленнями у реальному часі. Розглянуто основні методи та технології, використані у розробці, проаналізовано архітектурні рішення та засоби забезпечення безпеки даних користувачів. Основні результати роботи у цьому розділі включають:

- **Обґрунтування архітектури:** Вибрано клієнт-серверну архітектуру з WebSocket-сервером, що забезпечує миттєвий обмін повідомленнями та масштабованість.
- **Розробка моделей даних:** Створено концептуальну та фізичну моделі, що визначають основні сутності системи — користувачі, чати та повідомлення, їхні атрибути та зв'язки.
- **Вибір методу захищеного обміну повідомленнями:** Обрано алгоритм Double Ratchet для наскрізного шифрування (E2EE) у групових чатах, що забезпечує forward secrecy.
- **Захист даних користувачів:** Повідомлення зберігатимуться на сервері у зашифрованому вигляді, а ключі для їх розшифрування — лише на пристроях користувачів.
- **Моделювання потоків даних:** Розроблено DFD-діаграми, що відображають основні процеси системи — створення повідомлень, обробка запитів, авторизація та зберігання даних.
- **Аналіз безпекових аспектів:** Розглянуто механізм шифрування повідомлень, забезпечення forward secrecy та захисту даних користувачів у групових чатах.

Запропонована структура та методи захисту забезпечують високу конфіденційність та безпеку обміну повідомленнями між користувачами, а також гарантують зручність і масштабованість системи. Це дозволяє використовувати Web-мессенджер не лише для особистого, а й для корпоративного спілкування, де захист даних є критично важливим.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ WEB-МЕССЕНДЖЕРА

3.1 Вибір інструментів та технологічного стеку розробки

Розробка сучасного захищеного Web-мессенджера для обміну повідомленнями в реальному часі вимагає ретельно підібраного набору інструментів, що забезпечує гнучкість, надійність, масштабованість та зручність у підтримці. У межах цієї дипломної роботи було використано перевірені технології, які дозволили реалізувати систему, що задовольняє функціональні вимоги, відповідає актуальним стандартам інформаційної безпеки та забезпечує ефективну взаємодію з користувачем.

Вибір засобів реалізації клієнтської частини (frontend)

Клієнтська частина Web-мессенджера реалізована у вигляді односторінкового застосунку (SPA) з використанням **React [24]**, що забезпечує швидку та динамічну взаємодію з користувачем. Компонентна структура бібліотеки дозволила розділити інтерфейс на логічно незалежні елементи: список чатів, форма входу, вікно повідомлень тощо.

Для покращення структурованості та запобігання помилкам застосовано **TypeScript [25]**, що додає статичну типізацію. Керування станом реалізовано через **useState**, що дозволяє зберігати поточні дані інтерфейсу та оновлювати його без перезавантаження сторінки.

HTTP-запити до сервера виконуються за допомогою **Axios [26]**, зокрема для авторизації, створення чатів та отримання повідомлень. Оформлення інтерфейсу здійснено з використанням **SCSS [27]**, що забезпечує гнучке та чисте стилізування. Інтерфейс адаптований для роботи на різних типах пристроїв, включаючи мобільні.

Таким чином, використання React, TypeScript, SCSS та Axios дозволило створити швидкий, зручний і безпечний інтерфейс для обміну повідомленнями в реальному часі.

Вибір засобів реалізації серверної частини (backend)

Серверна частина Web-мессенджера відповідає за обробку запитів, зберігання даних, передачу повідомлень і реалізацію шифрування. В основі використано **Node.js** — платформу для створення високопродуктивних асинхронних застосунків на JavaScript [28].

Для реалізації HTTP-запитів застосовано **Express.js [29]**, який забезпечує просту маршрутизацію та підтримку REST API. Обмін повідомленнями здійснюється через **WebSocket**, реалізований за допомогою бібліотеки **ws**, що дозволяє підтримувати постійне з'єднання між клієнтом і сервером [30].

Шифрування повідомлень реалізовано за допомогою модуля **crypto**, який підтримує алгоритми AES-256-GCM, X25519 та HKDF відповідно до протоколу Double Ratchet [23]. Взаємодія з базою даних здійснюється через ORM **Sequelize**, що дозволяє працювати з моделями без написання SQL-коду [31].

Для зберігання налаштувань середовища (паролі, URI тощо) використовується **dotenv**, що дозволяє легко керувати конфігурацією системи [32].

Таким чином, використання **Node.js**, **Express**, **WebSocket**, **crypto**, **Sequelize** та dotenv забезпечило створення гнучкої, безпечної та масштабованої серверної частини системи.

Вибір системи контейнеризації та системи контролю версій

Для зручного розгортання застосовано **Docker [33]**, який дозволяє запускати клієнтську частину, сервер, базу даних і **Nginx [34]** у окремих контейнерах. Це забезпечує стабільність роботи, ізольованість компонентів і просте масштабування, зокрема на сервері **AWS EC2 [35]**.

Nginx виконує роль зворотного проксі, обробляє HTTPS-з'єднання та перенаправляє запити між фронтендом і бекендом. Всі взаємозв'язки між сервісами описано у файлі **docker-compose.yml**, що дозволяє запускати всю систему однією командою.

Для контролю версій використовується **Git**, а зберігання коду — на **GitHub**, що забезпечує історію змін і підтримку командної роботи [36].

Вибір системи управління базами даних

У якості системи управління базами даних для Web-мессенджера було обрано **PostgreSQL** — надійну об’єктно-реляційну СУБД, яка має високий рівень безпеки та добре масштабується [37]. Вона ідеально підходить для зберігання структурованих даних користувачів, чатів та ***зашифрованих повідомлень***.

Вибір середовища розробки проєкту

Розробка Web-мессенджера здійснювалася у середовищі **Visual Studio Code** — зручному та функціональному редакторі коду, який підтримує **JavaScript**, **TypeScript**, **SCSS**, **Docker**, **Git** та інші технології, що використовуються в проєкті [38] (рис. 3.1).

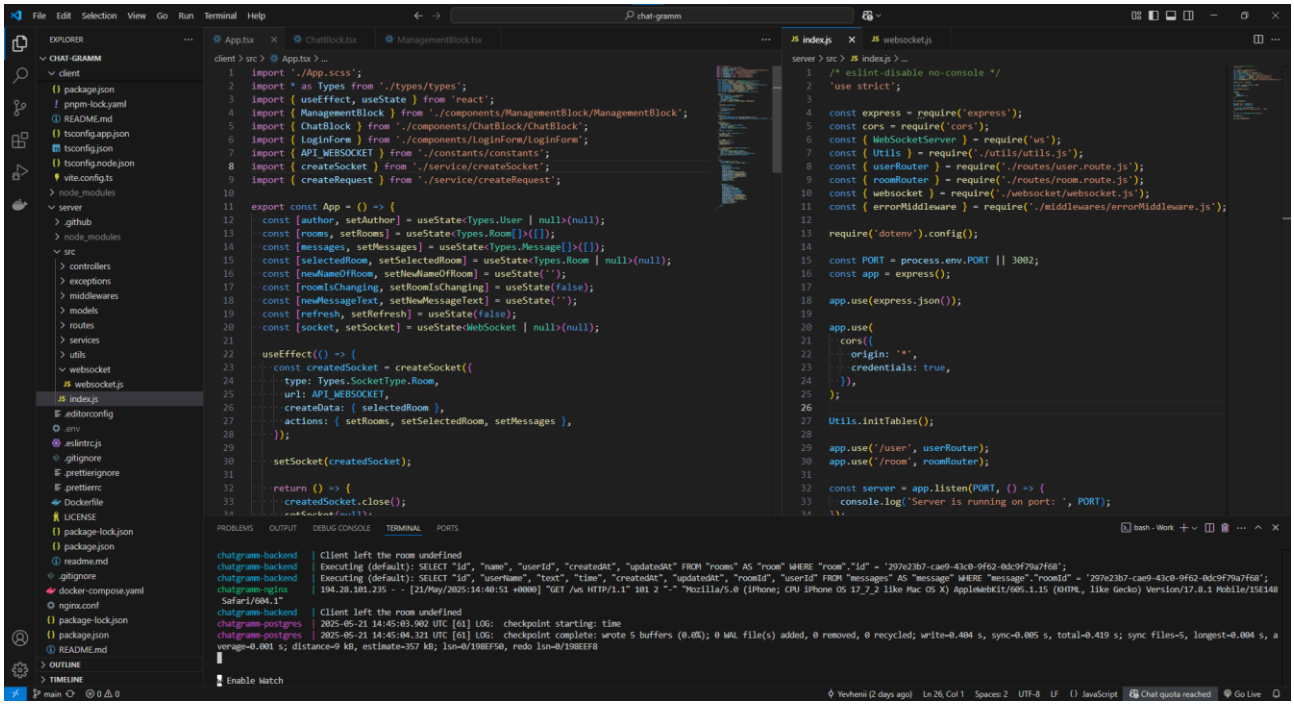


Рис. 3.1. Робочий інтерфейс VS Code

Таким чином, поєднання **React + TypeScript** на фронтенді, **Node.js + Express** на бекенді та **PostgreSQL** як СУБД забезпечило реалізацію сучасної, масштабованої та захищеної системи обміну повідомленнями. Додаткові інструменти, такі як **SCSS**, **Axios**, **WebSocket (ws)**, **crypto**, **Sequelize**, **dotenv**, **Docker**, **Git** та **Nginx** були обрані відповідно до вимог проєкту — для забезпечення інтерактивності, безпеки, зручності розгортання на **AWS** та ефективної розробки.

3.2 Загальна архітектура системи

Архітектура захищеного Web-мессенджера реалізована у вигляді **контейнеризованої клієнт-серверної системи**, де кожен компонент розгортається як окремий Docker-контейнер на хмарній платформі **AWS [35]**. Такий підхід забезпечує модульність, масштабованість, ізоляцію середовищ та зручність у розгортанні (табл. 3.1).

Основна комунікація між клієнтами та сервером здійснюється за допомогою **WebSocket-з'єднань** для передачі повідомлень у реальному часі, а також через **REST API** для обробки запитів, пов'язаних із реєстрацією, створенням чатів тощо. Компоненти системи взаємодіють через внутрішню Docker-мережу [33] та взаємопов'язані через конфігураційний файл docker-compose.yml.

Таблиця 3.1

Основні компоненти архітектури

Компонент	Опис
Client (frontend)	Реалізований на базі React із використанням TypeScript [25]. Відповідає за взаємодію з користувачем, відображення інтерфейсу, шифрування повідомлень та надсилання їх на сервер. Запити надсилаються через REST API.
Nginx	Виконує функцію зворотного проксі-сервера [34]. Приймає зовнішні запити від клієнта, маршрутизує їх на backend або frontend, а також забезпечує шифрування HTTPS-з'єднання.
Server (backend)	Написаний на Node.js [28] з використанням Express.js [29]. Обробляє REST API-запити, керує обліковими записами, чатами, повідомленнями. Має вбудований WebSocket-сервер для обміну повідомленнями в режимі реального часу [30].
WebSocket	Інтегрований у сервер. Забезпечує постійне двостороннє з'єднання між клієнтами для обміну зашифрованими повідомленнями без затримок.
Database	Сховище даних, у якому зберігається інформація про користувачів, кімнати та зашифровані повідомлення. Сервер взаємодіє з базою даних через ORM Sequelize [31].
Docker	Кожен з компонентів (frontend, backend, nginx, база даних) ізольовано у власному контейнері. Це спрощує розгортання, оновлення та обслуговування застосунку [33].

На рисунку 3.2 нижче представлено логічну структуру Web-мессенджера, яка демонструє взаємодію між основними компонентами системи:

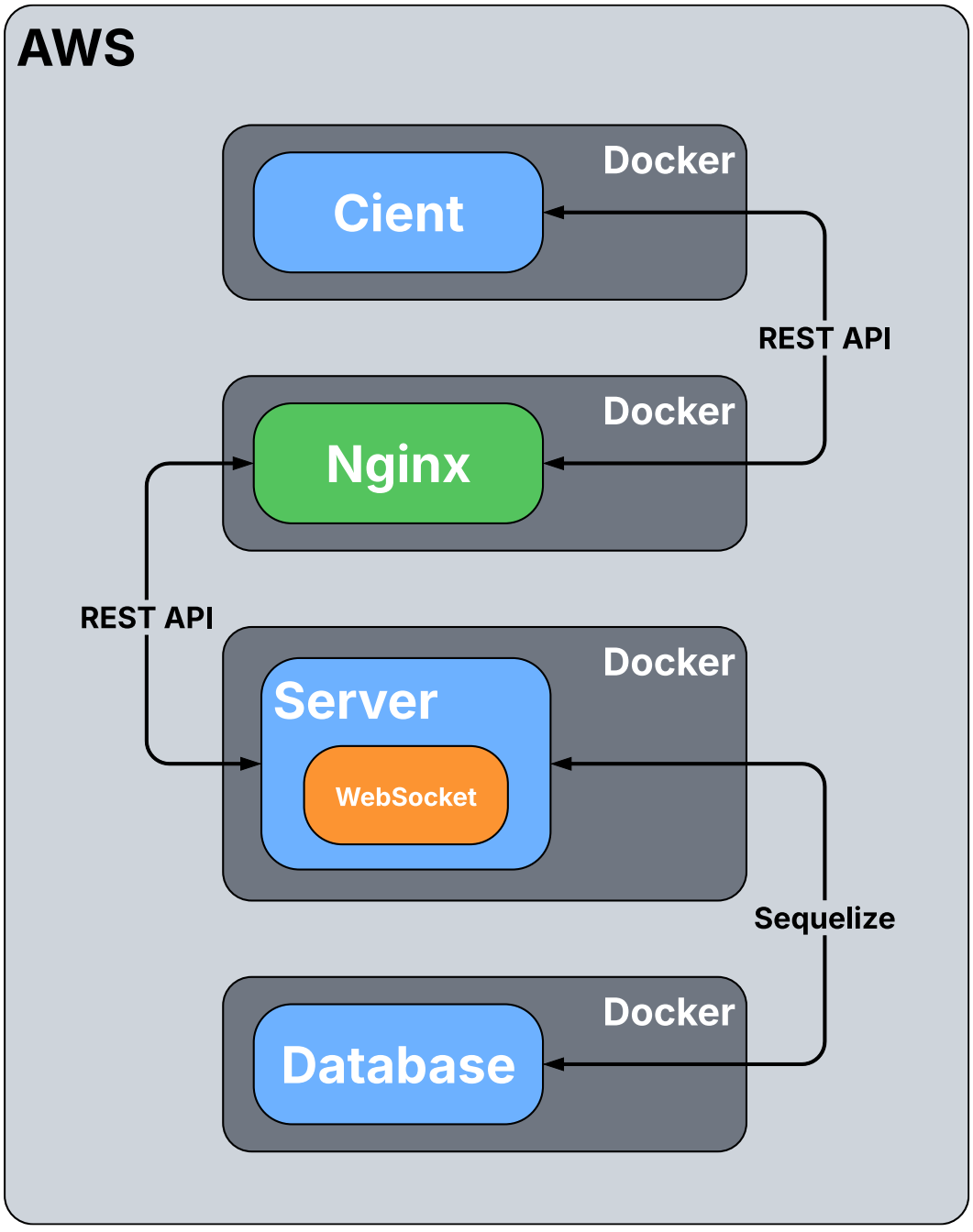


Рис. 3.2. Архітектура Web-мессенджера у Docker-середовищі

Взаємозв'язок компонентів (відповідно до діаграми):

- **Client** → **Nginx** (REST API): запити до інтерфейсу та API.
- **Nginx** → **Server**: маршрутизація запитів до бекенду.
- **Server** ↔ **Client**: обмін повідомленнями через WebSocket.
- **Server** ↔ **Database**: зберігання даних через Sequelize.

3.3 Реалізація клієнтської частини (frontend)

Клієнтська частина Web-мессенджера реалізована як односторінковий застосунок (SPA) на основі **React [24]** із використанням **TypeScript [25]**. Вона відповідає за обробку дій користувача, відображення інтерфейсу, взаємодію з сервером через REST API, ініціалізацію WebSocket-з'єднання [30] та виконання наскрізного шифрування повідомлень на боці клієнта [21].

Основний код клієнта розміщено у структурованій директорії src, де логіка розділена на компоненти, сервіси, константи та типи. Окрему увагу приділено модульності — кожен елемент інтерфейсу реалізований у вигляді окремого компонента (рис. 3.3).

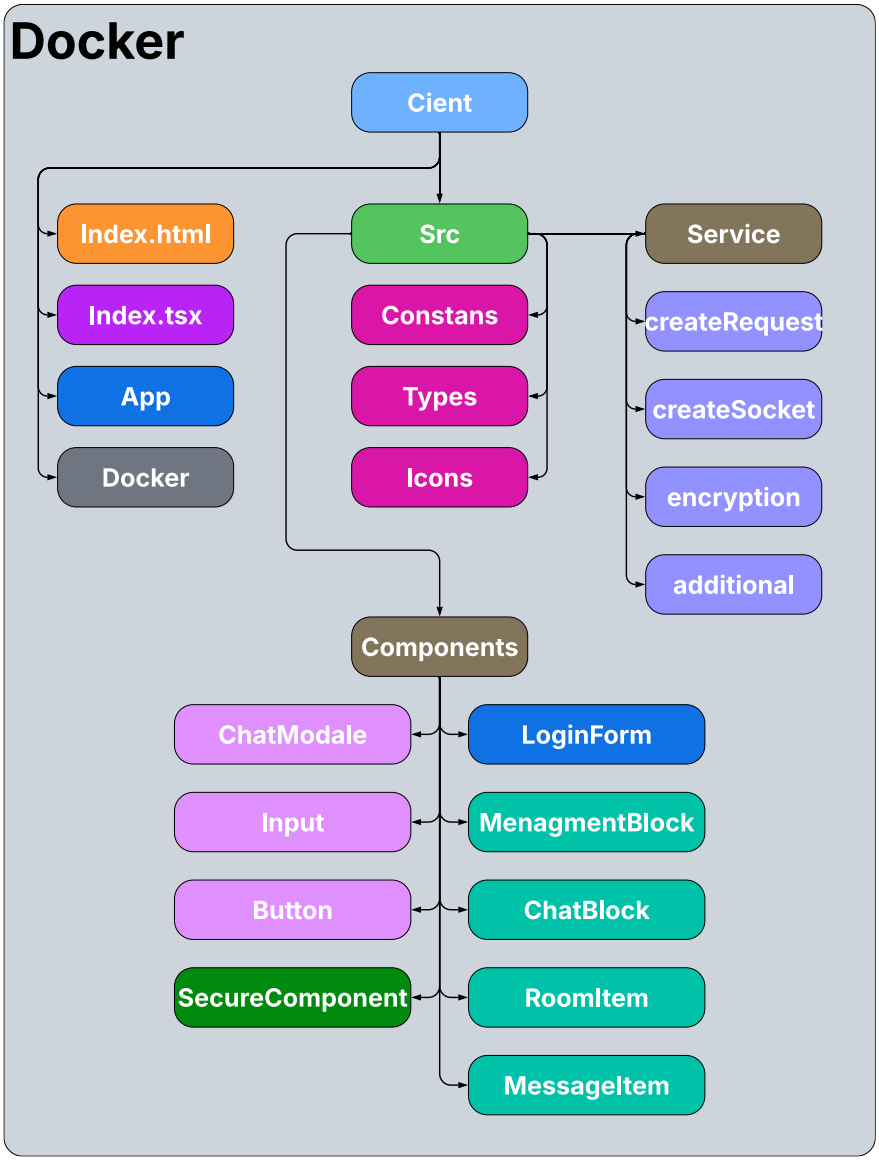


Рис. 3.3. Архітектура клієнтської частини Web-мессенджера

Опис основних компонентів клієнта

Клієнтська частина Web-мессенджера реалізована як набір модулів, що логічно поділені на компоненти інтерфейсу та службові сервіси. Такий підхід забезпечує зрозумілу структуру, легку підтримку й повторне використання коду. У таблиці 3.2 наведено короткий опис ключових компонентів клієнта.

Таблиця 3.2

Опис основних сервісів та компонентів клієнта

Компонент / Модуль	**Призначення**
Service/createRequest.ts	Формування REST-запитів до сервера: авторизація, створення чатів, отримання даних.
Service/createSocket.ts	Ініціалізація WebSocket-з'єднання, обробка подій: init, newMessage, senderKey тощо.
Service/encryption.ts	Шифрування та дешифрування повідомлень за алгоритмом AES-256-GCM [21].
Service/additional.ts	Допоміжні функції: генерація ID, форматування дати, перевірка вхідних даних.
Components/ChatModal	Модальне вікно для створення та редагування кімнат.
Components/Input	Компонент текстового поля для форми входу, надсилання повідомлень тощо.
Components/Button	Універсальний компонент кнопки з варіативним оформленням.
Components/SecureComponent	Візуальний компонент, який приховує список чатів до авторизації користувача.
Components/LoginForm	Форма введення імені користувача, ініціалізація сесії.
Components/ManagementBlock	Блок управління чатами: створення, *редагування, видалення* (*лише для авторів*).
Components/ChatBlock	Основний візуальний блок чату: історія повідомлень, поле введення, кнопка відправки.
Components/RoomItem	Компонент кімнати у списку доступних чатів.
Components/MessageItem	Компонент повідомлення з автором, часом, текстом та стилізацією.

Шифрування повідомлень

Шифрування повідомлень у клієнтській частині мессенджера реалізоване за принципами **алгоритму Double Ratchet [21]**, який поєднує симетричне шифрування (**AES-256-GCM**) із механізмом **оновлення ключів (KDF) [21]**. Кожне повідомлення шифрується ключем шифрування в **sessionStorage** учасників чата, при цьому для кожного повідомлення генерується новий вектор ініціалізації (**IV**), що забезпечує унікальність шифрування (рис. 3 4).

```
export async function encryptMessage(text: string, key: string): Promise<string> {
  const encoder = new TextEncoder();
  const data = encoder.encode(text);

  const cryptoKey = await crypto.subtle.importKey(
    "raw",
    Uint8Array.from(atob(key), (c) => c.charCodeAt(0)),
    { name: "AES-GCM" },
    false,
    ["encrypt"]
  );

  const iv = crypto.getRandomValues(new Uint8Array(12));
  const encryptedData = await crypto.subtle.encrypt(
    { name: "AES-GCM", iv },
    cryptoKey,
    data
  );

  return JSON.stringify({
    iv: Array.from(iv),
    data: btoa(String.fromCharCode(...new Uint8Array(encryptedData)))
  });
}
```

Рис. 3.4. Реалізація методу шифрування помідомлень

Опис етапів (рис. 3.5):

1. Користувач вводить текст повідомлення.
2. Дані перетворюються у байтовий масив.
3. Ключ кімнати (base64) імпортується в AES-GCM формат.
4. Генерується унікальний IV.
5. Повідомлення шифрується.
6. Результат пакується в JSON: { iv, data }.
7. Застосовується **KDF** для оновлення симетричного ключа чата.

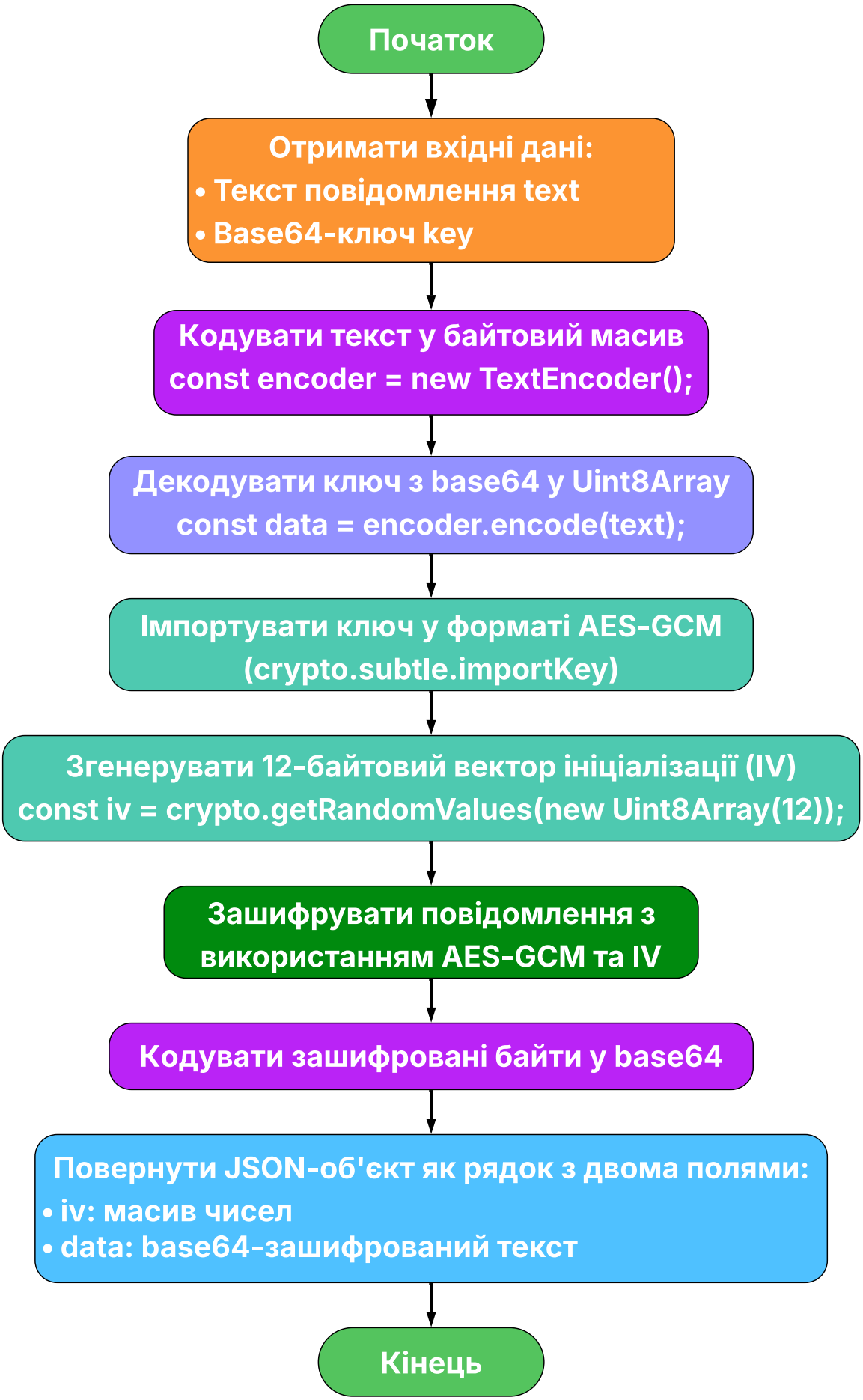


Рис. 3.5. Алгоритм шифрування повідомлення

Дешифрування повідомлень

Дешифрування відбувається на стороні клієнта одразу після отримання повідомлення з сервера. Усі повідомлення приходять у форматі **JSON (iv + data)** та не розшифровуються на сервері, що відповідає моделі **end-to-end encryption**.

Для розшифрування застосовується той самий ключ, що був збережений у **sessionStorage** під час входу в кімнату. **IV** з повідомлення використовується для ініціалізації AES-GCM, після чого дані розшифровуються (рис. 3.6). Ключ одразу ж оновлюється за допомогою **KDF**, що не дозволяє використовувати його повторно — навіть користувач, що отримає доступ до пам'яті пристрою після дешифрування, не зможе відновити ключі попередніх чи наступних повідомлень [21].

```
export async function decryptMessage(encryptedText: string, key: string): Promise<string> {
  const encryptedData: { iv: number[]; data: string } = JSON.parse(encryptedText);

  const cryptoKey = await crypto.subtle.importKey(
    "raw",
    Uint8Array.from(atob(key), (c) => c.charCodeAt(0)),
    { name: "AES-GCM" },
    false,
    ["decrypt"]
  );

  const iv = new Uint8Array(encryptedData.iv);
  const encryptedArray = Uint8Array.from(atob(encryptedData.data), (c) => c.charCodeAt(0));
  const decryptedData = await crypto.subtle.decrypt(
    { name: "AES-GCM", iv },
    cryptoKey,
    encryptedArray
  );

  return new TextDecoder().decode(decryptedData);
}
```

Рис. 3.6. Реалізація методу дешифрування помідомлень

Опис етапів (рис. 3.7):

1. Отримання JSON-структури (iv, data).
2. Імпорт симетричного ключа.
3. Перетворення IV та даних у байтові масиви.
4. Розшифрування повідомлення через AES-GCM.
5. Перетворення байтів у текст (TextDecoder).
6. **Оновлення ключа через KDF** для збереження forward secrecy.

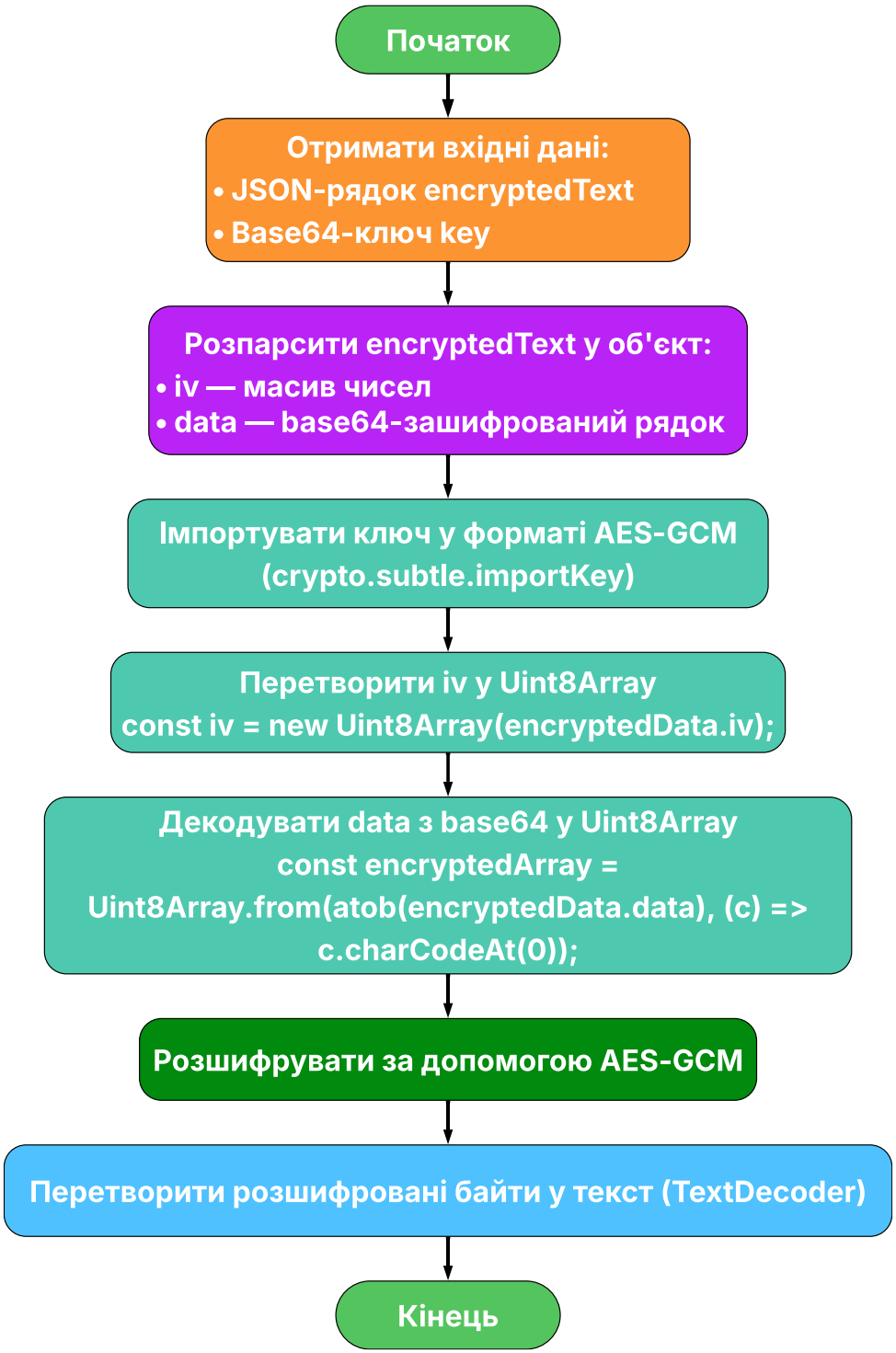


Рис. 3.7. Алгоритм дешифрування повідомлення

Таким чином, застосування алгоритму **Double Ratchet [21]** дозволяє:

- гарантувати **унікальність ключа для кожного повідомлення**;
- забезпечити **forward secrecy** (розкриття одного ключа не дає доступу до інших);
- реалізувати **end-to-end encryption** у групових чатах.

3.4 Реалізація серверної частини (backend)

Серверна частина Web-мессенджера реалізована з використанням **Node.js [28]** — платформи, що дозволяє створювати високопродуктивні асинхронні застосунки, та **Express.js** — мінімалістичного фреймворку для побудови **REST API [29]**. Для підтримки зв'язку в реальному часі використано **WebSocket [30]**, а для взаємодії з базою даних — **ORM Sequelize**, яка спрощує роботу з реляційними таблицями PostgreSQL [31, 37]. Система розділена на модулі, що відповідають за маршрутизацію, обробку запитів, бізнес-логіку, моделі даних та утиліти.

Архітектура backend частини побудована за принципом **чіткої модульності**, де кожен шар виконує конкретну функцію в системі (рис. 3.8).

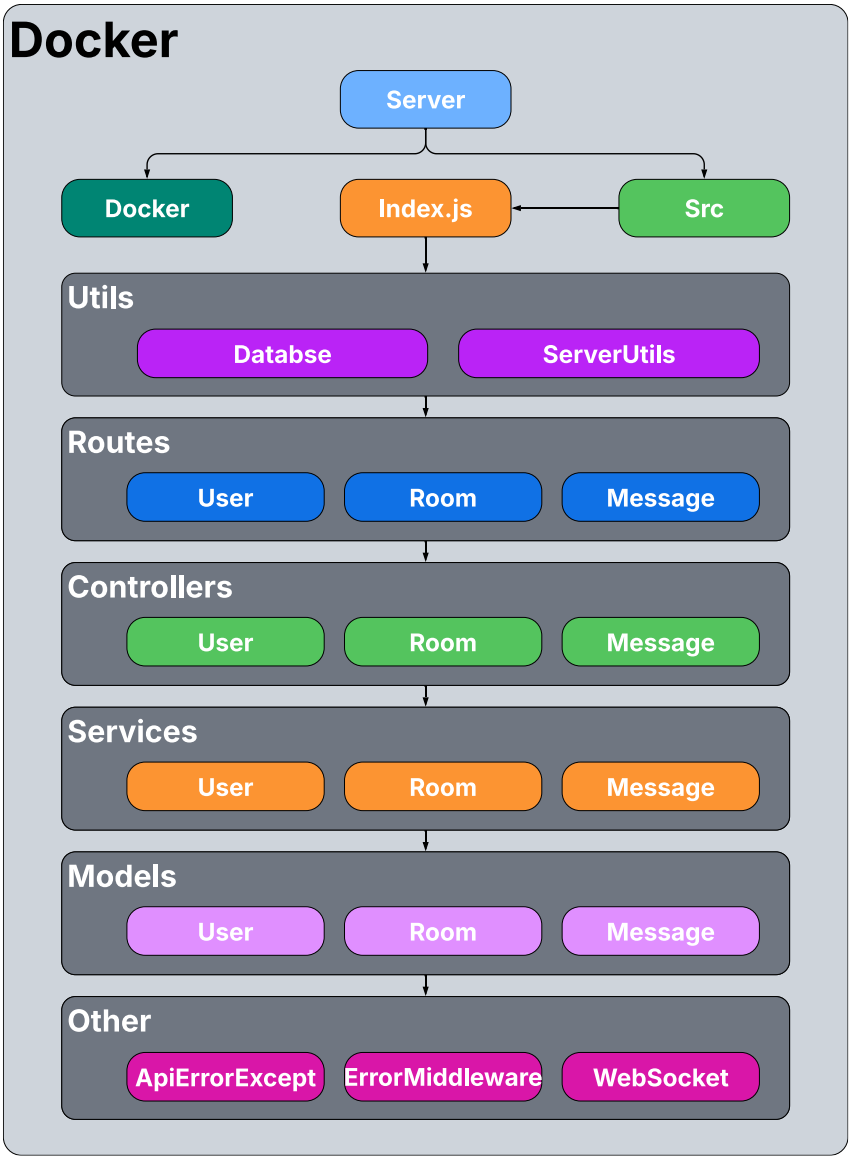


Рис. 3.8. Архітектура серверної частини Web-мессенджера

Опис основних компонентів сервера

Код серверної частини Web-мессенджера структуровано у вигляді окремих модулів, які логічно поділяються за призначенням: маршрутизація, обробка запитів, бізнес-логіка, робота з моделями та допоміжні утиліти. Така модульна структура забезпечує масштабованість, зрозуміле розділення відповідальностей та зручність у супроводі проєкту. У таблиці 3.3 наведено короткий опис основних компонентів серверної частини.

Таблиця 3.3

Опис основних сервісів та компонентів сервера

Компонент	Призначення
Index.js	Головний файл запуску застосунку. Ініціалізує Express, маршрути, WebSocket, підключення до БД.
Routes	Файли маршрутизації (user, room, message), які передають запити до контролерів.
Controllers	Обробники запитів: приймають вхідні дані, викликають відповідні сервіси.
Services	Містять бізнес-логіку застосунку: створення чатів, користувачів, повідомлень тощо.
Models	Sequelize-моделі таблиць бази даних: User, Room, Message.
Utils/Database	Налаштування та ініціалізація підключення до бази даних.
Utils/ServerUtils	Допоміжні функції, що використовуються під час запуску сервера або ініціалізації.
ApiErrorExcept	Клас для створення структурованих помилок API.
ErrorMiddleware	Глобальний middleware для обробки помилок і формування відповіді.
WebSocket	Логіка для обробки WebSocket-подій: ініціалізація, розсилка повідомлень, розподіл ключів тощо.

Опис REST API

REST API забезпечує обробку клієнтських запитів, що надсилаються з фронтенду до серверної частини Web-мессенджера. Інтерфейс реалізовано за допомогою Express.js [29] і структуровано за принципом **ресурсно-орієнтованого підходу**, де кожен тип сутності (користувачі, кімнати, повідомлення) обробляється власним маршрутизатором (рис. 3.9).

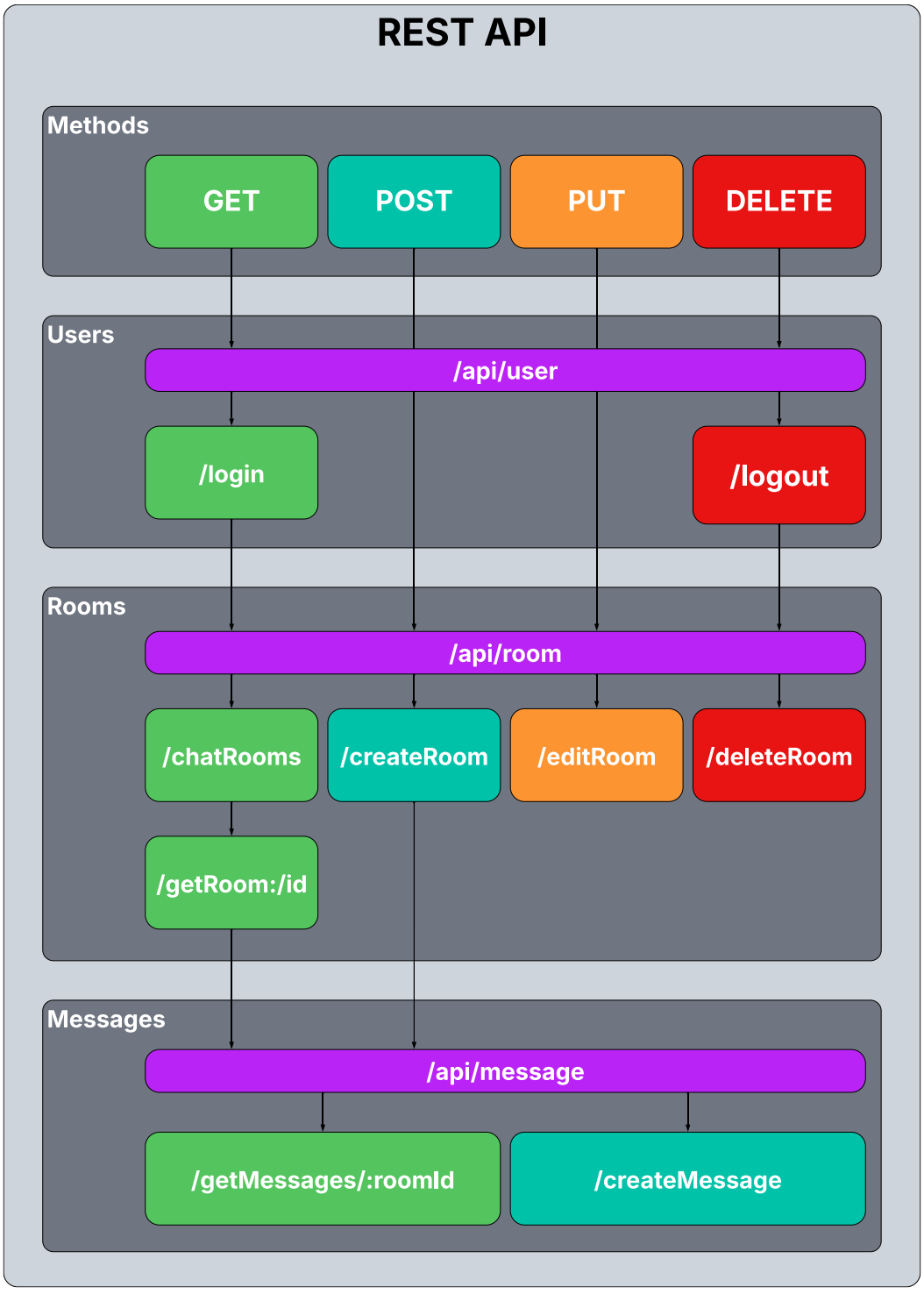


Рис. 3.9. Структура REST API Web-мессенджера

Маршрути групуються відповідно до HTTP-методів:

- ***GET*** — отримання даних;
- ***POST*** — створення ресурсів;
- ***PUT*** — редагування існуючих;
- ***DELETE*** — видалення.

Маршрутизація користувачів *api/user наведені у таблиці 3.4*:

Таблиця 3.4

Маршрути для *api/user*

Метод	Шлях	Призначення
GET	/login	Ініціалізація користувача, створення сесії.
DELETE	/logout	Завершення сесії, відключення користувача.

Маршрутизація кімнат *api/room наведені у таблиці 3.5*:

Таблиця 3.5

Маршрути для *api/room*

Метод	Шлях	Призначення
GET	/chatRooms	Отримання списку всіх кімнат.
GET	/getRoom/:id	Ініціація входу в конкретну кімнату.
POST	/createRoom	Створення нової кімнати.
PUT	/editRoom	Зміна назви або властивостей кімнати.
DELETE	/deleteRoom	Видалення кімнати з бази.

Маршрутизація повідомлень *api/message наведені у таблиці 3.6*:

Таблиця 3.6

Маршрути для *api/message*

Метод	Шлях	Призначення
GET	/getMessages/:roomId	Отримання історії повідомлень з кімнати.
POST	/createMessage	Збереження нового повідомлення в базі.

Опис запускового модуля index.js

Файл **index.js** виконує роль центральної точки запуску серверної частини Web-мессенджера. У ньому відбувається ініціалізація всіх необхідних сервісів, підключення маршрутів, запуск WebSocket-серверу [30], а також конфігурація бази даних і обробки помилок. Саме цей модуль зв’язує між собою всю архітектуру backend'у (рис. 3.10).

Рис. 3.10. Створення та ініціалізація сервера

Вибір середовища розгортання

Для хостингу серверної частини Web-мессенджера використано хмарну платформу **Amazon Web Services (AWS)**, а саме сервіс **EC2**, що дозволяє швидко розгорнути ізольоване серверне середовище з повним контролем доступу, безпекою та масштабуванням [35] (рис. 3.11).

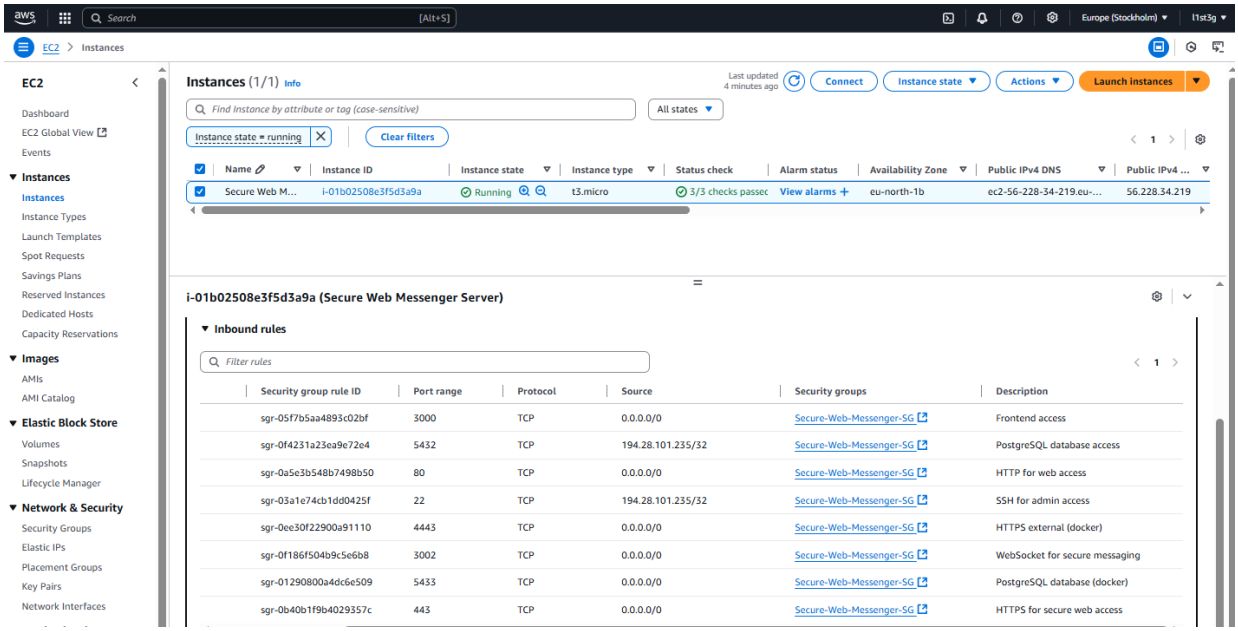


Рис. 3.11. Конфігурація EC2-сервера на AWS

У таблиці 3.7 наведено основні характеритиски середовища розгортання

Таблиця 3.7

Основні характеристики середовища розгортання

Параметр	Значення
Хостинг	Amazon Web Services (AWS)
Сервіс	EC2 (Elastic Compute Cloud)
Тип інстансу	t3.micro
ОС інстансу	Ubuntu Server
Контейнеризація	Docker + Docker Compose
Формат розгортання	Контейнери: frontend, backend, nginx, PostgreSQL
Схема доступу	Публічний IP, відкриті порти для HTTPS, WebSocket, REST API, PostgreSQL

Відкриття портів для коректної роботи

Для забезпечення стабільної та захищеної роботи Web-мессенджера на сервері AWS необхідно попередньо налаштувати **відкриті порти** у правилах безпеки інстансу. На рис. 3.12 зображено конфігурацію Inbound Rules, яка дозволяє доступ до компонентів системи через визначені протоколи та порти.

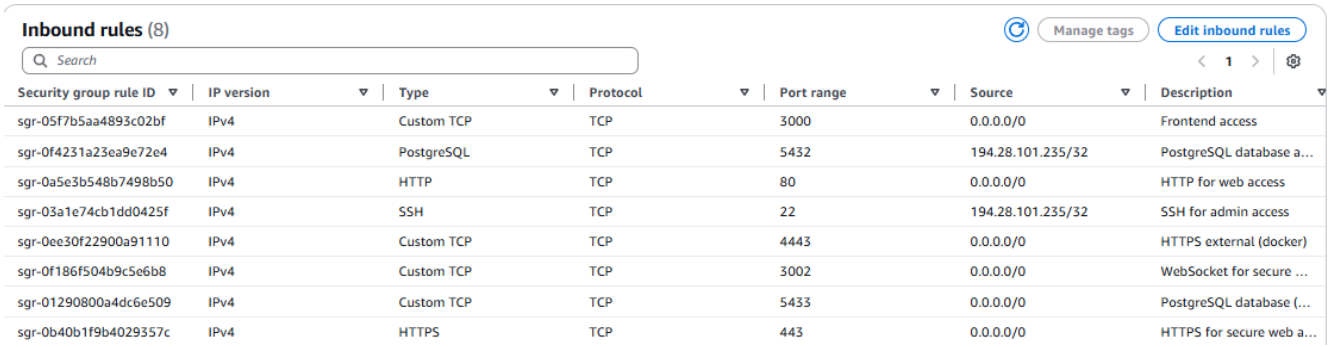


Рис. 3.12. Відкриті порти в групі безпеки EC2-сервера на AWS

Усі критично важливі для роботи застосунку порти відкриті відповідно до функцій кожного модуля. Детальна інформація наведена в таблиці 3.8.

Таблиця 3.8

Відкриті порти та їх призначення

Порт	Протокол	Джерело доступу	Призначення
3000	TCP	0.0.0.0/0	Відкритий доступ до фронтенду (розробницький режим, до проксі)
3002	TCP	0.0.0.0/0	Порт для REST API-запитів до бекенду
4443	TCP	0.0.0.0/0	Основний порт доступу до застосунку (HTTPS + WebSocket, обробляється Nginx)
5432	TCP	194.28.101.235/32	Зовнішній доступ до PostgreSQL (обмежений IP)
5433	TCP	0.0.0.0/0	Docker-доступ до PostgreSQL всередині хоста
80	TCP	0.0.0.0/0	HTTP-доступ (використовується для редиректу на HTTPS)
443	TCP	0.0.0.0/0	Альтернативний HTTPS, резервний або для майбутніх розширень
22	TCP	194.28.101.235/32	SSH-доступ до сервера лише з білого IP (для адміністрування)

3.5 База даних Web-мессенджера

У Web-мессенджері використовується реляційна система управління базами даних **PostgreSQL**, що забезпечує надійне зберігання інформації про користувачів, кімнати та повідомлення [37]. Взаємодія з базою реалізована через ORM **Sequelize**, що дозволяє описувати таблиці у вигляді моделей, автоматично створювати зв’язки між ними та зручно працювати з даними через об'єктний підхід [31].

Структура бази даних оптимізована для забезпечення швидкого доступу до повідомлень, збереження історії чатів і зручного управління правами доступу через зв’язки між користувачами та кімнатами (рис. 3.13).

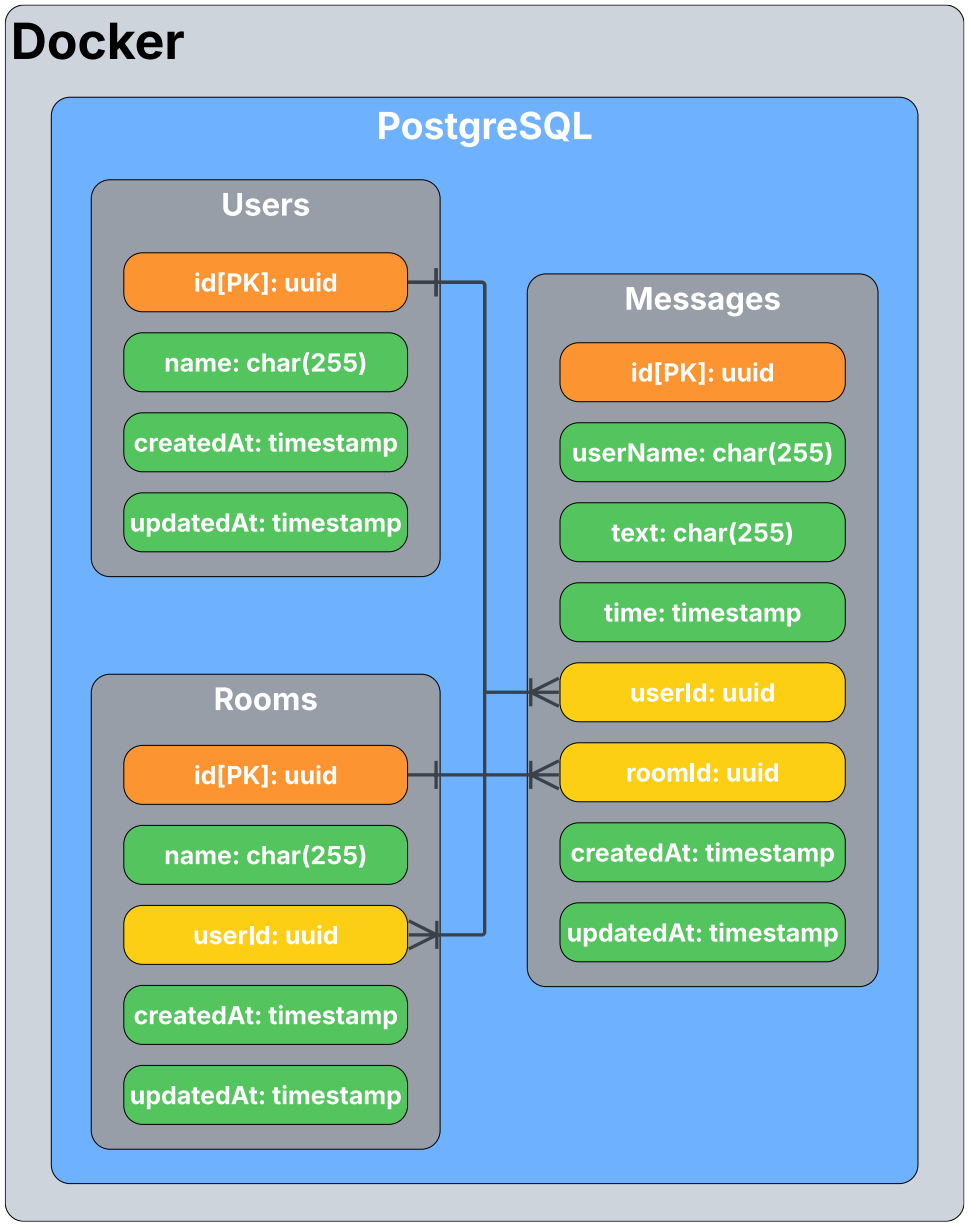


Рис. 3.13. Архітектура бази даних мобільного Web-мессенджера

Таблиця користувачів (*Users*)

У таблиці Users зберігається базова інформація про всіх користувачів системи. Структура цієї таблиці наведена нижче (табл. 3.9).

Таблиця 3.9

Структура таблиці Users

Поле	Тип даних	Опис
id	uuid	Унікальний ідентифікатор (PK)
name	char(255)	Ім’я користувача
createdAt	timestamp	Дата створення запису
updatedAt	timestamp	Дата останнього оновлення запису

Таблиця чат-кімнат (*Rooms*)

Таблиця Rooms зберігає інформацію про створені кімнати, їх назви та авторів. Структура цієї таблиці наведена нижче (табл. 3.10).

Таблиця 3.10

Структура таблиці Rooms

Поле	Тип даних	Опис
id	uuid	Унікальний ідентифікатор кімнати (PK)
name	char(255)	Назва кімнати
userId	uuid	ID автора кімнати (зв’язок із таблицею Users)
createdAt	timestamp	Дата створення кімнати
updatedAt	timestamp	Дата останнього оновлення

Таблиця повідомлень (Messages)

Таблиця Messages містить історію спілкування в кімнатах. Повідомлення прив’язані до користувачів і чатів. Структура цієї таблиці наведена нижче (табл. 3.11).

Таблиця 3.11

Структура таблиці Messages

Поле	Тип даних	Опис
id	uuid	Унікальний ідентифікатор повідомлення (PK)
userName	char(255)	Ім’я автора повідомлення
text	char(255)	Вміст повідомлення (зашифрований текст)
time	timestamp	Час надсилання повідомлення
userId	uuid	Ідентифікатор користувача (зв’язок із Users)
roomId	uuid	Ідентифікатор кімнати (зв’язок із Rooms)
createdAt	timestamp	Дата створення повідомлення
updatedAt	timestamp	Дата останнього оновлення повідомлення

Підключення до бази даних (db.js)

Файл db.js відповідає за ініціалізацію з’єднання з базою даних **PostgreSQL** за допомогою ORM-бібліотеки **Sequelize**. Усі параметри зчитуються з .env файлу: адреса хоста, порт, назва бази, логін і пароль. Встановлено діалект postgres і механізм повторного з’єднання [31, 37] (до 10 спроб) (рис. 3.14).

Після створення підключення змінна client експортується для використання в усьому застосунку, зокрема при ініціалізації таблиць.

Рис. 3.14. Конфігурація підключення до PostgreSQL

3.6 Тестування Web-мессенджера

З метою перевірки коректної роботи системи було проведено тестування Web-мессенджера на реальному прикладі спілкування двох користувачів — **Жені (десктопна версія)** та **Ані (мобільна версія)**. Тестування охоплює всі основні функціональні можливості: створення кімнати, підключення, шифрування та обмін повідомленнями в реальному часі.

Наочна демонстрація взаємодії двох користувачів у різних середовищах підтверджує, що вибір сучасного технологічного стеку (React, TypeScript, WebSocket, Node.js, PostgreSQL) дозволив реалізувати **зручний, стабільний та адаптивний** засіб комунікації.

Етап 1 – Авторизація користувачів

На першому етапі тестування двоє користувачів — **Женя (з комп'ютера)** та **Аня (з мобільного пристрою)** — проходять авторизацію у Web-мессенджері. Кожен із них вводить своє ім'я у відповідне поле форми авторизації та натискає кнопку **«Увійти»**.

Інтерфейс авторизації відображається однаково зручно як на десктопній, так і на мобільній версії. (рис. 3.15)

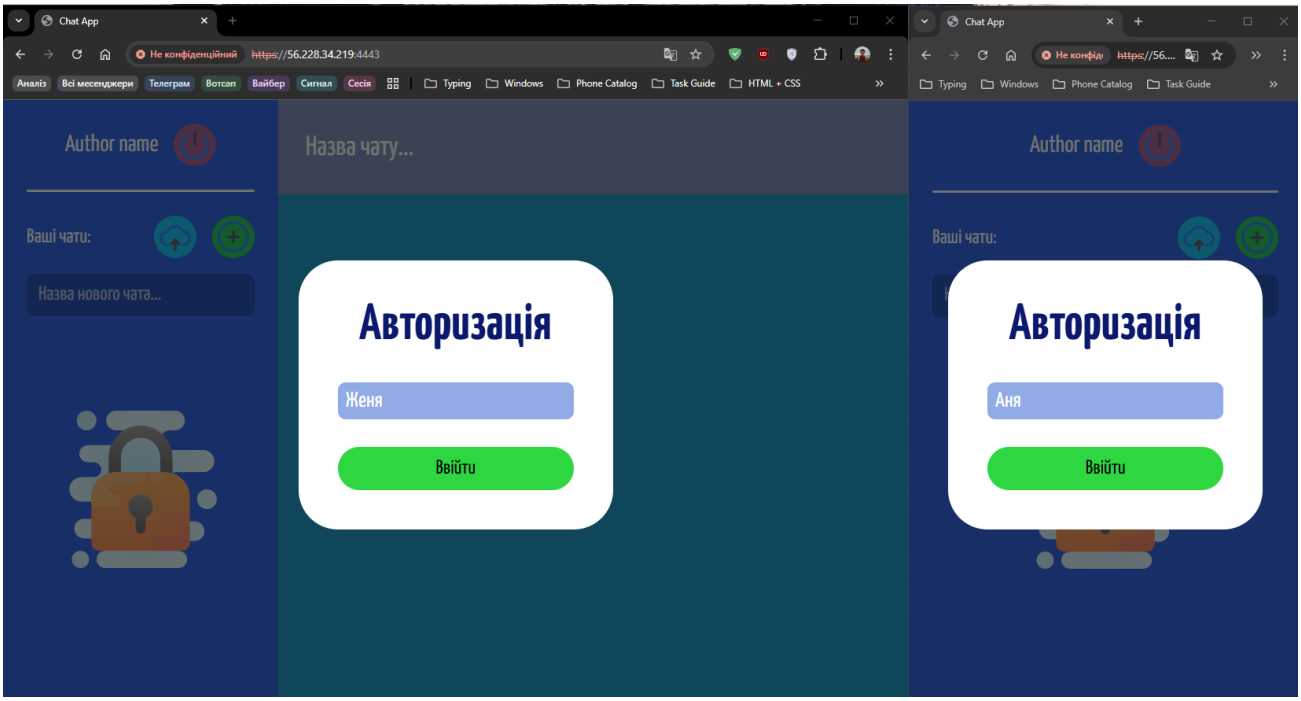


Рис. 3.15. Вхід користувачів Женя (ліворуч) і Аня (праворуч)

Етап 2 – Введення назви нової кімнати

Після авторизації користувач **Женя** на десктопі вводить назву нової кімнати — **«Тестовий чат»** — у відповідне поле. Проте на цьому етапі чат ще **не створено**, оскільки кнопка підтвердження (зелена з плюсом) не була натиснута (рис. 3.16).

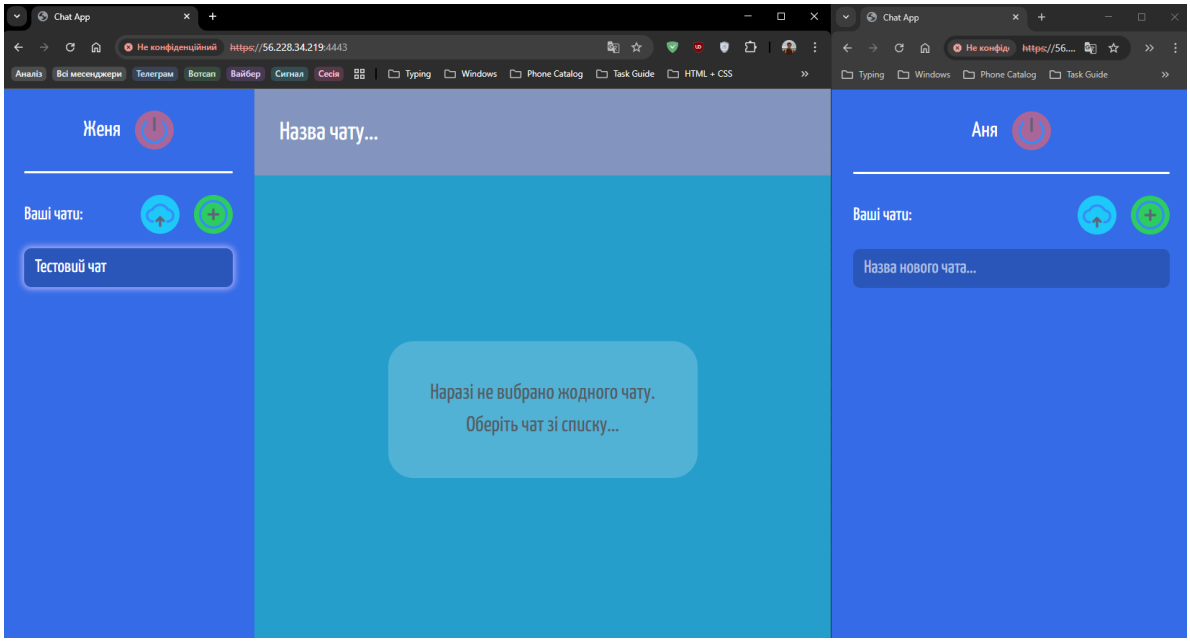


Рис. 3.16. Введення назви кімнати «Тестовий чат»

Паралельно з цим, після авторизації кожен користувач — **Женя** та **Аня** — автоматично зберігається у базі даних. На рисунку 3.17 представлено результат SQL-запиту до таблиці users, який підтверджує, що обидва користувачі були зареєстровані із унікальними uuid-ідентифікаторами, і система коректно зберегла їхні дані.

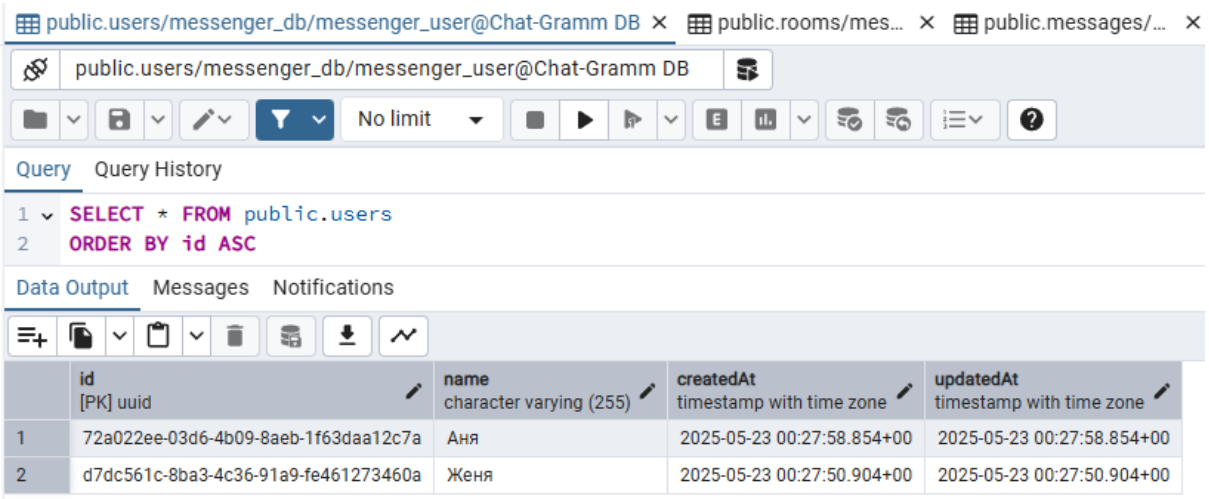


Рис. 3.17. Користувачі у таблиці users бази даних PostgreSQL

Етап 3 – Створення кімнати та підключення користувачів

Після введення назви кімнати **Женя** підтверджує її створення, у результаті чого нова кімната **«Тестовий чат»** з'являється у списку чатів в обох користувачів — Жені та Ані. Це підтверджує коректну роботу механізму створення чатів через REST API та розповсюдження оновлень через WebSocket [30]. Інтерфейс обох користувачів синхронно оновлюється (рис. 3.18).

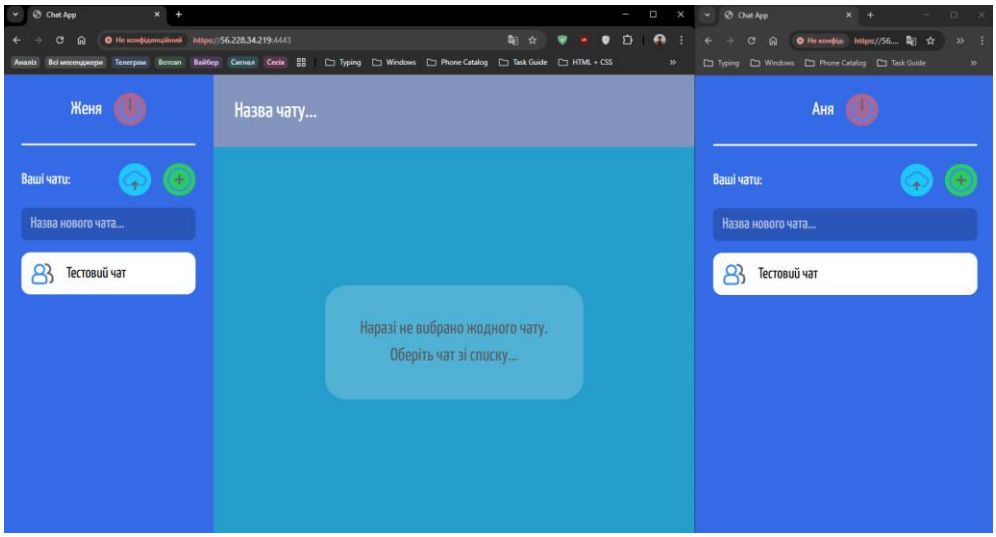


Рис. 3.18. Список чатів після створення кімнати «Тестовий чат»

Після появи кімнати обидва користувачі переходять у неї. Система повідомляє, що чат наразі порожній, і запрошує надіслати перше повідомлення. Ідентичний вигляд інтерфейсу на обох пристроях демонструє адаптивність і стабільність роботи клієнтської частини (рис. 3.19).

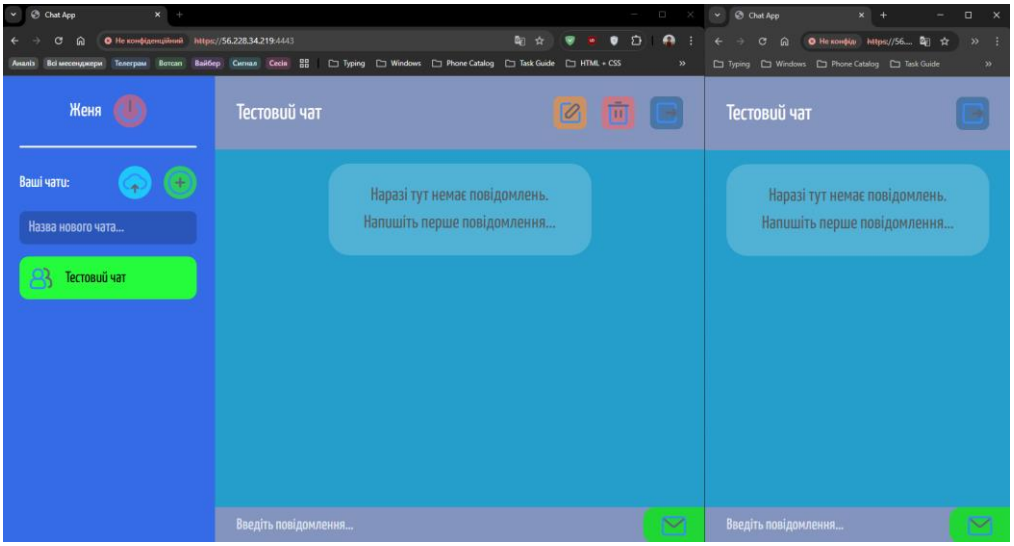


Рис. 3.19. Відображення порожнього чату на двох пристроях

Паралельно з діями користувачів у базі даних створюється новий запис у таблиці rooms. Як видно на рисунку 3.20, кімната з назвою «Тестовий чат» успішно збережена разом з унікальним id, userId автора та мітками часу створення й оновлення. Це підтверджує, що ORM Sequelize успішно обробляє запити до БД.

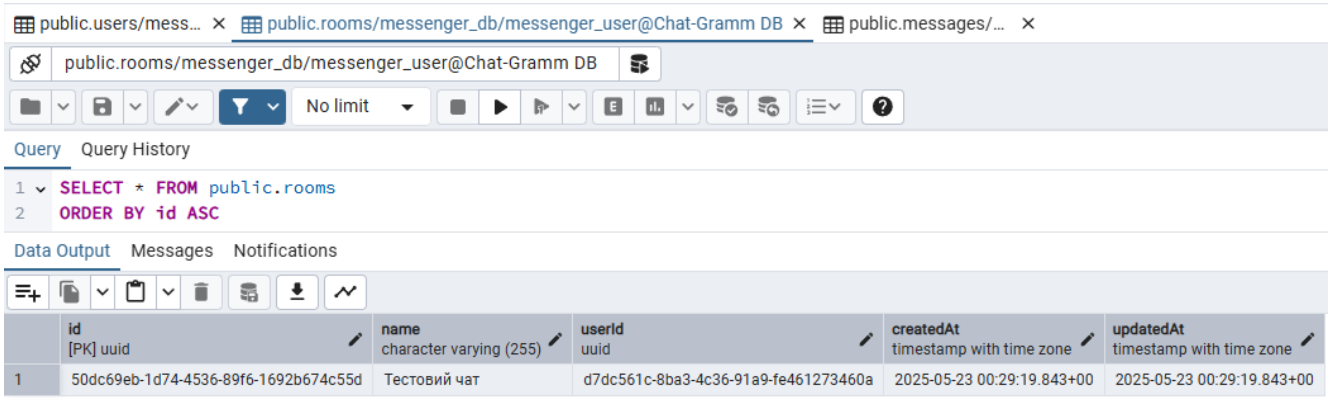


Рис. 3.20. SQL-запис у таблиці rooms після створення кімнати

Етап 4 – Обмін повідомленнями між користувачами

Після підключення до кімнати **Женя** та **Аня** починають активне спілкування. Кожен із користувачів надсилає повідомлення, які моментально з’являються на екрані обох пристроїв.

Також видно, що інтерфейс повідомлень має **адаптивний вигляд** із чітким поділом на вихідні та вхідні повідомлення, підписані іменем автора та міткою часу (рис. 3.21).

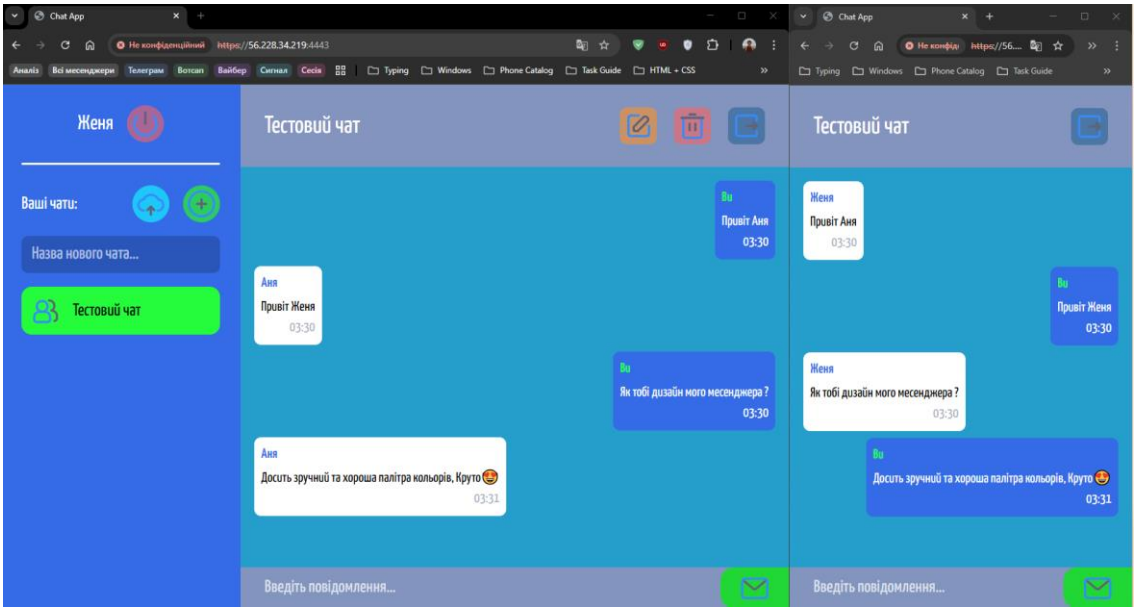


Рис. 3.21. Взаємне спілкування користувачів у кімнаті «Тестовий чат»

Етап 5 – Створення нового чату та його перейменування

Поки **Женя** залишається в існуючому чаті «**Тестовий чат**», **Аня** починає створення нового чату. Вона вводить у поле назву **«Плани на вихідні»**, готуючись до створення нової кімнати(рис. 3.22).

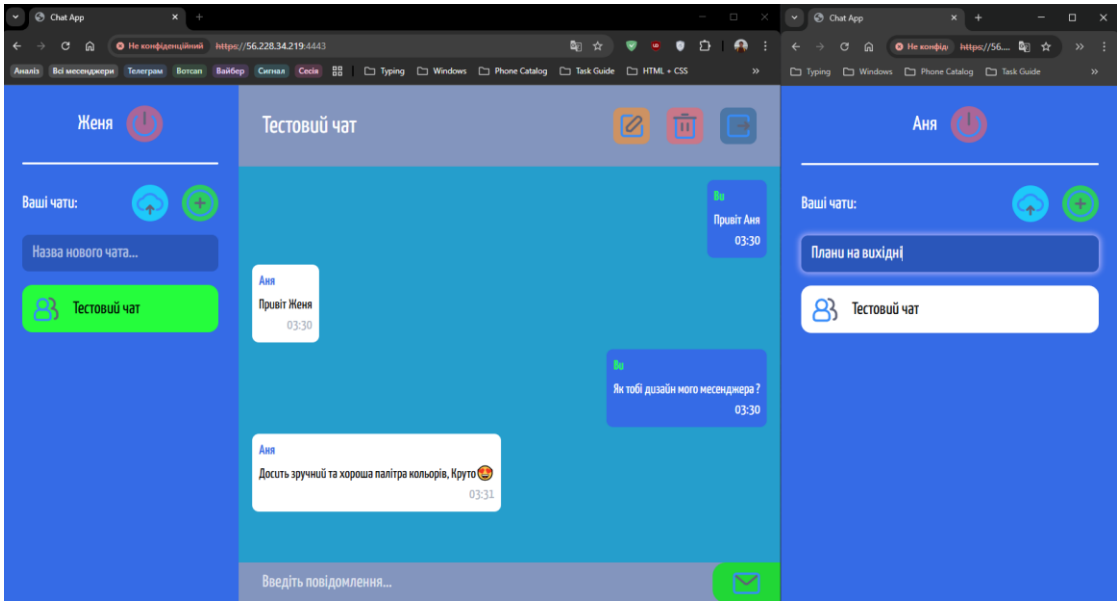


Рис. 3.22. Аня починає створення нового чату

Після підтвердження дії чат **«Плани на вихідні»** успішно створено. Він відображається в інтерфейсі обох користувачів, а також стає доступним для вибору та подальшого використання. Це демонструє багаточатовість системи й коректну логіку відображення змін на всіх підключених клієнтах (рис. 3.23).

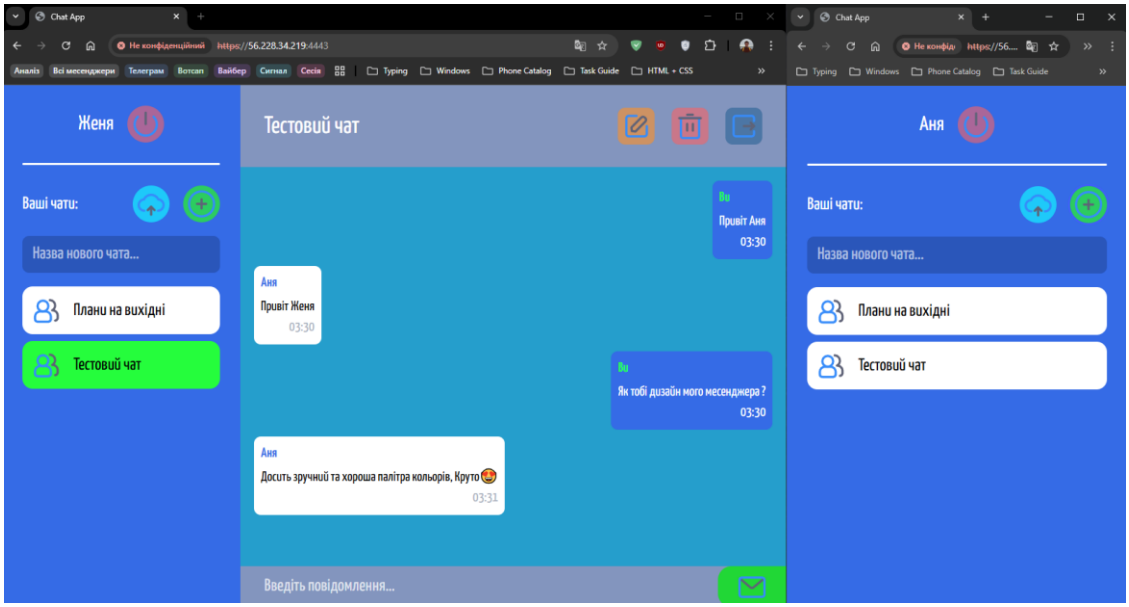


Рис. 3.23. Створений чат «Плани на вихідні» з’являється в обох користувачів

У межах активного діалогу **Аня** як автор кімнати ініціює її перейменування. Вона вводить нову назву — **«Дніпро 💪»**, яка краще відображає тему спілкування. Кнопки керування чатом стають доступними лише для користувача-автора, що реалізовано через логіку перевірки прав у клієнтській частині (рис. 3.24).

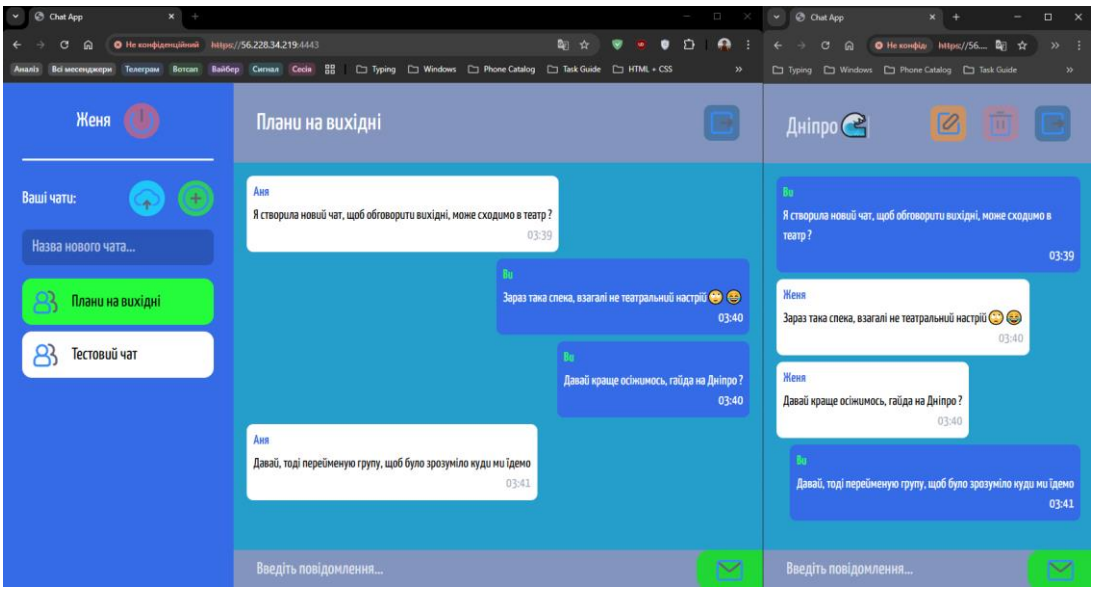


Рис. 3.24. Автор кімнати змінює її назву під час діалогу

Після підтвердження нова назва чату автоматично відображається як у **Жені**, так і в **Ані**. Це підтверджує правильну реалізацію механізму оновлення назви через REST-запит і розсилку змін по WebSocket для синхронізації інтерфейсу (рис. 3.25).

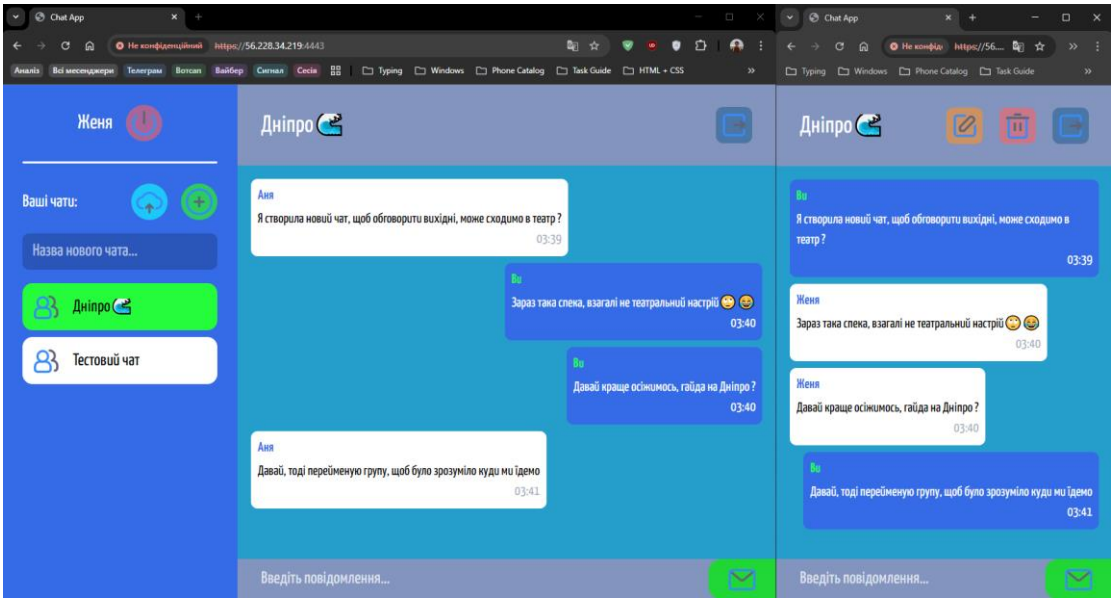


Рис. 3.25. Оновлена назва чату «Дніпро 💪» миттєво з’являється в обох користувачів

Етап 6 – Повторне перейменування кімнати автором

У таблиці rooms на рисунку 3.26 бачимо, що наразі в базі даних зареєстровані дві кімнати: **«Тестовий чат»** (створений Женею) та **«Дніпро»** (створений Анею). Останній запис має мітку оновлення, що свідчить про нещодавнє перейменування кімнати на попередньому етапі.

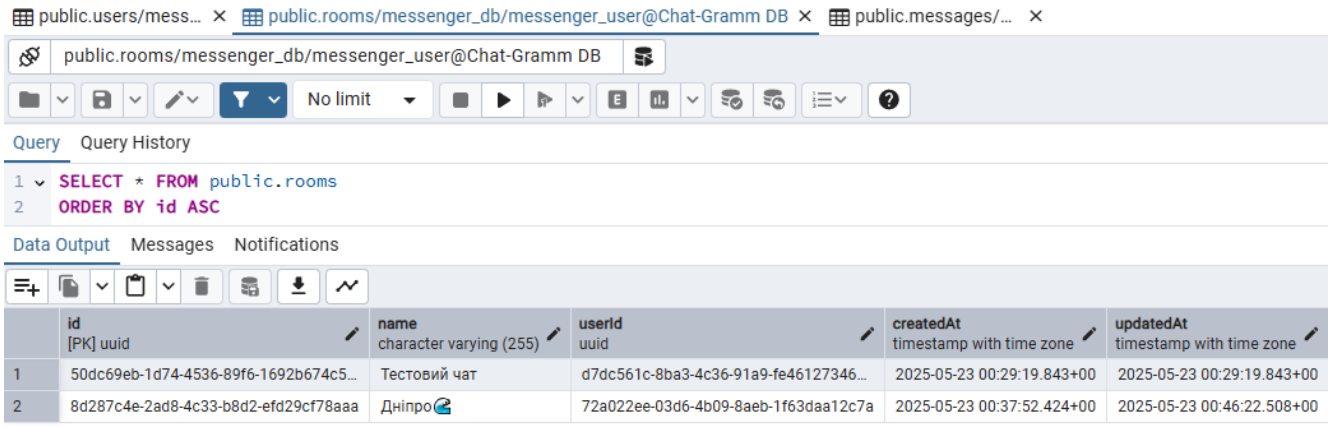


Рис. 3.26. Поточний стан таблиці rooms із записами створених кімнат

У ході подальшого спілкування **Женя** просить змінити назву кімнати ще раз. Оскільки лише **Аня** є автором цієї кімнати, відповідна кнопка редагування доступна лише їй. Вона успішно змінює назву, і зміни одразу відображаються в інтерфейсах обох користувачів (рис. 3.27).

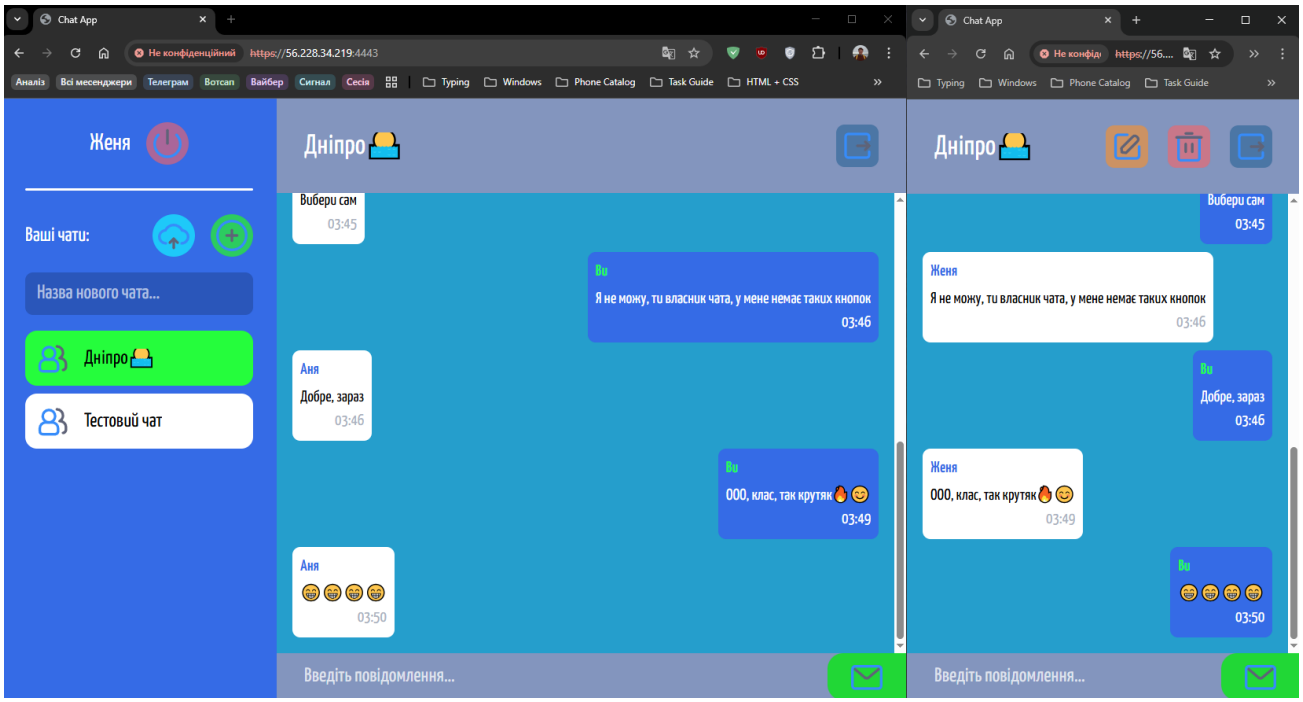


Рис. 3.27. Перейменування кімнати автором за запитом іншого користувача

Після повторного редагування у таблиці rooms оновлюється значення поля name на «**Дніпро**» для відповідної кімнати. Також змінюється updatedAt (рис. 3.28). Це підтверджує, що всі зміни вносяться не лише на рівні інтерфейсу, а й фізично в базі даних.

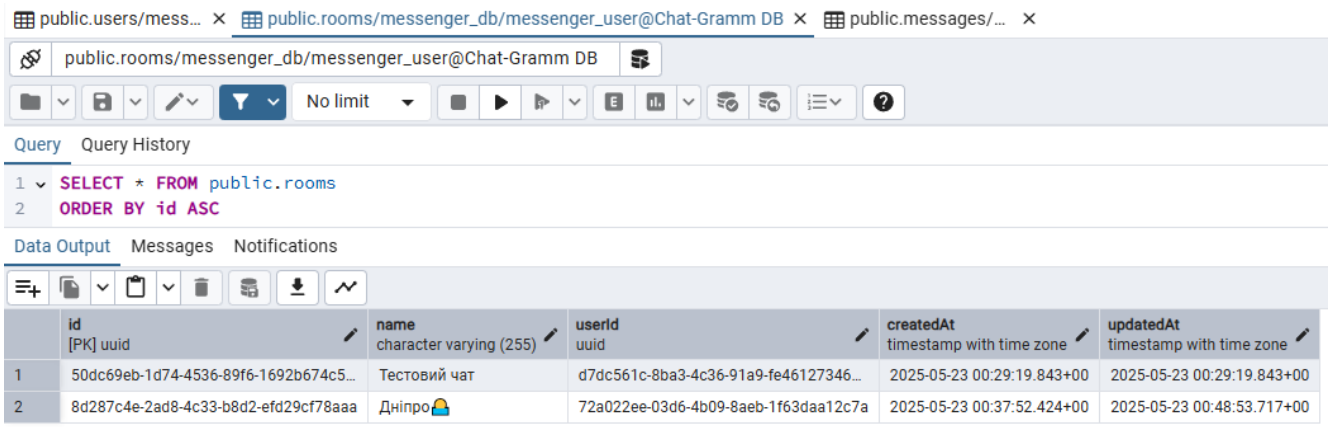


Рис. 3.28. Оновлена назва чату «Дніпро»

Етап 7 – Створення та видалення кімнати

Женя випадково створює новий чат із довільною назвою **«аврекрекп»**. Кімната автоматично додається до списку чатів у обох користувачів та відображається у вигляді активної (рис. 3.29). Обидва користувачі одразу бачать створений чат і мають можливість перейти до обміну повідомленнями.

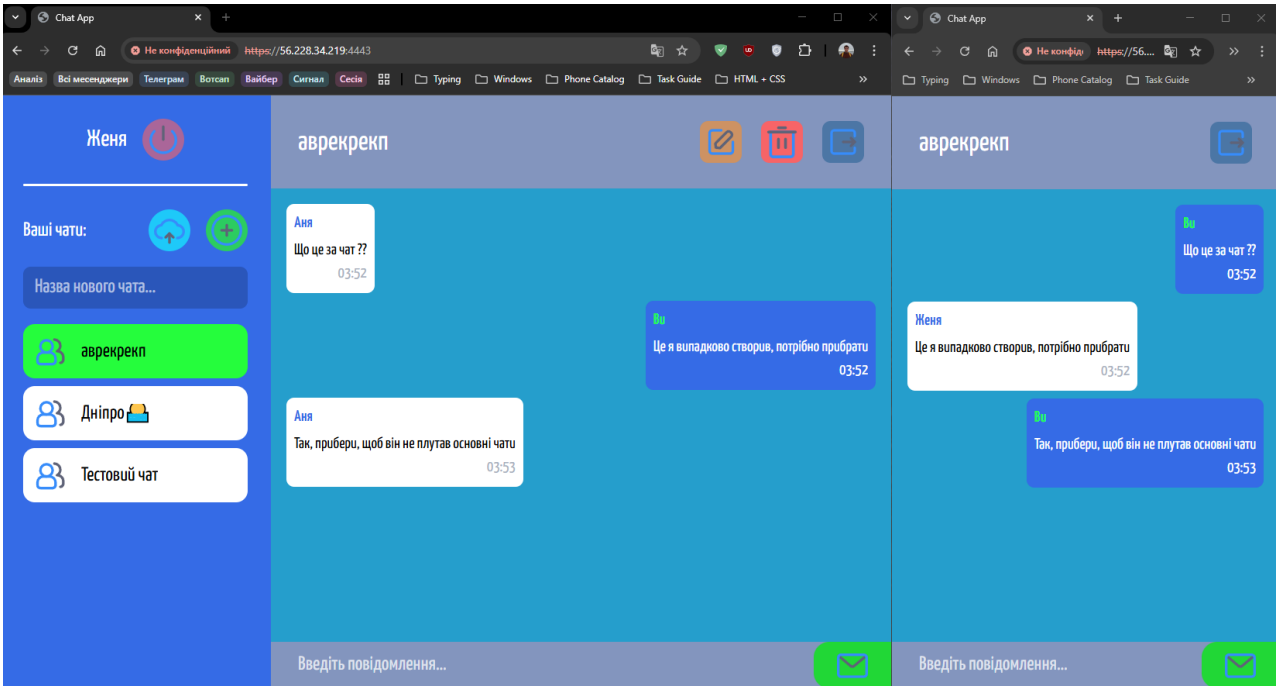


Рис. 3.29. Випадкове створення кімнати «аврекрекп»

У таблиці rooms (рис. 3.30) відображається новий запис із відповідною назвою, ідентифікатором користувача-автора та мітками часу створення й оновлення. Це підтверджує, що нова кімната коректно додана до бази даних.

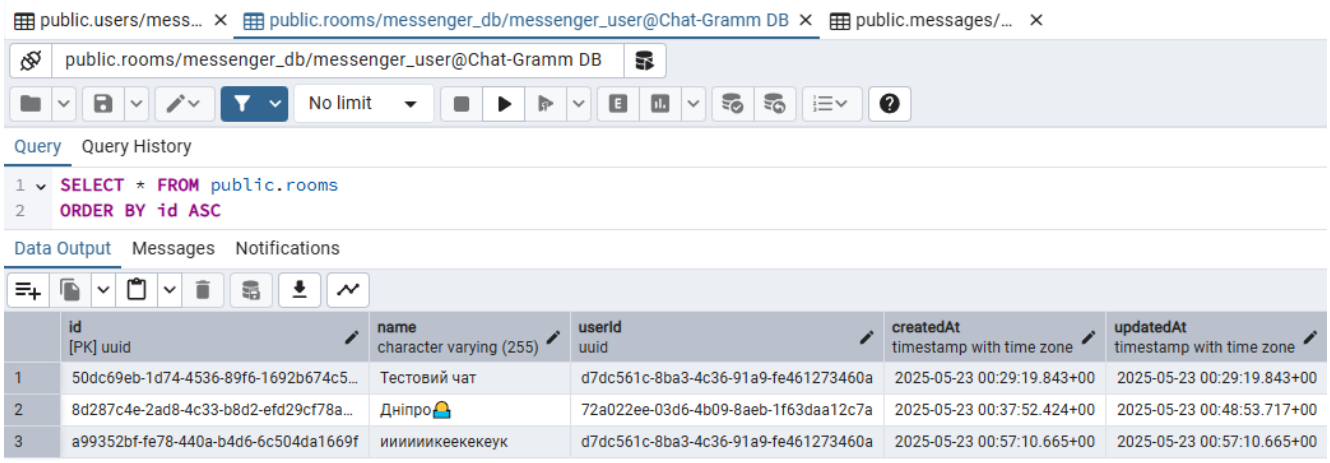


Рис. 3.30. Запис випадково створеної кімнати в базі даних

Оскільки чат був створений випадково, **Женя** як його автор натискає кнопку видалення. Після цього і його, і **Аню** одразу викидає до головного інтерфейсу зі списком чатів. Це свідчить про те, що система не дозволяє користувачам залишатись у видаленій кімнаті та коректно очищає клієнтський стан (рис. 3.31).

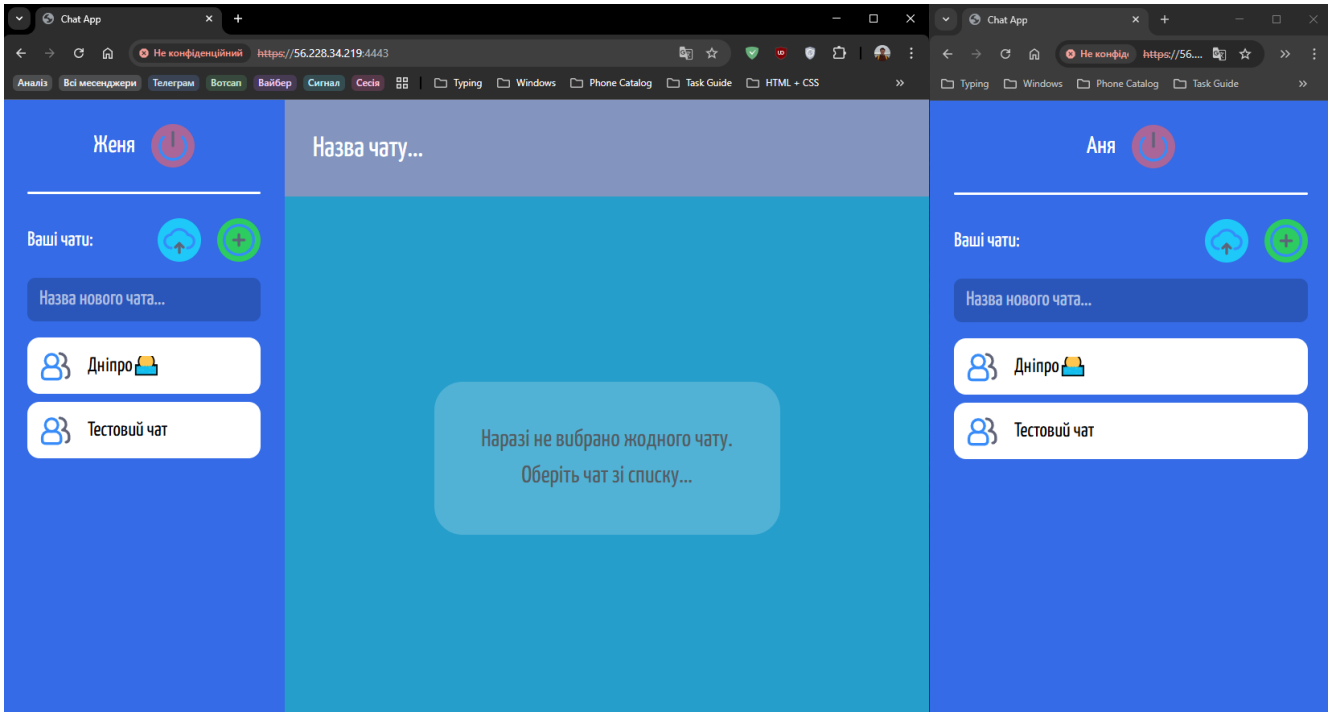


Рис. 3.31. Сторінка користувачів після видалення випадкового чату

Оновлений стан бази даних (рис. 3.32) підтверджує, що кімната з назвою **«аврекрекп»** була повністю видалена. У таблиці залишились лише актуальні кімнати, а записи було фізично очищено з таблиці rooms.

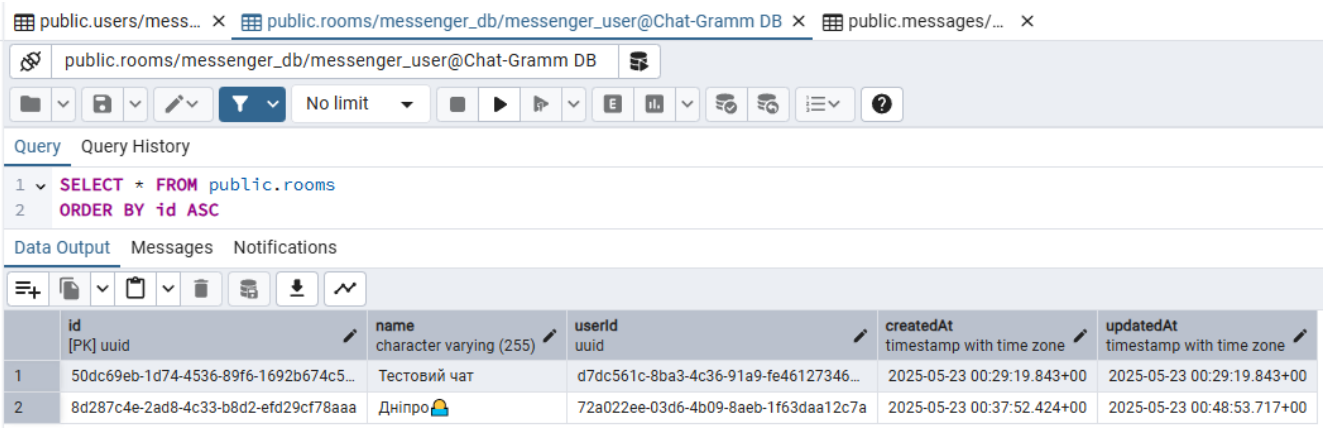


Рис. 3.32. Видалення випадкового чату підтверджено у базі даних

Таблиця messages містить усі повідомлення з усіх чатів (рис. 3.33). Для кожного повідомлення збережено ім’я автора (**userName**), час надсилання (**time**), ідентифікатор кімнати (**roomId**) та дату створення. Головною особливістю є те, що поле **text** містить зашифровані дані (JSON з **iv** і **data**), що забезпечує **наскрізне шифрування** повідомлень і захист вмісту навіть у разі компрометації бази.

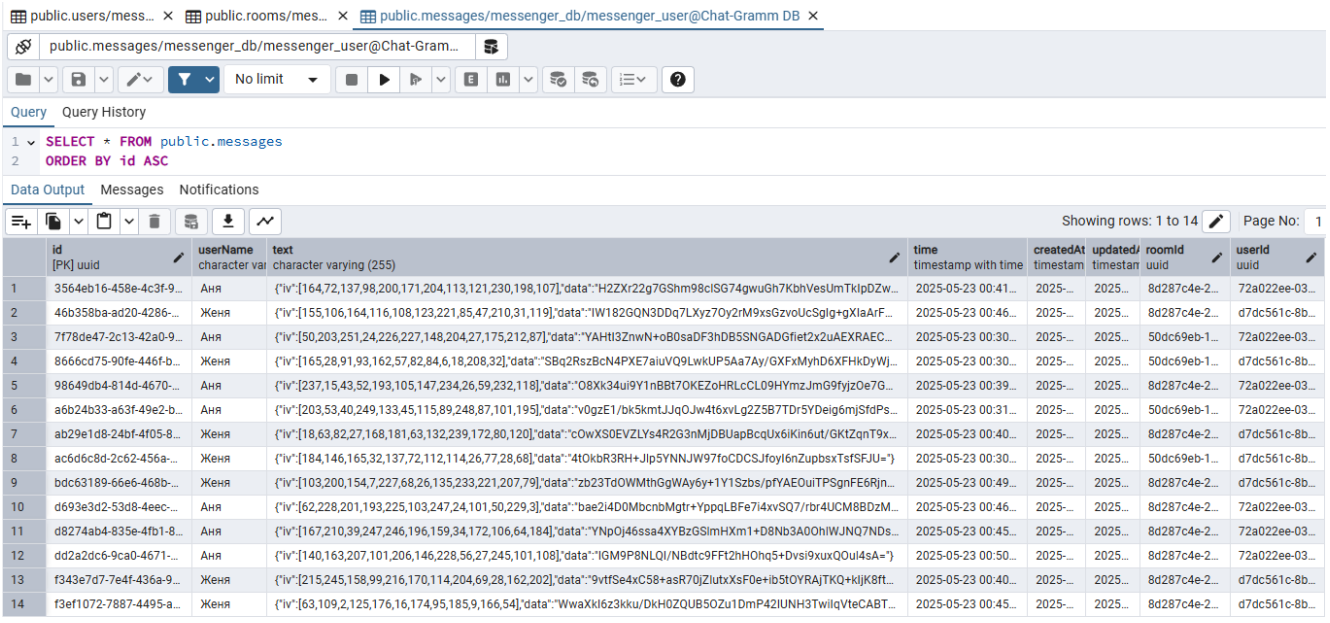


Рис. 3.33. Таблиця повідомлень із зашифрованими текстами в базі даних

3.7 Висновки до розділу 3

У межах третього розділу було реалізовано повноцінну клієнт-серверну архітектуру захищеного Web-мессенджера. Здійснено обґрунтований вибір технологій, які забезпечили стабільність, масштабованість та інформаційну безпеку системи.

Клієнтська частина побудована з використанням **React** і **TypeScript**, що забезпечує компонентну структуру, типізацію та зручність підтримки. Для HTTP-запитів застосовано **Axios**, стилі оформлено за допомогою **SCSS**, а керування станом реалізовано через **useState**.

Серверна частина реалізована на платформі **Node.js** з використанням **Express.js** для REST API та **ws** для обміну повідомленнями. Безпеку даних забезпечує **AES-256-GCM** у поєднанні з **HKDF** згідно з принципами **Double Ratchet**. Збереження даних реалізовано у **PostgreSQL** з використанням ORM **Sequelize**.

Усі модулі запущено в контейнерах **Docker**, що спростило розгортання системи на сервері **AWS EC2** із зарання налаштованими портами.

Проведене тестування продемонструвало коректну роботу основних функцій:

- авторизація користувачів;
- створення, редагування та видалення чатів;
- синхронний обмін повідомленнями через WebSocket;
- відображення змін в інтерфейсі в реальному часі;
- збереження повідомлень у зашифрованому вигляді;
- адаптивність інтерфейсу для різних пристроїв;
- контроль прав доступу до дій у кімнаті.

Таким чином, реалізований Web-мессенджер повністю відповідає вимогам безпеки, продуктивності та зручності, а його модульна структура дозволяє подальший розвиток і масштабування.

4. ЕРГОНОМІЧНЕ ЗАБЕЗПЕЧЕННЯ РОЗРОБКИ WEB-МЕССЕНДЖЕРА

4.1 Вимоги до інтерфейсу з погляду користувача

У процесі розробки Web-застосунку особливу увагу необхідно приділяти саме інтерфейсу користувача, адже саме він формує перше враження про систему, визначає зручність її використання, а в умовах обміну повідомленнями в реальному часі — ще й впливає на швидкість і ефективність взаємодії.

Інтерфейс повинен бути не лише привабливим з точки зору дизайну, а й інтуїтивно зрозумілим, логічним, послідовним. Особливо це важливо для захищених систем, де неправильне розуміння елементів управління може призвести до втрати або компрометації даних. Зручність користування напряму впливає на загальну продуктивність системи — чим менше часу користувач витрачає на освоєння та виконання дій, тим вище ефективність взаємодії з програмним продуктом.

Згідно з принципами ергономіки в IT-системах, інтерфейс повинен відповідати не лише естетичним вимогам, а й таким технічним критеріям, як швидкість виконання типових задач, кількість помилок, що допускаються користувачем, та легкість у самостійному освоєнні функціоналу.

Нижче в таблиці 4.1 наведено основні ергономічні вимоги, які були враховані при проєктуванні інтерфейсу Web-мессенджера:

Таблиця 4.1

Основні вимоги до інтерфейсу користувача Web-мессенджера

№	Вимога	Опис
1	Мінімізація часу на виконання дій	Усі основні дії (вхід до чату, надсилання повідомлення, створення кімнати) — за 1–2 кліки.
2	Низький рівень помилок	Інтерфейс інтуїтивний, не допускає неправильних дій (наприклад, випадкове видалення чату).
3	Простота освоєння (самонавчання)	Елементи мають очікувану поведінку, система не потребує інструкцій або попереднього навчання.

Продовження таблиці 4.1

4	Висока стандартизація	Застосовано відомі шаблони UX — список чатів зліва, поле введення знизу, кнопка справа.
5	Мінімізація введення	Використовується автозаповнення, localeStorage зберігає ім’я користувача.
6	Візуальна привабливість	Єдина кольорова схема, читабельні шрифти, контрастні кнопки, адаптація до розміру екрана.

Реалізація кожної з вищенаведених вимог у межах дипломного проєкту відображена в таблиці 4.2:

Таблиця 4.2

Реалізація вимог до інтерфейсу у Web-мессенджері

№	Реалізована вимога	Реалізація в інтерфейсі Web-мессенджера
1	Мінімізація часу на виконання дій	Кнопки розташовані логічно; створення чату — один клік; надсилання повідомлення — натиснення Enter або кнопки.
2	Низький рівень помилок	Заблоковано надсилання порожніх повідомлень; при редагуванні чату кнопка видалення тимчасово деактивується.
3	Простота освоєння	Застосовано звичну структуру інтерфейсу (список чатів, вікно повідомлень); використано іконки з очікуваною поведінкою.
4	Висока стандартизація	Компоненти побудовані за шаблоном популярних мессенджерів (Telegram/WhatsApp); повторювана структура дій.
5	Мінімізація введення	Ім’я користувача (логін) зберігається у localStorage; автоматичне фокусування поля введення при вході в чат.
6	Візуальна привабливість	Стилізація SCSS; світла кольорова гама; великі кнопки та поля введення; адаптивна верстка під мобільні пристрої.

Таким чином, при розробці інтерфейсу було реалізовано цілу низку ергономічних вимог, які суттєво підвищують зручність використання системи. Завдяки адаптивному дизайну, мінімалізму, відсутності складних меню і підтверджень — Web-мессенджер є простим у використанні як для досвідчених користувачів, так і для новачків.

4.2 Основні підходи до проєктування інтерфейсу

Проєктування користувацького інтерфейсу є не лише питанням естетики, а й критичним фактором ефективності взаємодії користувача із системою. Особливо це стосується застосунків реального часу, таких як Web-мессенджери, де кожна затримка або неправильна дія може вплинути на якість комунікації. Саме тому на етапі проєктування інтерфейсу було обрано низку підходів, що поєднують найкращі практики з UX/UI-дизайну, компонентної архітектури та адаптивної верстки.

Ключовим рішенням стало використання **React** — JavaScript-бібліотеки, що забезпечує побудову інтерфейсу у вигляді окремих повторно використовуваних компонентів. Це дозволяє зберігати логічну структуру застосунку, спрощує супровід і подальший розвиток, а також дозволяє розділити логіку на самостійні модулі: форма входу, список чатів, вікно повідомлень тощо. Кожен компонент має власний стан, який керується за допомогою хуків (***useState***, ***useEffect***), що відповідає підходу React-екосистеми.

Адаптивна верстка та кросплатформність інтерфейсу

Однією з важливих умов сучасного Web-застосунку є повна адаптивність інтерфейсу — здатність автоматично підлаштовуватись під розміри екрану користувача. Це особливо актуально для Web-мессенджера, яким користуються як на комп’ютерах, так і на планшетах та смартфонах.

У межах проєкту реалізовано адаптивну верстку з підтримкою трьох ключових типів пристроїв:

- **Десктопна версія** (ширина екрана від 900 пікселів);
- **Планшетна версія** (від 500 до 900 пікселів);
- **Мобільна версія** (від 320 до 500 пікселів).

Кожна з версій має оптимізоване розташування елементів, зручні зони взаємодії та коректне масштабування тексту й кнопок.

Десктопна версія

У десктопному інтерфейсі реалізовано повноцінне двоколонкове компонування: ліворуч розміщується вертикальне меню з переліком чатів, кнопками створення і вибору, а також панеллю користувача; праворуч — активне вікно діалогу з відображенням повідомлень. Окрему увагу приділено читабельності: великі поля введення, контрастна кольорова схема, достатній відступ між повідомленнями. Екран входу також зберігає центроване розміщення та простоту взаємодії (рис. 4.1).

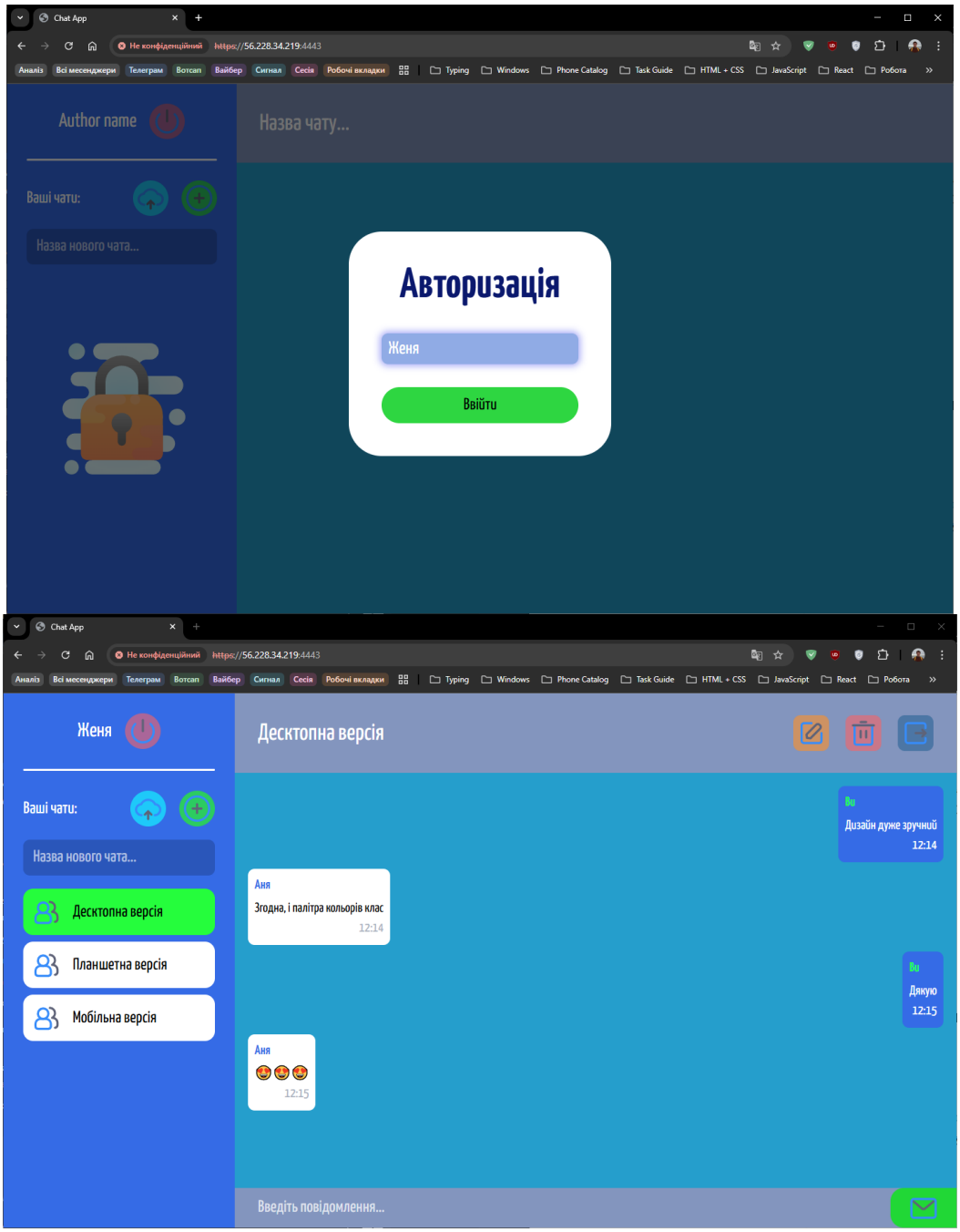


Рис. 4.1. Інтерфейс десктопної версії Web-мессенджера

Планшетна версія

Інтерфейс для пристроїв шириною від 500 до 900 пікселів адаптований під вертикальне або горизонтальне використання планшета. У цьому форматі частина елементів масштабується, а частина змінює своє положення для кращого охоплення користувачем. Вікно входу зберігає компактність, а компонування чатів залишається читабельним. Повідомлення, кнопки, поля — збільшені відповідно до можливостей сенсорної взаємодії. Загальний інтерфейс залишається максимально наближеним до десктопного, проте більш стислим і зручним для пальцевого керування (рис. 4.2 – 4.3).

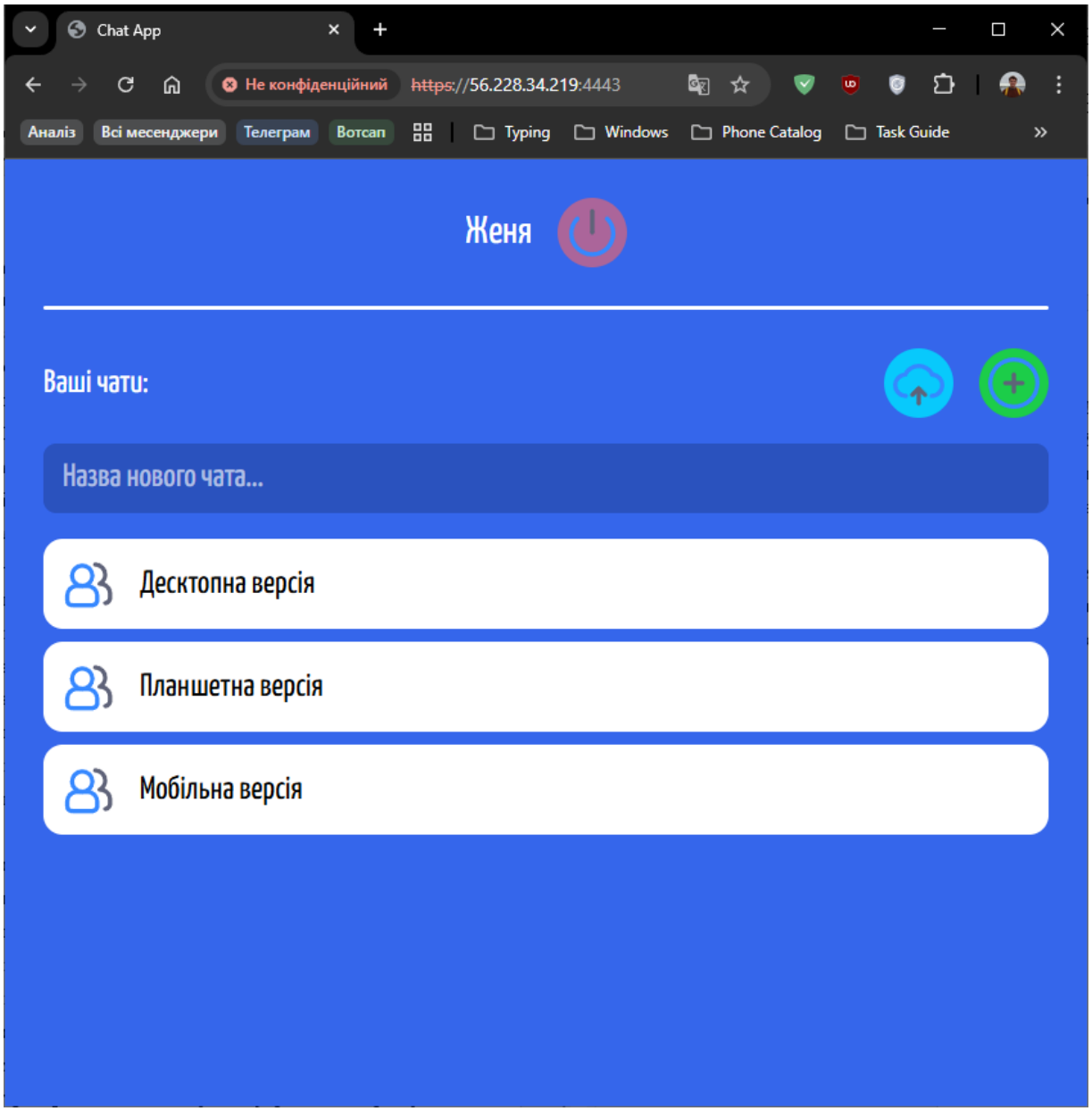


Рис. 4.2. Список чатів планшетної версії Web-мессенджера

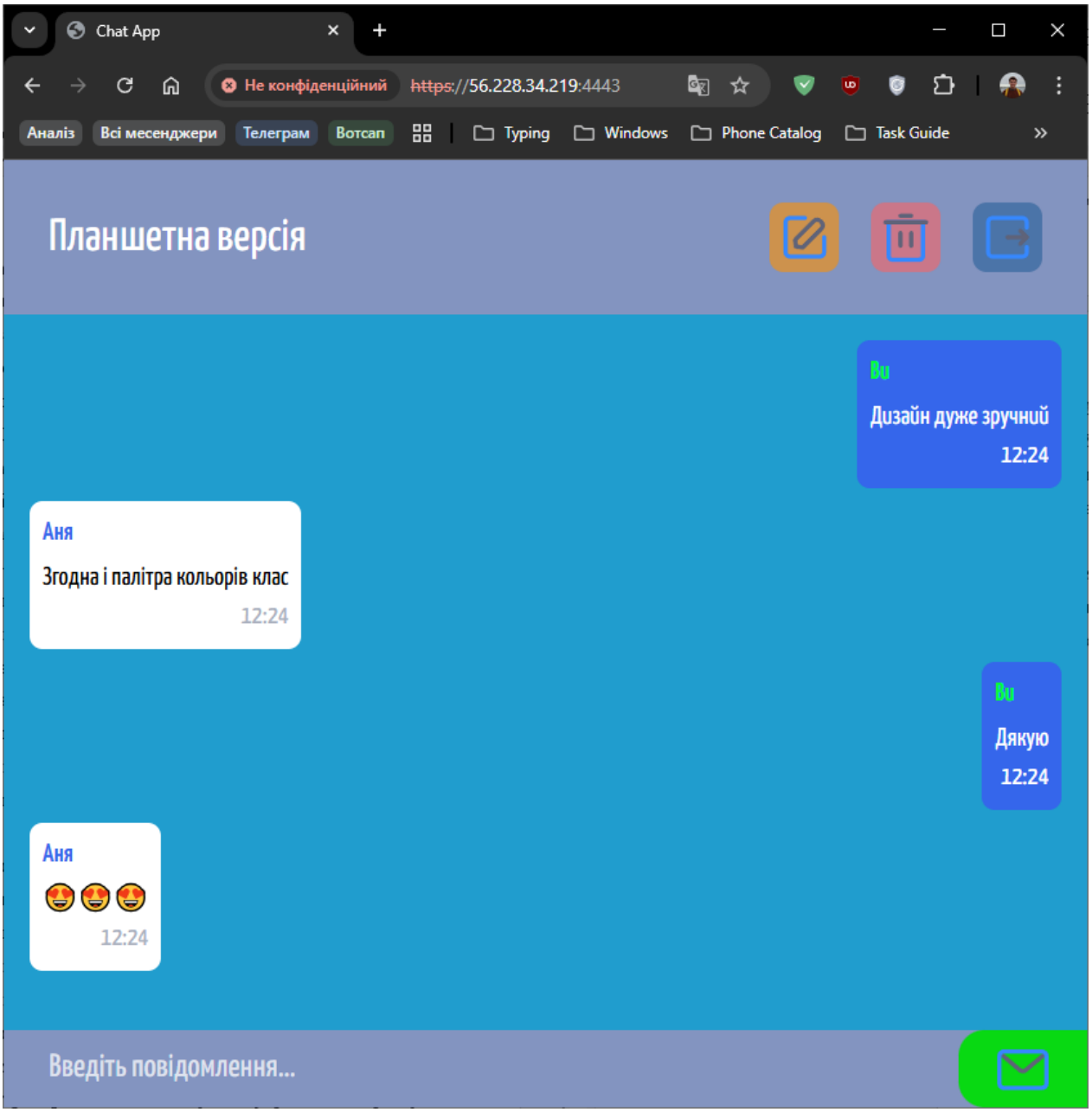


Рис. 4.3. Перехід в конкретний чат на планшетній версії

Мобільна версія

Інтерфейс для пристроїв шириною від 320 до 500 пікселів є найбільш компактним і водночас збереженим за функціональністю. У такому форматі всі елементи масштабується вертикально: список чатів, панель користувача, поле введення. Кожна кнопка має збільшену зону натискання, а текст залишається чітким і добре помітним. Панель повідомлень адаптована під вертикальне гортання, що дозволяє зручно взаємодіяти однією рукою. Навігація спрощена до мінімуму, а інтерфейс залишається чистим, легким і зрозумілим (рис. 4.4).

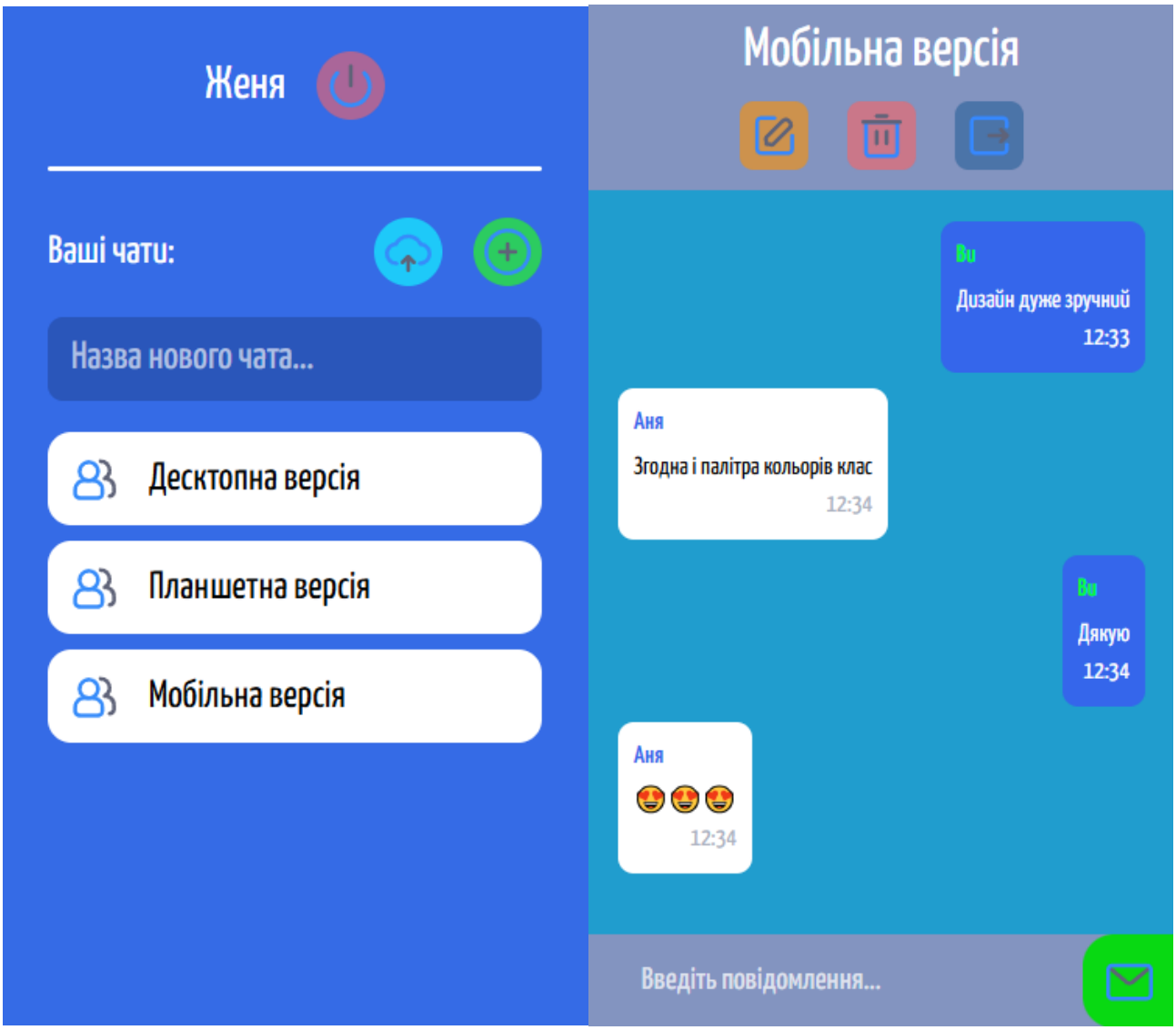


Рис. 4.4. Інтерфейс мобільної версії Web-мессенджера

Особливу увагу приділено **простоті взаємодії**: більшість дій користувач може виконати буквально однією кнопкою або жестом. Наприклад, надсилання повідомлення — це Enter або клік; створення кімнати — простий ввід назви та підтвердження. Усі дії не потребують додаткових переходів, підтверджень чи перезавантажень сторінки.

Також було використано принципи **мінімалістичного дизайну**. Відсутність зайвих елементів дозволяє зосередити увагу на основному: спілкуванні. Кольорова гама нейтральна (світло-синій фон, сірі акценти), кнопки помітні, але не нав’язливі. Розмір тексту адаптовано під користувачів різного віку, поля введення мають достатній простір для зручного введення тексту навіть на сенсорних пристроях.

4.3 Реалізація та особливості інтерфейсу Web-мессенджера

Інтерфейс Web-мессенджера реалізовано у вигляді набору взаємозалежних React-компонентів, кожен із яких відповідає за окрему частину інтерфейсу або функціональність.

Компоненти інтерфейсу згруповані за функціональними блоками:

1. **LoginForm** — форма входу користувача;
2. **ManagementBlock** — блок керування чатами (включає компоненти SecureComponent і RoomItem);
3. **ChatBlock** — основний блок чату (містить ChatModale і MessageItem);
4. **Службові компоненти** — загальні елементи (кнопки, поля введення).

Кожен з цих компонентів має свою логіку, структуру та особливості візуалізації, що детально розглянуто нижче.

LoginForm

Компонент LoginForm відповідає за процес ідентифікації користувача. Інтерфейс реалізовано у вигляді центрованої форми з полем введення логіну, який зберігається у ***localStorage***, що дозволяє зберегти сесію між оновленнями сторінки, та кнопкою підтвердження (рис. 4.5).

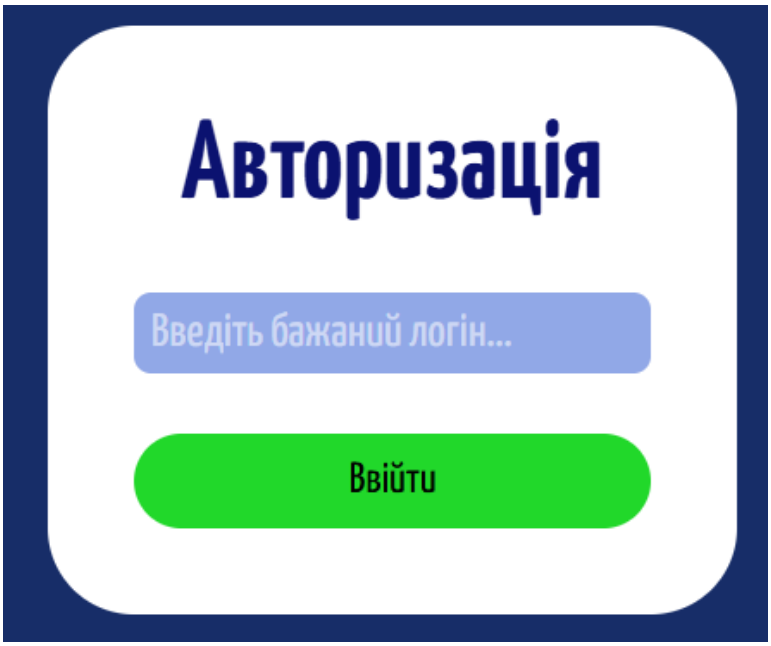


Рис. 4.5. Інтерфейс компонента LoginForm з полем введення логіну

ManagementBlock (панель керування чатами)

Компонент ManagementBlock є лівосторонньою панеллю, яка поєднує кілька важливих елементів інтерфейсу, необхідних для керування чатами та навігації користувача:

- блок з іменем користувача та кнопкою виходу;
- поле для створення нової кімнати;
- інтерактивні кнопки: оновлення чату та створення нового;
- список наявних чатів, що реалізується через дочірній компонент **RoomItem**;
- компонент **SecureComponent**, який виконує функцію доступу: поки користувач не ввійде до системи, цей компонент блокує вивід повідомлень, забезпечуючи тим самим візуальну приватність.

Такий підхід дозволяє зберігати логічну структуру навіть у неавторизованому стані, не показуючи приватну інформацію до входу в систему (рис. 4.6).

Рис. 4.6. Інтерфейс ManagementBlock з полем введення та списком чатів

RoomItem (елемент списку чатів)

Компонент RoomItem реалізує окремий запис списку чатів. Усі кімнати виводяться у вигляді інтерактивних блоків з іконкою та назвою. Активний чат має яскраве підсвічування, інші — нейтральне оформлення (рис. 4.7).

Рис. 4.7. Компоненти RoomItem для доступу до різних кімнат

SecureComponent (захист контенту до авторизації)

Компонент SecureComponent використовується для захисту приватності чату до моменту авторизації. Його завдання — приховати основний вміст (зокрема список повідомлень), доки користувач не введе свій логін. Візуально компонент представлений у вигляді іконки замка, що символізує заблокований доступ. Після входу SecureComponent автоматично зникає, відкриваючи основний функціонал чату (рис. 4.8).

Рис. 4.8. SecureComponent — блокування чату до входу користувача

ChatBlock (основна панель чату)

Компонент ChatBlock відповідає за відображення основного вмісту діалогу між користувачами. Він займає праву частину інтерфейсу і включає в себе такі елементи:

- заголовок активної кімнати;
- кнопки редагування, видалення та виходу з чату;
- поле для відображення повідомлень (**MessageItem**);
- інформативні блоки у випадку порожніх чатів (**ChatModale**);
- поле введення нового повідомлення та кнопка надсилання.

Цей блок динамічно оновлюється залежно від контексту: якщо жодного чату не вибрано або у чаті немає повідомлень, користувач отримує відповідні підказки. Усі елементи адаптовані до взаємодії як мишею, так і сенсором (рис. 4.9).

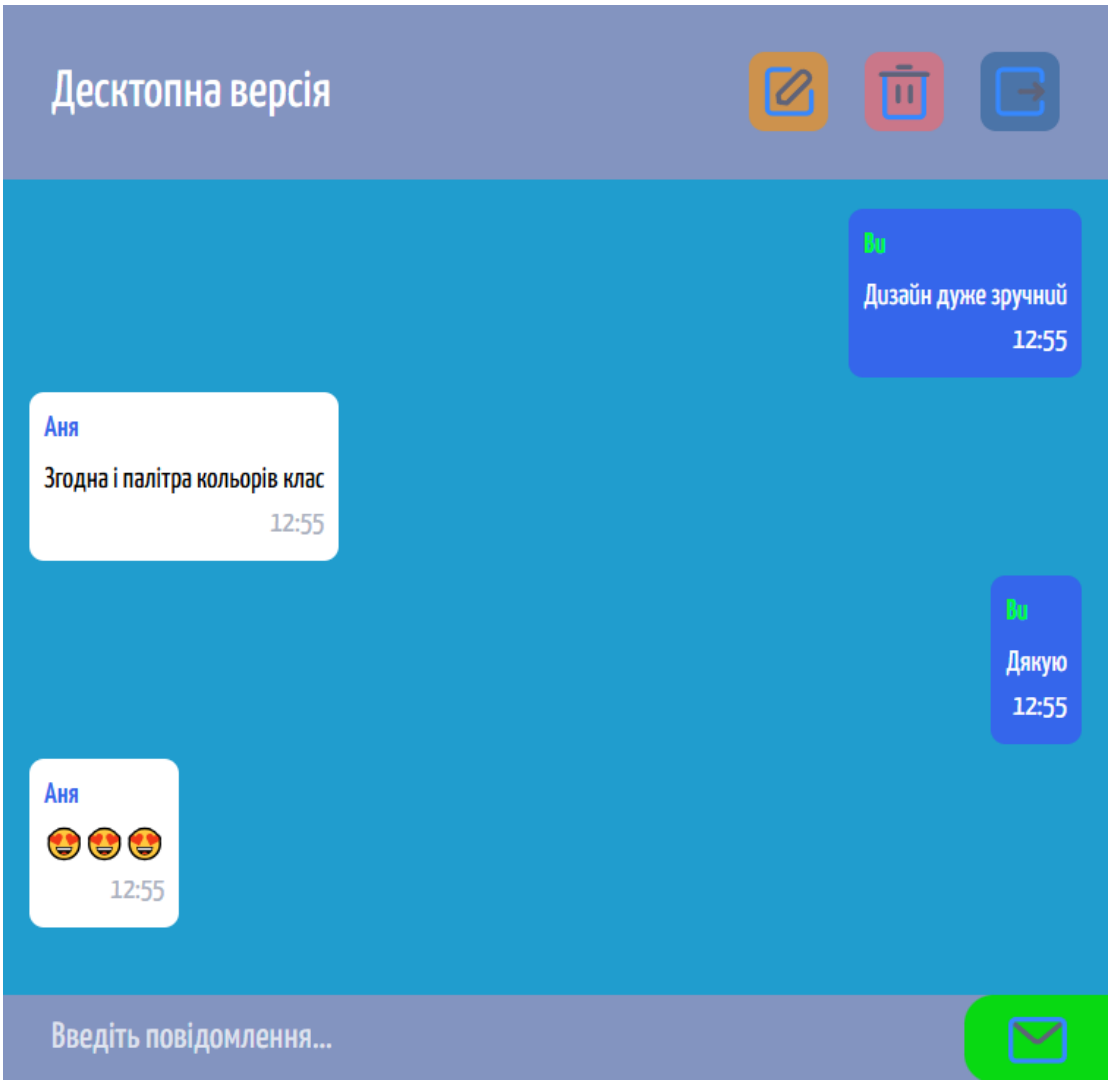


Рис. 4.9. Загальний вигляд ChatBlock з активною кімнатою та повідомленнями

MessageItem (повідомлення у чаті)

Компонент MessageItem — це структурний елемент, що відповідає за відображення одного повідомлення в чаті. Усі повідомлення мають:

- відображення імені автора;
- текст повідомлення;
- мітку часу;
- кольорову стилізацію залежно від автора (власні повідомлення виділено синім, отримані — білим фоном).

Такий дизайн дозволяє швидко ідентифікувати учасників розмови та підтримує візуальний поділ повідомлень (рис. 4.10).

Рис. 4.10. Компоненти MessageItem — приклад вхідних і вихідних повідомлень

ChatModale (інформаційні повідомлення в чаті)

Компонент ChatModale використовується для інформування користувача у випадках, коли:

- жодна кімната не обрана;
- у вибраній кімнаті немає жодного повідомлення.

Замість порожнього екрану, користувачу виводиться стилізований блок із текстовим повідомленням, яке пояснює поточний стан. Це знижує когнітивне навантаження і робить інтерфейс дружнім (рис. 4.11).

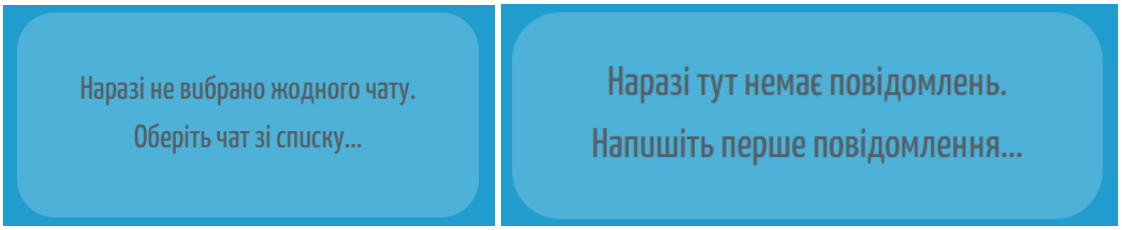


Рис. 4.11. Компоненти ChatModale — НЕ обрано чат/обрано чат

Input (текстові поля)

Компонент Input є базовим елементом введення текстової інформації в інтерфейсі Web-мессенджера. Його основна функція — забезпечити зручний, зрозумілий і візуально привабливий механізм введення тексту в різних частинах застосунку. Усі поля мають округлі краї, приємне підсвічування по контуру та застосовують ефекти фокусування.

В інтерфейсі використано чотири типи полів введення, кожне з яких має свою логіку:

1. **Вхід користувача.** Відображається при старті застосунку (LoginForm). Призначене для введення бажаного логіну (рис. 4.12):

Рис. 4.12. Поле для введення логіну при вході в систему

2. **Створення нового чату.** Розташовується в блоці ManagementBlock. Дає змогу ввести назву нової кімнати (рис. 4.13):

Рис. 4.13. Поле створення нового чату

3. **Введення повідомлення.** Розташоване в нижній частині ChatBlock. Призначене для набору тексту під час спілкування (рис. 4.14):

Рис. 4.14. Поле для введення повідомлення

4. **Редагування назви чату.** Активується при зміні імені кімнати. Візуально повторює структуру стандартних полів (рис. 4.15):

Рис. 4.15. Поле для редагування назви чату

Button (кнопки)

Компонент Button реалізовано як універсальний механізм взаємодії користувача з Web-мессенджером. Кожна кнопка має характерний стиль: округлі краї, яскраву кольорову палітру, зручну зону натискання. При наведенні або дотику (на мобільних та планшетах) кнопка змінює відтінок, що створює ефект зворотного зв’язку.

У проєкті реалізовано кілька функціональних типів кнопок:

1. **Кнопка входу (LoginForm)** — використовується для підтвердження введеного логіну (рис. 4.16 – 4.17).

Рис. 4.16. Кнопка входу (звичайний стан)

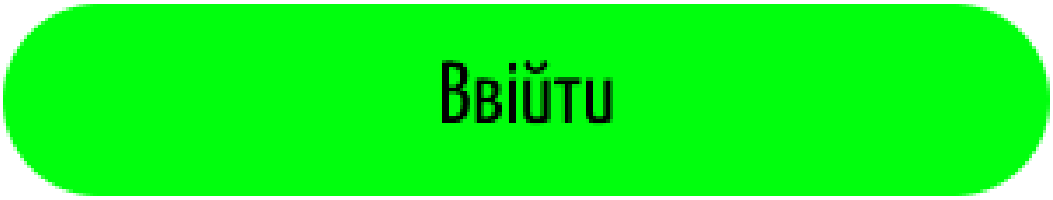


Рис. 4.17. Кнопка входу (стан при наведенні або натисканні)

2. **Кнопка виходу (ManagementBlock)** — дозволяє завершити сесію (рис. 4.18 – 4.19).

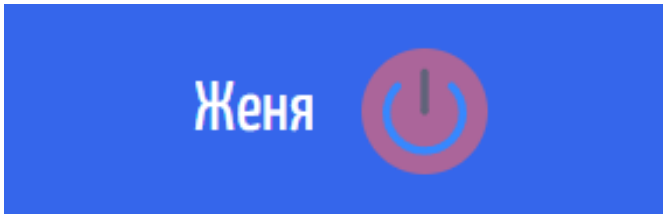


Рис. 4.18. Кнопка виходу — звичайний стан

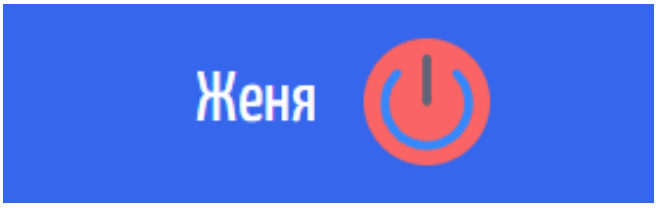


Рис. 4.19. Кнопка виходу — стан при наведенні

3. **Кнопки створення та оновлення списку чатів** — розміщені поруч із полем чату, мають піктограми (рис. 4.20 – 4.22).

Рис. 4.20. Кнопки чатів — стандартний вигляд

Рис. 4.21. Кнопка оновлення списку чатів — стан при наведенні курсору

Рис. 4.22. Кнопка додавання нового чату — стан при наведенні курсору

4. **Кнопки керування чатом (редагування, видалення, вихід).**

   У правій частині заголовка активного чату знаходяться три кнопки, кожна з яких має власну іконку та колір (рис. 4.23):

Рис. 4.23. Кнопки керування чатами — звичайний стан

Редагування назви чату — помаранчева кнопка з іконкою олівця (рис. 4.24):

Рис. 4.24. Наведення на кнопку редагування назви чату

Видалення чату — червона кнопка з іконкою смітника (рис. 4.25):

Рис. 4.25. Наведення на кнопку видалення чату

Вихід із чату — синя кнопка зі стрілкою «назовні» (рис. 4.26):

Рис. 4.26. Наведення на кнопку виходу з чату

Усі ці кнопки мають змінений стан при наведенні курсору або дотику: фон стає яскравішим, що дає користувачу чіткий сигнал про активність елемента.

5. **Кнопка надсилання повідомлення**

Розміщується в правому нижньому куті блоку введення повідомлення. Стилістично — це зелена кнопка з піктограмою конверта. При наведенні або дотику в мобільному режимі межа контуру підсвічується, що сигналізує про можливість дії. (рис. 4.27 – 4.28).

Рис. 4.27. Кнопка надсилання повідомлення — звичайний стан

Рис. 4.28. Кнопка надсилання повідомлення — стан при наведенні

Інтерфейс Web-мессенджера реалізовано як набір модульних компонентів. Така архітектура забезпечує гнучкість, розширюваність і зручність у підтримці коду. Усі компоненти згруповані за функціональним призначенням, що дозволяє зберігати логічну і візуальну цілісність системи (табл. 4.3).

Таблиця 4.3

Функціональні групи інтерфейсних компонентів Web-мессенджера

№	Назва групи	Складові компоненти	Опис функціональності
1	LoginForm	—	Форма авторизації, зберігає логін у localStorage, проста та адаптивна
2	ManagementBlock	SecureComponent, RoomItem	Панель керування чатами, приховування вмісту до входу, список і створення кімнат
3	ChatBlock	ChatModale, MessageItem	Основний блок діалогу з повідомленнями, інформативні модальні стани, поле введення
4	Службові компоненти	Input, Button	Універсальні поля введення і кнопки з єдиним стилем, адаптовані для всіх пристроїв

4.4 Проблеми, що виникали при розробці, та шляхи їх вирішення

Під час реалізації Web-мессенджера виникали низка проблем, які торкались різних аспектів системи — від синхронізації даних до поведінки елементів інтерфейсу та розгортання на сервері. Всі труднощі умовно можна поділити на три основні групи:

1. **Технічні проблеми**
2. **Візуальні (UI/UX) проблеми**
3. **Проблеми при докеризації та хостингу**

1. Технічні проблеми

На початковому етапі розробки Web-мессенджера було виявлено критичні технічні труднощі, пов’язані з **відсутністю повноцінної синхронізації подій між клієнтами** в реальному часі. Це спричиняло розбіжності у відображенні стану чатів на різних пристроях і користувачах, що призводило до втрати цілісності роботи системи.

Основні прояви проблеми:

- **Створення кімнати:** новостворена кімната відображалась лише у списку кімнат користувача, який її створив. Інші учасники не бачили її до моменту натискання кнопки оновлення списку кімнат.
- **При редагуванні назви кімнати** — зміни відображались лише у автора. Інші учасники продовжували бачити стару назву до моменту ручного перезаходу в кімнату.
- **При видаленні кімнати** — кімната зникала лише в того користувача, який ініціював видалення. Для інших вона залишалась у списку, хоча на сервері вже не існувала.
- **Вихід користувача з системи:** після виходу користувача (автора кімнати), інші користувачі, що залишались у створених ним кімнатах, продовжували бачити їх у своєму списку та перебувати в них. Проте фактично ці кімнати вже не існували на сервері.

Синхронізація операції створення кімнати

Щоб усунути проблему, коли створена кімната відображалась лише у користувача-автора, було реалізовано наступний механізм розсилки WebSocket.

Суть рішення:

1. **Клієнт, який створює кімнату**, після успішного POST-запиту на створення кімнати передає WebSocket-повідомлення типу ***RoomOperation.Create*** (рис. 4.29):

```
case Types.RequestType.RoomCreate: {
  const { data: newRoom } = await axios.post(`${API_URL}room/createRoom`, body);
  setNewRoomName('');
  socket?.send(JSON.stringify({
    type: Types.RoomOperation.Create,
    room: newRoom
  }));
  break;
}
```

Рис. 4.29. Відправка WebSocket-повідомлення про створення кімнати на клієнті

2. **Сервер приймає це повідомлення**, і надсилає всім іншим клієнтам повідомлення типу ***roomCreated*** з даними про нову кімнату (рис. 4.30):

```
// Створення кімнати
if (data.type === 'roomCreated') {
  wss.clients.forEach((client) => {
    client.send(
      JSON.stringify({ type: 'roomCreated', room: data.room }),
    );
  });

  return;
}
```

Рис. 4.30. Обробка на сервері: розсилка повідомлення roomCreated всім клієнтам

3. **Кожен клієнт, що отримав повідомлення**, виконує оновлення свого локального списку кімнат, додаючи нову (рис. 4.31):

```
case Types.RoomOperation.Create:
  setRooms(currentRooms => [data.room, ...currentRooms]);
  break;
```

Рис. 4.31. Додавання кімнати в локальний стан на клієнті (setRooms)

Синхронізація операції редагування кімнати

Щоб уникнути ситуації, коли нова назва кімнати відображалась лише у користувача-автора, було реалізовано наступний механізм розсилки WebSocket.

Суть рішення:

1. **Після редагування кімнати**, клієнт надсилає повідомлення типу RoomOperation.Rename до WebSocket-сервера, передаючи оновлену інформацію про кімнату (рис. 4.32):

```
case Types.RequestType.RoomRename: {
  const { data: updatedRoom } = await axios.post(`${API_URL}room/editRoom`, body);
  setRoomIsChanging(false);
  socket?.send(JSON.stringify({
    type: Types.RoomOperation.Rename,
    room: updatedRoom
  }));
  break;
}
```

Рис. 4.32. Відправка повідомлення про редагування кімнати WebSocket-клієнтом

2. **Сервер отримує повідомлення roomRenamed**, і транслює його всім активним клієнтам — включно з тими, хто не ініціював зміну (рис. 4.33).

```
// Редагування кімнати
if (data.type === 'roomRenamed') {
  wss.clients.forEach((client) => {
    client.send(
      JSON.stringify({ type: 'roomRenamed', room: data.room }),
    );
  });

  return;
}
```

Рис. 4.33. Обробка на сервері: сповіщення клієнтів про зміну назви кімнати

3. **Клієнти на боці отримання**, виконують оновлення своєї локальної колекції кімнат, замінюючи стару назву новою, якщо room.id збігається (рис. 4.34).

```
case Types.RoomOperation.Rename: {
  setRooms(currentRooms =>
    currentRooms.map(room =>
      room.id === data.room.id ? data.room : room
    )
  );
  if (data.room.userId === selectedRoom?.userId) {
    setSelectedRoom(data.room);
  }
  break;
}
```

Рис. 4.34. Оновлення назви кімнати в списку на клієнті

Синхронізація операції видалення кімнати

Щоб уникнути ситуації, коли кімната видалялась лише локально для одного користувача, було реалізовано наступний механізм розсилки WebSocket.

Суть рішення:

1. **Користувач, що ініціює видалення**, виконує POST-запит до сервера і після цього надсилає WebSocket-повідомлення з типом RoomOperation.Delete та ID кімнати (рис. 4.35):

```
case Types.RequestType.RoomDelete: {
  await axios.post(`${API_URL}room/deleteRoom`, body);
  sessionStorage.removeItem(`sharedRoomKey-${body?.id}`);
  socket?.send(JSON.stringify({
    type: Types.RoomOperation.Delete,
    deletedRoomId: body?.id
  }));
  break;
}
```

Рис. 4.35. Відправка повідомлення WebSocket при видаленні кімнати

2. **Сервер отримує повідомлення**, і через WebSocket інформує всіх клієнтів про необхідність видалення кімнати, передаючи її ідентифікатор (рис. 4.36):

```
// Видалення кімнати
if (data.type === 'roomDeleted') {
  wss.clients.forEach((client) => {
    client.send(
      JSON.stringify({
        type: 'roomDeleted',
        deletedRoomId: data.deletedRoomId,
      }),
    );
  });
  delete rooms[data.deletedRoomId];

  return;
}
```

Рис. 4.36. Обробка на сервері: розсилка команди roomDeleted всім клієнтам

3. **Клієнти, які отримали це повідомлення**, видаляють відповідну кімнату зі свого локального стану, очищують вибраний чат та сесійні ключі (рис. 4.37):

```
case Types.RoomOperation.Delete: {
  setRooms(currentRooms => currentRooms.filter(({ id }) => id !== data.deletedRoomId));
  if (selectedRoom?.id === data.deletedRoomId) {
    setSelectedRoom(null);
    setMessages([]);
    sessionStorage.removeItem(`sharedRoomKey-${data.deletedRoomId}`);
  }
  break;
}
```

Рис. 4.37. Видалення кімнати на клієнті

Синхронізація операції виходу користувача з системи

Щоб уникнути ситуації, після виходу користувача його кімнати не зникали у списках інших учасників, було реалізовано наступний механізм розсилки WebSocket.

Суть рішення:

1. **Після виходу користувача** клієнт надсилає WebSocket-повідомлення типу RoomOperation.Logout, де передається userId користувача (рис. 4.38):

```
case Types.RequestType.Logout: {
    await axios.post(`${API_URL}user/logout`, body);
    onLogout();
    localStorage.removeItem('author');
    sessionStorage.clear();
    socket?.send(JSON.stringify({
        type: Types.RoomOperation.Logout,
        userId: body?.user?.id
    }));
    break;
}
```

Рис. 4.38. Відправка повідомлення про вихід користувача через WebSocket

2. **Сервер отримує повідомлення**, і через WebSocket транслює його всім клієнтам з командою userLogout. Після цього на сервері видаляються всі кімнати, створені вийшовшим користувачем (рис. 4.39):

```
// Видалення користувача
if (data.type === 'userLogout') {
    wss.clients.forEach((client) => {
        client.send(
            JSON.stringify({
                type: 'userLogout',
                userId: data.userId,
            }),
        );
    });
    roomService.getAllRooms().forEach((room) => delete rooms[room.id])

    return;
}
```

Рис. 4.39. Серверна обробка виходу користувача: сповіщення та очищення кімнат

3. **Клієнтська логіка реагує на повідомлення** та виконує очищення відповідних кімнат на клієнті (рис. 4.40):

```
case Types.RoomOperation.Logout: {
    setRooms(currentRooms => currentRooms.filter(({ userId }) => userId !== data.userId));
    if (selectedRoom?.userId === data.userId) {
        setSelectedRoom(null);
        setMessages([]);
        sessionStorage.clear();
    }
    break;
}
```

Рис. 4.40. Реакція клієнта: видалення кімнат, що більше не існують

2. Візуальні проблеми

У процесі розробки інтерфейсу виникла **UI-проблема**, яка могла призвести до втрати даних: при редагуванні кімнати не блокувалась кнопка її видалення. Це створювало **ризик випадкового видалення кімнати**, особливо при швидкій роботі користувача або на мобільних пристроях.

Опис проблеми:

Під час зміни назви кімнати (***EditName***) кнопка видалення (***DeleteRoom***) залишалась активною, що могло призвести до ненавмисної втрати кімнати у разі неправильного натискання.

Рішення:

Для уникнення цього ризику реалізовано **блокування кнопки видалення**, якщо активний режим редагування. Було використано логічну змінну ***roomIsChanging***, яка встановлюється у ***true*** під час редагування, і ***false*** — після завершення.

1. Встановлення умови disabled на кнопці видалення — вона неактивна під час редагування (рис. 4.41):

```
<>
  <Button type={Types.Button.Rename} onClick={handleEditName} />
  <Button type={Types.Button.Delete} disabled={roomIsChanging} onClick={() => handleDeleteRoom(selectedRoom?.id)} />
</>
```

Рис. 4.41. Умова блокування кнопки Delete при редагуванні кімнати

2. Розширення логіки кнопки у загальному компоненті Button — підтримка передачі пропса disabled, який впливає на події та стилі (рис. 4.42):

```
return (
  <button
    type={type === Types.Button.Refresh ? 'button' : 'submit'}
    className={`btn__container btn__container--${type}`}
    onClick={onClick}
    disabled={disabled}
  >
    <div className={`btn btn__${type}`} />
  </button>
);
```

Рис. 4.42. Компонент кнопки — підтримка пропса disabled у логіці кнопки

3. Стилізація :disabled у SCSS — зменшення прозорості, блокування подій, зміна курсору (рис. 4.43):

```scss
&--delete {
  background-color: $red-color-off;

  &:disabled {
    opacity: 0.5;
    pointer-events: none;
    cursor: not-allowed;
  }

  &:hover {
    background-color: $red-color-on;
  }
}
```

Рис. 4.43. SCSS стилізація стану disabled: курсор, прозорість, блок подій

4. Візуальний ефект блокування у реальному інтерфейсі — кнопка стає напівпрозорою та не реагує на натискання (рис. 4.44 – 4.45):

Рис. 4.44. Інтерфейс: кнопка видалення у звичайному стані

Рис. 4.45. Інтерфейс: кнопка видалення у стані disabled під час редагування

3. Проблеми при докеризації та хостингу

Фінальним етапом розробки Web-мессенджера було його **повноцінне розгортання у Docker-середовищі** з налаштуванням усіх ключових сервісів: frontend, backend, база даних PostgreSQL та проксі-сервер NGINX. Проте саме на цьому етапі виникла низка технічних складнощів.

Основні проблеми:

- **Взаємна несумісність між контейнерами.** Сервіси не бачили одне одного через неправильно вказані мережі та залежності.
- **Відсутність шифрування повідомлень.** Через те, що фронтенд працював через незахищений протокол HTTP, бібліотека crypto.subtle (використовувана для Double Ratchet-шифрування) не могла ініціалізуватись. Це призводило до **повної недоступності захищеного обміну повідомленнями**.

Рішення:

Було розроблено **уніфікований docker-compose.yaml (рис. 4.47)**, в якому:

- об'єднано всі сервіси в одну мережу chat-gramm;
- створено окремий контейнер NGINX, який виконує функцію проксі і маршрутизує трафік до frontend та backend;
- відкрито порт **4443 для HTTPS-з'єднання**, на якому й працює мессенджер.
- підключено **самозгенерований SSL-сертифікат**, що дозволяє frontend запускатись на https:// та повноцінно використовувати криптографічні API браузера (рис. 4.46):

```
volumes:
  - ./nginx.conf:/etc/nginx/nginx.conf:ro
  - /etc/ssl/certs/server.crt:/etc/ssl/certs/server.crt:ro
  - /etc/ssl/private/server.key:/etc/ssl/private/server.key:ro
```

Рис. 4.46. Підключення SSL-сертифікату в NGINX

Завдяки такій структурі вдалося **усунути всі проблеми взаємодії між контейнерами**, а також **гарантувати захищене з'єднання**, що було обов’язковою умовою для безпечного обміну повідомленнями.

Рис. 4.47. Конфігурація docker-compose.yaml з підключенням SSL-сертифікату

4.5 Висновки до розділу 4

У цьому розділі було детально розглянуто ергономічне забезпечення Web-мессенджера, що включає аналіз вимог до інтерфейсу, обґрунтування архітектурних підходів, реалізацію UI-компонентів, вирішення проблем, а також адаптивність системи до різних типів пристроїв.

Ключові підсумки наведено в таблиці 4.4.

Таблиця 4.4

Основні підсумки реалізації ергономічного забезпечення Web-мессенджера

№	Аспект	Короткий зміст
1	Вимоги до інтерфейсу	Мінімізація дій користувача, інтуїтивність, стандарти UX, візуальна простота
2	Архітектурний підхід	Модульна структура на React, компоненти поділено за функціональними блоками
3	Основні реалізовані блоки	LoginForm, ManagementBlock, ChatBlock, службові компоненти Input, Button
4	Адаптивність інтерфейсу	Підтримка трьох основних форматів — десктопного, планшетного, мобільного
5	Вирішення технічних проблем	Повна синхронізація подій через WebSocket: створення, редагування, видалення, вихід
6	Вирішення UI-проблем	Блокування кнопки видалення під час редагування кімнати
7	Докеризація та безпека	Налаштована взаємодія всіх сервісів через docker-compose.yaml, запуск із захистом HTTPS

Таким чином, реалізований Web-мессенджер відповідає сучасним вимогам до безпечних, адаптивних і зручних для користувача інформаційних систем. Його інтерфейс повністю відповідає поставленим ергономічним цілям, а структура реалізації дозволяє легко масштабувати або вдосконалювати продукт у майбутньому.

ЗАГАЛЬНІ ВИСНОВКИ

При виконанні дипмлоної роботи на тему **«Програмна розробка захищеного Web-мессенджера для обміну повідомленнями в реальному часі»** було виконано повноцінний цикл аналізу, проєктування, реалізації та тестування інформаційної системи, що відповідає сучасним вимогам безпеки, зручності та масштабованості.

У першому розділі було здійснено ґрунтовний **аналіз сучасних засобів безпечного обміну повідомленнями**, досліджено особливості Web-мессенджерів, функціонал, архітектурні та криптографічні особливості найпопулярніших рішень: **Telegram**, **WhatsApp**, **Signal**, **Viber** та **Session**. Проведено **порівняльний аналіз**, що дозволив виявити сильні та слабкі сторони кожного з них. Основним висновком стало те, що жоден з розглянутих мессенджерів не поєднує одночасно: високу безпеку, конфіденційність, повноцінну Web-версію та прозору архітектуру.

У другому розділі було виконано **архітектурне та логічне проєктування Web-мессенджера**, включаючи побудову інформаційної моделі, потоків даних, концептуальної, логічної й фізичної моделей бази даних. Особливу увагу приділено **забезпеченню конфіденційності** через наскрізне шифрування повідомлень за алгоритмом **Double Ratchet**. Обрано клієнт-серверну архітектуру з використанням WebSocket для обміну повідомленнями в реальному часі. Усі дані зберігаються на сервері виключно в зашифрованому вигляді, а ключі — лише на пристроях користувачів. Це гарантує **високу стійкість до загроз інформаційної безпеки**.

У третьому розділі реалізовано **програмну частину Web-мессенджера**. Клієнтську частину створено на базі **React + TypeScript**, серверну — з використанням **Node.js**, **Express.js**, **PostgreSQL** і **WebSocket**. Застосунок розгорнуто в **Docker-середовищі** з проксінгом через **NGINX** та підтримкою **HTTPS**. У розділі детально описано реалізацію шифрування/дешифрування повідомлень, структуру інтерфейсу та результат **тестування на реальних сценаріях**. Продемонстровано синхронізацію даних, адаптивність дизайну та

працездатність системи на різних пристроях. Підтверджено ефективність реалізованого механізму захисту та зручність користування застосунком.

У четвертому розділі розглянуто **ергономіку інтерфейсу**, описано ключові принципи зручності користування, адаптивності та візуальної привабливості. Кожен елемент інтерфейсу реалізовано як **окремий React-компонент** із власною логікою та стилем. Інтерфейс протестовано у трьох адаптивних режимах: для ПК, планшетів та смартфонів. У межах розділу також **детально описано проблеми, що виникали при розробці**, поділені на три групи: технічні, візуальні та пов'язані з докеризацією. Для кожної проблеми надано рішення з прикладами коду, реалізацією WebSocket-синхронізації та покращенням логіки інтерфейсу.

Унікальність розробленого Web-мессенджера полягає у:

- **Повній клієнтській реалізації наскрізного шифрування** із динамічним оновленням ключів (Double Ratchet).
- **Мінімалістичному, але адаптивному інтерфейсі**, який зберігає функціональність на будь-якому пристрої.
- **Наявності повноцінної Web-версії** без потреби встановлення — з високою безпекою, що рідко зустрічається серед подібних рішень.
- **Модульній та масштабованій архітектурі**, що забезпечує зручність супроводу, розгортання, тестування та розвитку системи.

Розроблений Web-мессенджер є **авторським, унікальним рішенням**, яке не має прямих аналогів з аналогічною архітектурою, функціоналом і рівнем безпеки у форматі web-only із груповими чатами. Проєкт повністю відповідає вимогам до кваліфікаційної роботи та демонструє високий рівень технічної реалізації та аналітичної підготовки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цікаві факти про цифрові технології у світі та в Україні [Електронний ресурс] // MediaMaker: [сайт]. – URL: https://mediamaker.me/najczikavishe-zi-zvitu-digital-2025-pro-vzayemodiyu-z-czyfrovymy-tehnologiyamy-16257/ (дата звернення: 15.01.2025).
2. DataReportal. Digital 2025: Ukraine [Електронний ресурс] // DataReportal: [сайт]. – URL: https://datareportal.com/reports/digital-2025-ukraine (дата звернення: 15.01.2025).
3. Messaging App Statistics (2024) [Електронний ресурс] // Exploding Topics: [сайт]. – URL: https://explodingtopics.com/blog/messaging-apps-stats (дата звернення: 15.01.2025).
4. Як змінилося користування мобільними застосунками за 5 років: соцмережі та мессенджери [Електронний ресурс] // CASES: [сайт]. – URL: https://cases.media/en/article/yak-zminilosya-koristuvannya-mobilnimi-zastosunkami-za-5-rokiv-socmerezhi-ta-mesendzheri (дата звернення: 15.01.2025).
5. Telegram FAQ [Електронний ресурс] // Telegram: [сайт]. – URL: https://telegram.org/faq (дата звернення: 16.01.2025).
6. Telegram Web: усе, що потрібно знати про сервіс [Електронний ресурс] // Gagadget: [сайт]. – URL: https://gagadget.com/uk/45599-telegram-web-onlain-vse-shcho-potribno-znati-pro-servis/ (дата звернення: 16.01.2025).
7. WhatsApp [Електронний ресурс] // WhatsApp: [сайт]. – URL: https://www.whatsapp.com/stayconnected (дата звернення: 17.01.2025).
8. Як працює WhatsApp Web [Електронний ресурс] // WhatsApp FAQ: [сайт]. – URL: https://faq.whatsapp.com/668538004658079?helpref=hc_fnav#whatsapp-web (дата звернення: 17.01.2025).
9. Що таке WhatsApp Web? [Електронний ресурс] // ProIT: [сайт]. – URL: https://proit.ua/shcho-takie-whatsapp-web/ (дата звернення: 17.01.2025).

10. WhatsApp Web в Україні: переваги та недоліки [Електронний ресурс] // Chanty Blog: [сайт]. – URL: https://www.chanty.com/blog/uk/whatsapp-web-uk/ (дата звернення: 17.01.2025).
11. Офіційний сайт Viber [Електронний ресурс] // Viber: [сайт]. – URL: https://www.viber.com/ua/ (дата звернення: 18.01.2025).
12. Viber – статистика користування [Електронний ресурс] // SimilarWeb: [сайт]. – URL: https://www.similarweb.com/app/google/com.viber.voip/#overview (дата звернення: 18.01.2025).
13. Signal – офіційний сайт [Електронний ресурс] // Signal.org: [сайт]. – URL: https://signal.org/uk/#signal (дата звернення: 19.01.2025).
14. Signal Messenger: конфіденційність і безпека [Електронний ресурс] // Cyberset: [сайт]. – URL: https://cyberset.com.ua/privacy-vs-anonymization/signal-messenger/ (дата звернення: 19.01.2025).
15. Session – офіційний сайт [Електронний ресурс] // GetSession: [сайт]. – URL: https://getsession.org/ (дата звернення: 20.01.2025).
16. Топ мессенджерів: рейтинг 2024 [Електронний ресурс] // Vlada-Rykova.com: [сайт]. – URL: https://vlada-rykova.com/top-messendzherov/#Telegram (дата звернення: 20.01.2025).
17. Найкращі мессенджери із наскрізним шифруванням [Електронний ресурс] // Acer Blog: [сайт]. – URL: https://blog.acer.com/ua/discussion/2537/naykraschi-mesendzheri-iz-naskriznim-shifruvannyam-privatnih-povidomlen-u-2025-roci (дата звернення: 20.01.2025).
18. Найзахищеніші мессенджери 2025 року [Електронний ресурс] // KR-Labs: [сайт]. – URL: https://kr-labs.com.ua/blog/top-secure-and-privacy-messaging-apps/ (дата звернення: 20.01.2025).
19. GeeksforGeeks. Understanding Encapsulation, Inheritance, Polymorphism and Abstraction in OOPs [Електронний ресурс] // GeeksforGeeks: [сайт]. – URL: https://www.geeksforgeeks.org/understanding-encapsulation-inheritance-polymorphism-abstraction-in-oops/ (дата звернення: 10.04.2025).

20. GeeksforGeeks. Agile Software Process and its Principles [Електронний ресурс] // GeeksforGeeks: [сайт]. – URL: https://www.geeksforgeeks.org/agile-software-process-and-its-principles/ (дата звернення: 10.04.2025).
21. Signal >> Specifications >> The Double Ratchet Algorithm [Електронний ресурс] // Signal: [сайт]. – URL: https://signal.org/docs/specifications/doubleratchet/ (дата звернення: 15.04.2025).
22. Wikipedia contributors. Double Ratchet Algorithm [Електронний ресурс] // Wikipedia: [сайт]. – URL: https://en.wikipedia.org/wiki/Double_Ratchet_Algorithm (дата звернення: 15.45.2025).
23. Marlinspike, M., & Perrin, T. The Double Ratchet Algorithm – Technical Specification [Електронний ресурс] // Signal: [сайт]. – URL: https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf (дата звернення: 15.04.2025).
24. React – офіційна документація [Електронний ресурс] // React.dev: [сайт]. – URL: https://react.dev (дата звернення: 01.05.2025).
25. TypeScript Handbook – офіційний сайт [Електронний ресурс] // TypeScriptlang.org: [сайт]. – URL: https://www.typescriptlang.org/docs (дата звернення: 01.05.2025).
26. Axios – HTTP клієнт для браузера і Node.js [Електронний ресурс] // GitHub.com: [сайт]. – URL: https://github.com/axios/axios (дата звернення: 01.05.2025).
27. Sass (SCSS) – офіційна документація [Електронний ресурс] // Sass-lang.com: [сайт]. – URL: https://sass-lang.com/documentation (дата звернення: 01.05.2025).
28. Node.js – офіційна документація [Електронний ресурс] // Nodejs.org: [сайт]. – URL: https://nodejs.org/en/docs (дата звернення: 05.05.2025).
29. Express – фреймворк для Node.js [Електронний ресурс] // Expressjs.com: [сайт]. – URL: https://expressjs.com/ (дата звернення: 05.05.2025).
30. ws – WebSocket бібліотека для Node.js [Електронний ресурс] // Npmjs.com: [сайт]. – URL: https://www.npmjs.com/package/ws (дата звернення: 05.05.2025).
31. Sequelize – ORM для Node.js [Електронний ресурс] // Sequelize.org: [сайт]. – URL: https://sequelize.org/docs/v6/ (дата звернення: 05.05.2025).

32. dotenv – модуль керування змінними середовища [Електронний ресурс] // Npmjs.com: [сайт]. – URL: https://www.npmjs.com/package/dotenv (дата звернення: 05.05.2025).
33. Docker – офіційна документація [Електронний ресурс] // Docker.com: [сайт]. – URL: https://docs.docker.com (дата звернення: 07.05.2025).
34. Nginx – офіційна документація [Електронний ресурс] // Nginx.org: [сайт]. – URL: https://nginx.org/en/docs (дата звернення: 07.05.2025).
35. AWS EC2 – хмарні обчислення на запит [Електронний ресурс] // Amazon Web Services: [сайт]. – URL: https://aws.amazon.com/ru/ec2/ (дата звернення: 07.05.2025).
36. Git та GitHub – система контролю версій [Електронний ресурс] // Git-scm.com: [сайт]. – URL: https://github.com/ (дата звернення: 08.05.2025).
37. PostgreSQL – офіційна документація [Електронний ресурс] // PostgreSQL.org: [сайт]. – URL: https://www.postgresql.org/docs/ (дата звернення: 08.05.2025).
38. Visual Studio Code – офіційний редактор коду [Електронний ресурс] // Code.visualstudio.com: [сайт]. – URL: https://code.visualstudio.com/ (дата звернення: 01.05.2025).

ДОДАТКИ

ДОДАТОК А – App.tsx

```
import './App.scss';
import * as Types from './types/types';
import { useEffect, useState } from 'react';
import { ManagementBlock } from
'./components/ManagementBlock/ManagementBlock';
import { ChatBlock } from './components/ChatBlock/ChatBlock';
import { LoginForm } from './components/LoginForm/LoginForm';
import { API_WEBSOCKET } from './constants/constants';
import { createSocket } from './service/createSocket';
import { createRequest } from './service/createRequest';

export const App = () => {
  const [author, setAuthor] = useState<Types.User | null>(null);
  const [rooms, setRooms] = useState<Types.Room[]>([]);
  const [messages, setMessages] = useState<Types.Message[]>([]);
  const [selectedRoom, setSelectedRoom] = useState<Types.Room |
null>(null);
  const [newNameOfRoom, setNewNameOfRoom] = useState('');
  const [roomIsChanging, setRoomIsChanging] = useState(false);
  const [newMessageText, setNewMessageText] = useState('');
  const [refresh, setRefresh] = useState(false);
  const [socket, setSocket] = useState<WebSocket | null>(null);

  useEffect(() => {
    const createdSocket = createSocket({
      type: Types.SocketType.Room,
      url: API_WEBSOCKET,
      createData: { selectedRoom },
      actions: { setRooms, setSelectedRoom, setMessages },
    });

    setSocket(createdSocket);

    return () => {
      createdSocket.close();
```

```
      setSocket(null);
    };
  }, [selectedRoom]);


  useEffect(() => {
    const authorFromStorage = localStorage.getItem('author');
    setAuthor(authorFromStorage ? JSON.parse(authorFromStorage) as
Types.User : null);
  }, []);

  useEffect(() => {
    createRequest({
      type: Types.RequestType.FethRooms,
      actions: { setRooms },
      errorText: Types.RequestError.FethRooms,
    });
  }, [author, refresh]);

  const login = (user: Types.User) => {
    setAuthor(user);
    setSelectedRoom(null);
  };

  const logout = () => {
    setAuthor(null);
    setSelectedRoom(null);
    setMessages([]);
  };

  const handleSelectRoom = (room: Types.Room) => {
    createRequest({
      type: Types.RequestType.RoomSelect,
      actions: { setSelectedRoom, setRefresh },
      body: { id: room.id, prevRoomId: selectedRoom?.id },
      errorText: Types.RequestError.RoomSelect,
    });
  };
```

```
  return (
    <main className='aplication'>
      {!author && <LoginForm onLogin={login} />}
      <ManagementBlock
        author={author}
        selectedRoom={selectedRoom}
        rooms={rooms}
        setRooms={setRooms}
        setSelectedRoom={handleSelectRoom}
        setRoomIsChanging={setRoomIsChanging}
        setNewNameOfRoom={setNewNameOfRoom}
        setNewMessageText={setNewMessageText}
        setRefresh={setRefresh}
        onLogout={logout}
        roomSocket={socket}
      />

      <ChatBlock
        author={author}
        messages={messages}
        setMessages={setMessages}
        selectedRoom={selectedRoom}
        roomIsChanging={roomIsChanging}
        newNameOfRoom={newNameOfRoom}
        newMessageText={newMessageText}
        setNewNameOfRoom={setNewNameOfRoom}
        setNewMessageText={setNewMessageText}
        setSelectedRoom={setSelectedRoom}
        setRoomIsChanging={setRoomIsChanging}
        setRooms={setRooms}
        setRefresh={setRefresh}
        roomSocket={socket}
      />
    </main>
  );
}
```

ДОДАТОК Б – LoginForm.tsx

```
/* eslint-disable react/prop-types */
import './LoginForm.scss';
import { useState } from 'react';
import { Button } from '../Button/Button';
import { Input } from '../Input/Input';
import { createRequest } from '../../service/createRequest';
import * as Types from '../../types/types';
interface Props {
  onLogin: (user: Types.User) => void,
}
export const LoginForm: React.FC<Props> = ({ onLogin }) => {
  const [username, setUsername] = useState('');
  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    if (!username.trim()) {
      setUsername('')
      return;
    }
    createRequest({
      type: Types.RequestType.Login,
      actions: { onLogin },
      body: { name: username.trim() },
      errorText: Types.RequestError.Login,
    });
  };

  return (
    <section className='login'>
      <form className='login__form' onSubmit={handleSubmit}>
        <h1 className='login__form--title'>Авторизація</h1>
        <Input
          type={Types.Input.Login}
          value={username}
          placeholder={Types.PlaceHolder.Login}
          onChange={setUsername}
        />
        <Button type={Types.Button.Login} />
      </form>
    </section>
  )
};
```

ДОДАТОК В – ManagementBlock.tsx

```
/* eslint-disable react/prop-types */
import './ManagementBlock.scss';
import { useCallback, useState } from 'react';
import { SecureComponent } from '../SecureComponent/SecureComponent';
import { Button } from '../Button/Button';
import { Input } from '../Input/Input';
import { RoomItem } from '../RoomItem/RoomItem';
import * as Types from '../../types/types';
import { createRequest } from '../../service/createRequest';

interface Props {
  author: Types.User | null,
  rooms: Types.Room[],
  selectedRoom: Types.Room | null,
  roomSocket: WebSocket | null,
  setRooms: (value: React.SetStateAction<Types.Room[]>) => void,
  setSelectedRoom: (value: Types.Room) => void,
  setRoomIsChanging: React.Dispatch<React.SetStateAction<boolean>>,
  setNewNameOfRoom: (value: string) => void,
  setNewMessageText: (value: string) => void,
  setRefresh: React.Dispatch<React.SetStateAction<boolean>>,
  onLogout: () => void,
}

export const ManagementBlock: React.FC<Props> = ({
  author,
  rooms,
  selectedRoom,
  roomSocket,
  setSelectedRoom,
  setRoomIsChanging,
  setNewNameOfRoom,
  setNewMessageText,
  setRefresh,
  onLogout,
}) => {
  const [newRoomName, setNewRoomName] = useState('');

  const handleLogout = () => {
    setNewRoomName('');
```

```
    createRequest({
      type: Types.RequestType.Logout,
      actions: { onLogout },
      body: { user: author },
      errorText: Types.RequestError.Logout,
      socket: roomSocket,
    });
  };

  const handleCreateRoom = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    if (!newRoomName.trim()) {
      setNewRoomName('');

      return;
    }

    createRequest({
      type: Types.RequestType.RoomCreate,
      actions: { setNewRoomName, },
      body: { userId: author?.id, name: newRoomName.trim() },
      errorText: Types.RequestError.RoomCreate,
      socket: roomSocket,
    });
  };

  const handleSelectRoom = (roomToSelect: Types.Room) => {
    setSelectedRoom(roomToSelect);
    setNewNameOfRoom(roomToSelect.name);
    setRoomIsChanging(false);
    setNewMessageText('');
  };

  const refreshRooms = useCallback(() => setRefresh(prevRefresh => !prevRefresh),
[setRefresh]);

  return (
    <section className='magagement'>
      <article className="magagement__user-block">
        <h1 className="magagement__user-block--title">{author ? author.name :
'Author name'}</h1>
```

```
        <Button type={Types.Button.Logout} onCLick={handleLogout} />
      </article>

      <article className="magagement__line" />

      <form className="magagement__creator" onSubmit={handleCreateRoom}>
        <div className="magagement__creator--top">
          <h1 className="magagement__creator--title">Ваші чати:</h1>

          <div className="magagement__creator--buttons">
            <Button type={Types.Button.Refresh} onCLick={refreshRooms} />
            <Button type={Types.Button.Create} />
          </div>
        </div>

        <Input
          type={Types.Input.RoomCreate}
          value={newRoomName}
          placeholder={Types.PlaceHolder.RoomCreate}
          onChange={setNewRoomName}
        />
      </form>

      <ul className='magagement__rooms'>
        <SecureComponent author={author} />

        {rooms.map(room => (
          <RoomItem
            key={room.id}
            room={room}
            selectedRoom={selectedRoom}
            author={author}
            onClick={handleSelectRoom}
          />
        ))}
      </ul>
    </section>
  );
}
```

ДОДАТОК Г– ChatBlock.tsx

```
/* eslint-disable react/prop-types */
import './ChatBlock.scss';
import classNames from 'classnames';
import { useEffect, useRef, useState } from 'react';
import { ChatModale } from '../ChatModale/ChatModale';
import { Button } from '../Button/Button';
import { Input } from '../Input/Input';
import { MessageItem } from '../MessageItem/MessageItem';
import { API_WEBSOCKET } from '../../constants/constants';
import { createSocket } from '../../service/createSocket';
import { createRequest } from '../../service/createRequest';
import * as Types from '../../types/types';
import * as encryptionService from '../../service/encryptionService';

interface Props {
  author: Types.User | null,
  messages: Types.Message[],
  setMessages: React.Dispatch<React.SetStateAction<Types.Message[]>>,
  selectedRoom: Types.Room | null,
  roomIsChanging: boolean,
  newNameOfRoom: string,
  newMessageText: string,
  setNewNameOfRoom: (value: string) => void
  setNewMessageText: (value: string) => void
  setSelectedRoom: (value: Types.Room | null) => void,
  setRoomIsChanging: React.Dispatch<React.SetStateAction<boolean>>,
  setRooms: (value: React.SetStateAction<Types.Room[]>) => void,
  setRefresh: (callback: (flag: boolean) => boolean) => void,
  roomSocket: WebSocket | null,
}

export const ChatBlock: React.FC<Props> = ({
  author,
  messages,
  setMessages,
  selectedRoom,
  roomIsChanging,
  newNameOfRoom,
  newMessageText,
  setNewNameOfRoom,
  setNewMessageText,
```

```
  setSelectedRoom,
  setRoomIsChanging,
  setRooms,
  setRefresh,
  roomSocket,
}) => {
  const messageListRef = useRef<HTMLUListElement | null>(null);
  const [socket, setSocket] = useState<WebSocket | null>(null);
  useEffect(() => {
    if (!selectedRoom) {
      return;
    }
    const createdSocket = createSocket({
      type: Types.SocketType.Message,
      url: API_WEBSOCKET,
      createData: { selectedRoom },
      actions: { setRooms, setMessages },
    });
    setSocket(createdSocket);
    return () => {
      createdSocket.close();
      setSocket(null);
    };
  }, [selectedRoom, setMessages, setRooms]);
  const sendMessage = async (text: string) => {
    if (socket && selectedRoom) {
      const sharedRoomKey = sessionStorage.getItem(`sharedRoomKey-
${selectedRoom.id}`);

      if (!sharedRoomKey) {
        console.error("Немає ключа чату для шифрування");
        return;
      }

      // Шифруємо повідомлення перед відправленням
      const encryptedText = await encryptionService.encryptMessage(text,
sharedRoomKey);
      createRequest({
        type: Types.RequestType.SendMessage,
        actions: {  },
        body: { userId: author?.id, messageText: encryptedText, roomId:
selectedRoom.id },
```

```
      errorText: Types.RequestError.SendMessage,
      socket,
    });
  }
};
const handleSubmitMessage = (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault();

  if (author && selectedRoom && newMessageText.trim()) {
    sendMessage(newMessageText.trim());
    setNewMessageText('');
  }
};
const editName = () => {
  if (selectedRoom && !newNameOfRoom.trim()) {
    setNewNameOfRoom(selectedRoom.name);
    setRoomIsChanging(false);
    return;
  }

  createRequest({
    type: Types.RequestType.RoomRename,
    actions: { setRoomIsChanging, setSelectedRoom },
    body: { id: selectedRoom?.id, newName: newNameOfRoom.trim() },
    errorText: Types.RequestError.RoomRename,
    socket: roomSocket,
  });
};
const handleEditName = () => {
  if (selectedRoom) {
    setNewNameOfRoom(selectedRoom.name);
    setRoomIsChanging(true);
  }
};
const closeHelper = () => {
  setSelectedRoom(null);
  setRoomIsChanging(false);
  setMessages([]);
  setNewMessageText('');
  setRefresh(cur => !cur);
  sessionStorage.removeItem(`sharedRoomKey-${selectedRoom?.id}`);
};
```

```
const handleDeleteRoom = (id: string | undefined) => {
  createRequest({
    type: Types.RequestType.RoomDelete,
    actions: {},
    body: { id },
    errorText: Types.RequestError.RoomDelete,
    socket: roomSocket,
  });
};
const scrollDown = () => {
  if (messageListRef.current) {
    messageListRef.current.scrollTop = messageListRef.current.scrollHeight;
  }
};
useEffect(() => scrollDown(), [messages]);

return (
  <section className={classNames('chat-block', {
    'chat-block__display': selectedRoom,
  })}>
    <article className="chat-block__info">
      {!roomIsChanging && (
        <h1 className='chat-block__info--title'>
          {selectedRoom ? `${selectedRoom.name}` : 'Назва чату...'}
        </h1>
      )}

      {selectedRoom && roomIsChanging && (
        <form className='chat-block__info--form' onSubmit={event => {
          event.preventDefault();
          editName();
        }}>
          <Input
            type={Types.Input.RoomRename}
            value={newNameOfRoom}
            placeholder={Types.PlaceHolder.RoomRename}
            onChange={setNewNameOfRoom}
            onBlur={editName}
          />
        </form>
      )}
```

```
        <div className={classNames('chat-block__info--buttons', {
          'chat-block__info--display': selectedRoom,
        })}>
          {author?.id === selectedRoom?.userId && (
            <>
              <Button type={Types.Button.Rename} onCLick={handleEditName} />
              <Button type={Types.Button.Delete} disabled={roomIsChanging}
onCLick={() => handleDeleteRoom(selectedRoom?.id)} />
            </>
          )}
          <Button type={Types.Button.Close} onCLick={closeHelper} />
        </div>
      </article>

      <article className="chat-block__chat">
        <ChatModale author={author} messages={messages}
selectedRoom={selectedRoom} />

        <ul ref={messageListRef} className="chat-block__chat--messages">
          {messages.map(message => (
            <MessageItem key={message.id} message={message} author={author} />
          ))}
        </ul>

        <form
          className={classNames('chat-block__form', {
            'chat-block__form--display': selectedRoom,
          })}
          onSubmit={handleSubmitMessage}
        >
          <Input
            type={Types.Input.Send}
            value={newMessageText}
            placeholder={Types.PlaceHolder.Send}
            onChange={setNewMessageText}
          />
          <Button type={Types.Button.Send} />
        </form>
      </article>
    </section>
  );
}
```

ДОДАТОК Ґ – index.js

```
/* eslint-disable no-console */
'use strict';

const express = require('express');
const cors = require('cors');
const { WebSocketServer } = require('ws');
const { Utils } = require('./utils/utils.js');
const { userRouter } = require('./routes/user.route.js');
const { roomRouter } = require('./routes/room.route.js');
const { messageRouter } = require('./routes/message.route.js');
const { websocket } = require('./websocket/websocket.js');
const { errorMiddleware } = require('./middlewares/errorMiddleware.js');

require('dotenv').config();
const PORT = process.env.PORT || 3002;
const app = express();

app.use(express.json());

app.use(
  cors({
    origin: '*',
    credentials: true,
  }),
);

Utils.initTables();

app.use('/user', userRouter);
app.use('/room', roomRouter);
app.use('/message', messageRouter);

const server = app.listen(PORT, () => {
  console.log('Server is running on port: ', PORT);
});

const wss = new WebSocketServer({ server });

websocket(wss);

app.use(errorMiddleware);
```

ДОДАТОК Д – websocket.js

```
const crypto = require('crypto');
const { roomService } = require('../services/room.service.js');
const { messageService } = require('../services/message.service.js');

// Зберігає спільні ключі для кожної кімнати
const rooms = {};

// Ініціалізація кімнати та ключа
const initializeRoom = (roomId) => {
  if (!rooms[roomId]) {
    rooms[roomId] = crypto.randomBytes(32).toString('base64');
  }

  return rooms[roomId];
};

const PING_INTERVAL = 30000; // 30 секунд

const websocket = (wss) => {
  wss.on('connection', (connection) => {
    let roomId;

    connection.isAlive = true;

    connection.on('pong', () => {
      connection.isAlive = true;
    });

    connection.on('message', async (newMessage) => {
      try {
        const data = JSON.parse(newMessage);

        // Ініціалізація кімнати
        if (data.type === 'init' && data.roomId) {
          roomId = data.roomId;
          connection.roomId = roomId;

          const joinRooms = await roomService.getRoomById(roomId);
          if (joinRooms) {
            // Ініціалізація спільного ключа для чату
            const roomKey = initializeRoom(roomId);
```

```
    connection.send(
      JSON.stringify({
        type: 'roomKey',
        key: roomKey,
      }),
    );

    // Надсилаємо історію зашифрованих повідомлень
    const messages = await messageService.getMessagesByRoomId(roomId);

    connection.send(
      JSON.stringify({
        type: 'roomMessages',
        value: messages,
      }),
    );
  } else {
    connection.send(JSON.stringify({ error: 'Room not found' }));
  }

  return;
}

// Надсилання повідомлення
if (data.type === 'newMessage' && roomId) {
  const { message } = data;
  // const message = await messageService.createMessage({
  //   userId,
  //   text,
  //   roomId,
  // });

  wss.clients.forEach((client) => {
    if (client.roomId === roomId) {
      client.send(
        JSON.stringify({
          type: 'newMessage',
          value: message,
        }),
      );
    }
```

```
  });

  return;
}

// Створення кімнати
if (data.type === 'roomCreated') {
  wss.clients.forEach((client) => {
    client.send(
      JSON.stringify({ type: 'roomCreated', room: data.room }),
    );
  });

  return;
}

// Редагування кімнати
if (data.type === 'roomRenamed') {
  wss.clients.forEach((client) => {
    client.send(
      JSON.stringify({ type: 'roomRenamed', room: data.room }),
    );
  });

  return;
}

// Видалення кімнати
if (data.type === 'roomDeleted') {
  wss.clients.forEach((client) => {
    client.send(
      JSON.stringify({
        type: 'roomDeleted',
        deletedRoomId: data.deletedRoomId,
      }),
    );
  });
  delete rooms[data.deletedRoomId];

  return;
}
```

```
        // Видалення користувача
        if (data.type === 'userLogout') {
          wss.clients.forEach((client) => {
            client.send(
              JSON.stringify({
                type: 'userLogout',
                userId: data.userId,
              }),
            );
          });
          roomService.getAllRooms().forEach((room) => delete rooms[room.id]);

          return;
        }

        // Якщо команда не розпізнана
        connection.send(JSON.stringify({ error: 'Unknown command' }));
      } catch (error) {
        console.error('WebSocket Error:', error);
        connection.send(JSON.stringify({ error: 'Server error' }));
      }
    });

    connection.on('close', () => {
      console.log(`Client left the room ${roomId}`);
    });
  });

  // Серверний heartbeat/ping для всіх клієнтів
  setInterval(() => {
    wss.clients.forEach((ws) => {
      if (ws.isAlive === false) {
        ws.terminate();
        return;
      }
      ws.isAlive = false;
      ws.ping();
    });
  }, PING_INTERVAL);
};
module.exports = { websocket };
```

ДОДАТОК Е – message.route.js

```
const express = require('express');
const { messageController } = require('../controllers/message.controller.js');
const { Utils } = require('../utils/utils.js');

const messageRouter = express.Router();

messageRouter.get(
  '/getMessages/:roomId',
  Utils.catchError(messageController.getMessagesByRoomId),
);

messageRouter.post(
  '/createMessage',
  Utils.catchError(messageController.createMessage),
);

module.exports = { messageRouter };
```

ДОДАТОК Є – message.controller.js

```
/* eslint-disable no-console */
const { messageService } = require('../services/message.service.js');

class MessageController {
  getMessagesByRoomId = async (req, res) => {
    const { roomId } = req.body;

    const messages = await messageService.getMessagesByRoomId(roomId);

    res.send(messages);
  };
  createMessage = async (req, res) => {
    const newMessage = await messageService.createMessage(req.body);

    res.send(newMessage);
  };
}

const messageController = new MessageController();

module.exports = { messageController };
```

ДОДАТОК Ж – message.service.js

```
const { Message } = require('../models/message.model.js');
const { ApiError } = require('../exceptions/api.error.js');
const { roomService } = require('./room.service.js');
const { userService } = require('./user.service.js');
class MessageService {
  getMessagesByRoomId = async (roomId) => {
    const messages = await Message.findAll({ where: { roomId } });
    return messages;
  };
  createMessage = async ({ roomId, userId, text }) => {
    const room = await roomService.getRoomById(roomId);
    const userName = await userService.getUserNameById(userId);
    if (!room) {
      throw ApiError.notFound({
        message: 'Room not found',
      });
    }
    if (!userName) {
      throw ApiError.notFound({
        message: 'User not found',
      });
    }
    if (!text) {
      throw ApiError.badRequest({
        message: 'All data are required',
      });
    }
    const message = await Message.create({
      userId,
      text,
      time: new Date(),
      roomId,
      userName,
    });

    return message;
  };
}
const messageService = new MessageService();

module.exports = { messageService };
```

ДОДАТОК З – db.js

```
const { Sequelize } = require('sequelize');

require('dotenv').config();

const client = new Sequelize({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  username: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_DATABASE,
  dialect: 'postgres',
  retry: {
    max: 10, // кількість спроб
  },
});

module.exports = { client };
```

ДОДАТОК И – errorMiddleware.js

```
const { ApiError } = require('../exceptions/api.error.js');

const errorMiddleware = (error, req, res) => {
  if (error instanceof ApiError) {
    const { status, message, errors } = error;

    res.status(status).send({
      message,
      errors,
    });

    return;
  }

  res.status(500).send({ message: 'Server error' });
};

module.exports = { errorMiddleware };
```

ДОДАТОК І – api.error.js

```
class ApiError extends Error {
  constructor({ message, status, errors = {} }) {
    super(message);
    this.status = status;
    this.errors = errors;
  }

  static badRequest(message, errors) {
    return new ApiError({
      message,
      errors,
      status: 400,
    });
  }

  static unauthorized(errors) {
    return new ApiError({
      message: 'Unauthorized',
      errors,
      status: 401,
    });
  }

  static notFound(errors) {
    return new ApiError({
      message: 'Not found',
      errors,
      status: 404,
    });
  }
}

module.exports = { ApiError };
```

ДОДАТОК Ї – ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

Програмна розробка захищеного Web-мессенджера для обміну повідомленнями в реальному часі

Здобувач:
Студент групи КН-21-1
Полотняк Євгеній
Геннадійович

Науковий керівник:
доктор технічних наук, доцент
Поплавський Олександр
Анатолійович

Київ – 2025

2 Вступ

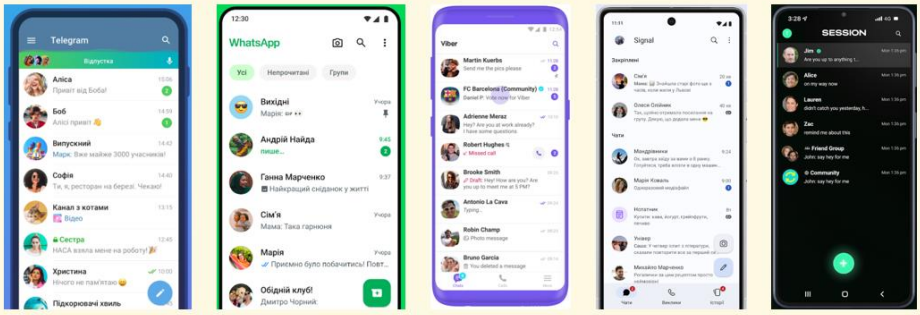
Мета роботи:

Розробка захищеного Web-месенджера для обміну повідомленнями в реальному часі з використанням наскрізного шифрування, який забезпечує конфіденційність, доступність через браузер та ергономічність.

Завдання дипломної роботи:

- ✓ Дослідити функціональні та безпекові особливості сучасних веб-месенджерів
- ✓ Створити клієнт-серверну архітектуру месенджера з підтримкою WebSocket
- ✓ Реалізувати шифрування повідомлень за принципом Double Ratchet
- ✓ Забезпечити групову комунікацію через захищені кімнати
- ✓ Спроєктувати інтуїтивно зрозумілий та адаптивний інтерфейс користувача
- ✓ Реалізувати робочий прототип месенджера

3 Популярні месенджери в Україні



4 Порівняльний аналіз месенджерів

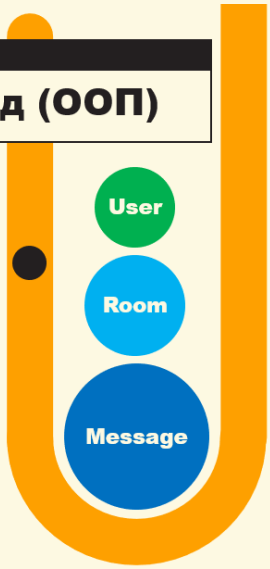
Критерій	Telegram	WhatsApp	Viber	Signal	Session
Web-версія	✅	✅	❌	❌	❌
Наскрізне шифрування	Частково	✅	Частково	✅	✅
Відкритий код	Частково	❌	❌	✅	✅
Централізація	✅	✅	✅	✅	❌
Анонімність	❌	❌	❌	Частково	✅

📌 **Висновок:**
Представлені месенджери не забезпечують одночасно ***конфіденційність***, ***відкритість***, ***анонімність*** та наявність ***web-інтерфейсу***. Це створює нішу для нового продукту, який буде поєднувати високий рівень захисту, відкриту архітектуру та web-доступність.

5 Об'єктно-орієнтований підхід (ООП)

Використано інтерфейси для опису основних сутностей:

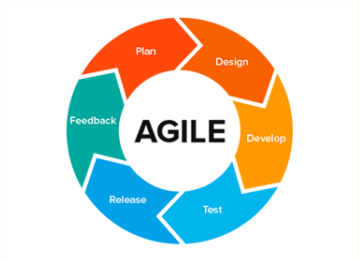
Принцип	Реалізація у проєкті
Інкапсуляція	Дані об'єктів приховані від прямого доступу
Наслідування	Спеціалізація інтерфейсів під типи повідомлень
Поліморфізм	Єдиний спосіб обробки різних типів повідомлень
Абстракція	Інтерфейси описують сутності з визначеними полями



🛠 **ООП забезпечує гнучкість, масштабованість та зручність супроводу коду.**

6 Методологія розробки: Agile

Характеристика	Реалізація в проєкті
Ітераційність	Робота поділена на цикли: **проектування → реалізація → тестування**
Гнучкість	Можливість змін у функціоналі на будь-якому етапі
Комунікація	Постійна взаємодія між frontend, backend та тестуванням
Пріоритетність	Спочатку реалізовано критичні функції (авторизація, обмін повідомленнями)
Зворотний зв'язок	Тестування після кожного етапу — швидке усунення недоліків



💡 **Agile дозволив:**
- ✓ ***оптимізувати розробку***
- ✓ ***мінімізувати ризики***
- ✓ ***швидко реалізувати MVP-версію месенджера.***

7 Основна архітектура системи

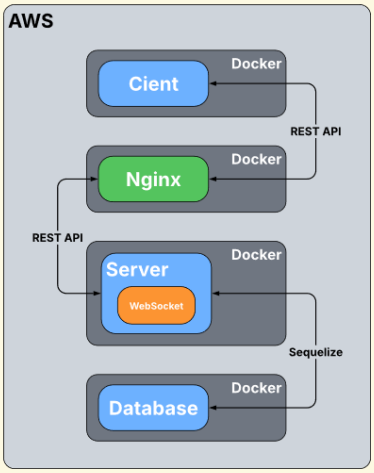
📌 **Компоненти розгорнуто в AWS-середовищі з використанням Docker-контейнерів:**

Client
- *браузерний інтерфейс на React, підключення до WebSocket та REST API*

Nginx
- *реверс-проксі, маршрутизація HTTPS-запитів*

Server
- *Node.js-логіка, WebSocket, REST API, взаємодія з БД через Sequelize*

Database
- *Зберігання зашифрованих повідомлень*



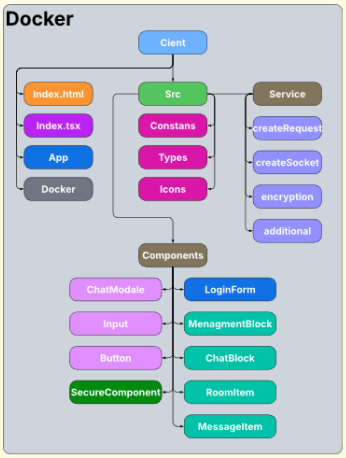
8 Архітектура клієнтської частини (frontend)

Технології: *React + TypeScript + SCSS + Axios + Docker*

Структура проєкту:
- *index.tsx, App, Docker – точка входу*
- *src/components/ – модульна побудова інтерфейсу*
- *services/ – запити, WebSocket, шифрування*
- *types/, constants/, icons/ – допоміжні модулі*

Ключові компоненти:
- *LoginForm, ChatBlock, RoomItem, MessageItem, SecureComponent*

🎯 **Реалізація забезпечує модульність, повторне використання коду та зручність підтримки.**

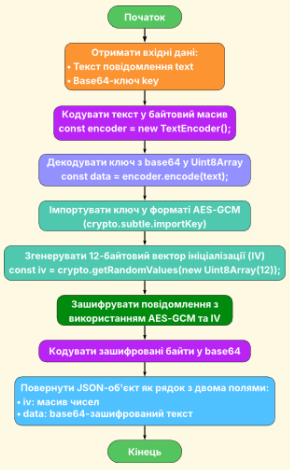


9 Шифрування повідомлень

Ключові етапи:

1. *Отримання тексту повідомлення та ключа шифрування*
2. *Кодування тексту та ключа у байтовий формат*
3. *Імпорт ключа у формат AES-GCM*
4. *Генерація вектора ініціалізації (IV)*
5. *Шифрування тексту з використанням ключа + IV*
6. *Формування JSON-об'єкта з iv та data*
7. *Оновлення ключа чата через **KDF***

🛡️ **Шифрування реалізовано на стороні клієнта за допомогою Web Crypto API (AES-256-GCM)**

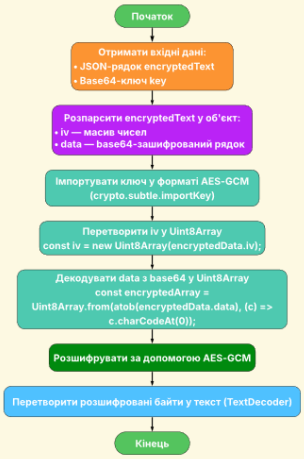


10 Дешифрування повідомлень

Ключові етапи:

1. *Отримання JSON-структури (iv, data)*
2. *Імпорт симетричного ключа*
3. *Перетворення IV та даних у байтові масиви*
4. *Розшифрування повідомлення через AES-GCM*
5. *Перетворення байтів у текст (TextDecoder)*
6. *Оновлення ключа чата через **KDF***

🔐 **Дешифрування виконується виключно на стороні клієнта. Сервер ніколи не бачить розшифрованих даних.**



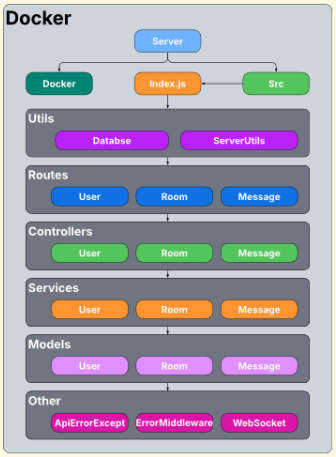
11 Архітектура серверної частини (backend)

Технології : *NodeJS + Sequelize + WebSocket + Docker*

Ключові модулі :

- *Index.js — точка входу до застосунку*
- *Routes / Controllers / Services / Models — розділення відповідальностей за функціональні блоки: User, Room, Message*
- *WebSocket — обробка обміну повідомленнями в реальному часі*
- *Utils / Middleware — підключення до БД, обробка помилок*

⚙️ **Така модульність спрощує масштабування, підтримку й розширення функціоналу.**

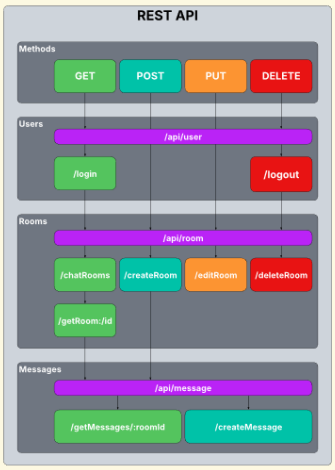


12 REST API: структура запитів

Групи запитів:

- **/api/user** — *вхід (/login), вихід (/logout)*
- **/api/room** — *перегляд (/chatRooms, /getRoom/:id), створення (/createRoom), редагування (/editRoom), видалення (/deleteRoom)*
- **/api/message** — *отримання (/getMessages/:roomId), надсилання (/createMessage)*

🔄 **Всі запити обробляються через відповідні контролери з валідацією та обробкою помилок.**



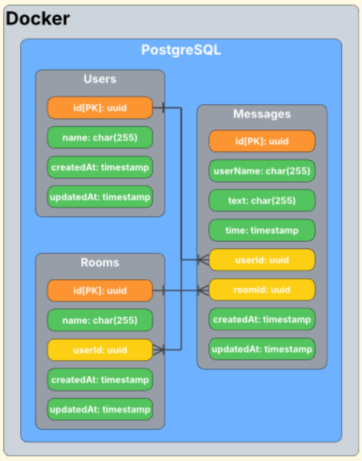
13 Архітектура бази даних

Таблиці:

- **Users**: *id, name, createdAt, updatedAt*
- **Rooms**: *id, name, userId (власник), createdAt, updatedAt*
- **Messages**: *id, userName, text, time, userId, roomId, createdAt, updatedAt*

🔗 **Таблиці пов'язані через userId і roomId**

💾 **Структура БД підтримує шифрування, зв'язки між сутностями та історію змін**



14 Подання інформації в базі даних

Таблиця Users:

- Користувачі: *Аня, Женя*

	id [PK] uuid	name character varying (255)	createdAt timestamp with time zone	updatedAt timestamp with time zone
1	72a022ee-03d6-4b09-8aeb-1f63daa12c7a	Аня	2025-05-23 00:27:58.854+00	2025-05-23 00:27:58.854+00
2	d7dc561c-8ba3-4c36-91a9-fe461273460a	Женя	2025-05-23 00:27:50.904+00	2025-05-23 00:27:50.904+00

Таблиця Rooms:

- Тестовий чат, Дніпро
- Зв'язок через userId

	id [PK] uuid	name character varying (255)	userId uuid	createdAt timestamp with time zone	updatedAt timestamp with time zone
1	50dc69eb-1d74-4536-89f6-1692b674c5...	Тестовий чат	d7dc561c-8ba3-4c36-91a9-fe46127346...	2025-05-23 00:29:19.843+00	2025-05-23 00:29:19.843+00
2	8d287c4e-2ad8-4c33-b8d2-efd29cf78aaa	Дніпро🌊	72a022ee-03d6-4b09-8aeb-1f63daa12c7a	2025-05-23 00:37:52.424+00	2025-05-23 00:46:22.508+00

Таблиця Messages:

- Повідомлення у зашифрованому вигляді (iv, data)
- Всі повідомлення пов'язані з roomId та userId

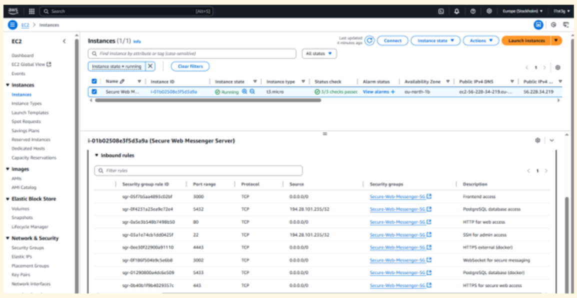
	id [PK] uuid	userName character va	text character varying (255)	time timestamp with time	createdAt timestam	updatedA timestan	roomId uuid	userId uuid
1	3564eb16-458e-4c3f-9...	Аня	{"iv":[164,72,137,98,200,171,204,113,121,230,198,107],"data":"H2ZXr22g7GShm98cISG74gwuGh7KbhVesUmTkIpDZw...	2025-05-23 00:41...	2025-...	2025...	8d287c4e-2...	72a022ee-03...
2	46b358ba-ad20-4286-...	Женя	{"iv":[155,106,164,116,108,123,221,85,47,210,31,119],"data":"IW182GQN3DDq7LXyz7Oy2rM9xsGzvoUcSgIg+gXIaArF...	2025-05-23 00:46...	2025-...	2025...	8d287c4e-2...	d7dc561c-8b...
3	7f78de47-2c13-42a0-9...	Аня	{"iv":[50,203,251,24,226,227,148,204,27,175,212,87],"data":"YAHtI3ZnwN+oB0saDF3hDB5SN0ADGfet2x2uAEXRAEC...	2025-05-23 00:30...	2025-...	2025...	50dc69eb-1...	72a022ee-03...
4	8666cd75-90fe-446f-b...	Женя	{"iv":[165,28,91,93,162,57,82,84,6,18,208,32],"data":"SBq2RszBcN4PXE7aiuVQ9LwkUP5Aa7Ay/GXFxMyhD6XFHkDyWj...	2025-05-23 00:30...	2025-...	2025...	50dc69eb-1...	d7dc561c-8b...
5	98649db4-814d-4670-...	Аня	{"iv":[237,15,43,52,193,105,147,234,26,59,232,118],"data":"O8Xk34ul9Y1nBBt7OKEZoHRLcCL09HYmzJmG9fyjzOe7G...	2025-05-23 00:39...	2025-...	2025...	8d287c4e-2...	72a022ee-03...
6	a6b24b33-a63f-49e2-b...	Аня	{"iv":[203,53,40,249,133,45,115,89,248,87,101,195],"data":"v0gzE1/bk5kmtJJqOJw4t6xvLg2Z5B7TDr5YDeig6mjSfdPs...	2025-05-23 00:31...	2025-...	2025...	50dc69eb-1...	72a022ee-03...
7	ab29e1d8-24bf-4f05-8...	Женя	{"iv":[18,63,82,27,168,181,63,132,239,172,80,120],"data":"cOwXS0EVZLYs4R2G3nMjDBUapBcqUx6iKin6ut/GKtZqnT9x...	2025-05-23 00:40...	2025-...	2025...	8d287c4e-2...	d7dc561c-8b...
8	ac6d6c8d-2c62-456a-...	Женя	{"iv":[184,146,165,32,137,72,112,114,26,77,28,68],"data":"4tOkbR3RH+Jlp5YNNJW97foCDCSJfoyl6nZupbsxTsfSFJU="}	2025-05-23 00:30...	2025-...	2025...	50dc69eb-1...	d7dc561c-8b...
9	bdc63189-66e6-468b-...	Женя	{"iv":[103,200,154,7,227,68,26,135,233,221,207,79],"data":"zb23Td0WMthGgWAy6y+1Y1Szbs/pfYAEOuITPSgnFE6Rjn...	2025-05-23 00:49...	2025-...	2025...	8d287c4e-2...	d7dc561c-8b...
10	d693e3d2-53d8-4eec-...	Аня	{"iv":[62,228,201,193,225,103,247,24,101,50,229,3],"data":"bae2i4D0MbcnbMgtr+YppqLBFe7i4xvSQ7/rbr4UCM8BDzM...	2025-05-23 00:46...	2025-...	2025...	8d287c4e-2...	72a022ee-03...
11	d8274ab4-835e-4fb1-8...	Аня	{"iv":[167,210,39,247,246,196,159,34,172,106,64,184],"data":"YNpOj46ssa4XYBzGSimHXm1+D8Nb3A0OhIWJNQ7NDs...	2025-05-23 00:45...	2025-...	2025...	8d287c4e-2...	72a022ee-03...
12	dd2a2dc6-9ca0-4671-...	Аня	{"iv":[140,163,207,101,206,146,228,56,27,245,101,108],"data":"IGM9P8NLQi/NBdtc9FFt2hHOhq5+Dvsi9xuxQOul4sA="}	2025-05-23 00:50...	2025-...	2025...	8d287c4e-2...	72a022ee-03...
13	f343e7d7-7e4f-436a-9...	Женя	{"iv":[215,245,158,99,216,170,114,204,69,28,162,202],"data":"9vtfSe4xC58+asR70jZlutxXsF0e+Ib5tOYRAjTKQ+kIjK8ft...	2025-05-23 00:40...	2025-...	2025...	8d287c4e-2...	d7dc561c-8b...
14	f3ef1072-7887-4495-a...	Женя	{"iv":[63,109,2,125,176,16,174,95,185,9,166,54],"data":"WwaXkI6z3kku/DkH0ZQUB5OZu1DmP42IUNH3TwiIqVteCABT...	2025-05-23 00:45...	2025-...	2025...	8d287c4e-2...	d7dc561c-8b...

🔐 **Повідомлення зберігаються у вигляді шифрованих об'єктів**

15 Середовище розгортання

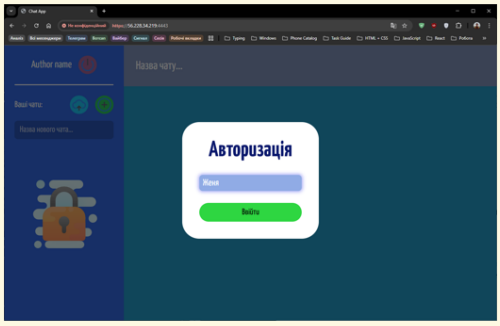
Характеристики:

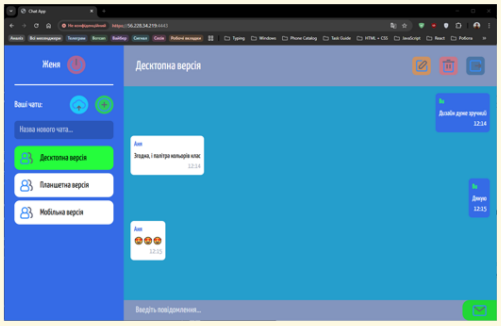
- Платформа: AWS EC2, тип інстансу: t3.micro
- Публічний доступ через HTTPS (порт 4443) та HTTP (порт 80)
- WebSocket: порт 3001, REST API: 443
- Сервер бази даних: PostgreSQL (порт 5432)



🚀 **Сервіс контейнеризовано у Docker, працює в хмарі на постійній основі**

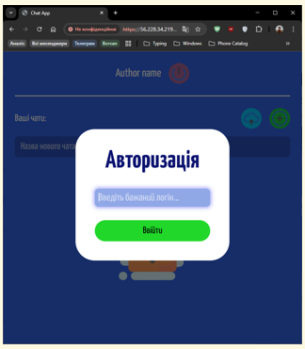
16 Інтерфейс Web-мессенджера (десктопна версія)

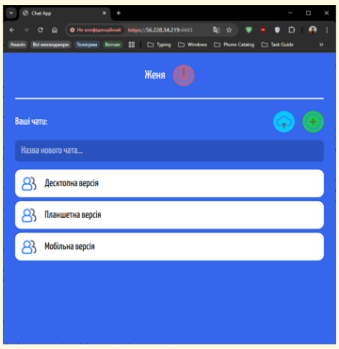


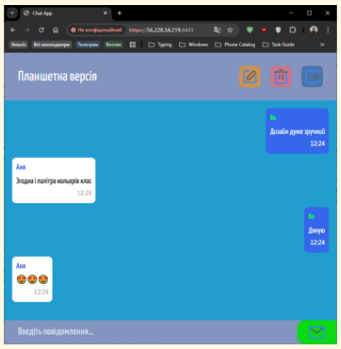


Інтерфейс оптимізований під розширення 900+ пікселів

17 Інтерфейс Web-мессенджера (планшетна версія)

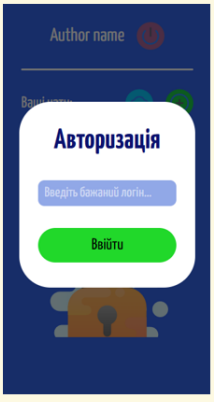


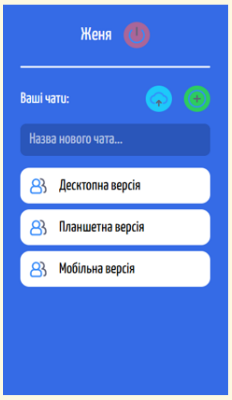


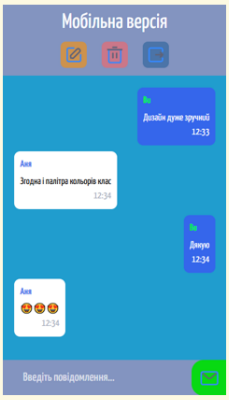


Інтерфейс оптимізований під розширення від 500 до 900 пікселів

18 Інтерфейс Web-мессенджера (мобільна версія)







Інтерфейс оптимізований під розширення від 320 до 500 пікселів

19 Висновки

У дипломній роботі досліджено функціональні та криптографічні особливості сучасних веб-месенджерів. Проведено порівняльний аналіз п’яти популярних рішень (Telegram, WhatsApp, Viber, Signal, Session), виділено їхні переваги та недоліки з точки зору безпеки, конфіденційності, доступності через браузер та підтримки групових чатів.

На основі аналізу сформульовано вимоги до власного веб-месенджера, у якому реалізовано наскрізне шифрування за алгоритмом **Double Ratchet**. Система побудована за **клієнт-серверною архітектурою**, реалізована з використанням **React**, **Node.js**, **PostgreSQL**, **Docker** та **WebSocket**, що дозволяє здійснювати обмін повідомленнями в реальному часі.

Спроєктовано адаптивний інтерфейс для **десктопних, планшетних і мобільних пристроїв**, розроблено REST API, логіку обміну повідомленнями та структуру збереження даних у базі. Система успішно розгорнута в хмарному середовищі **AWS EC2**.

У результаті виконання поставлених завдань створено функціональний захищений прототип веб-месенджера, який відповідає актуальним вимогам щодо конфіденційності та зручності використання. Отримані результати можуть бути основою для подальшого розширення функціоналу, зокрема додавання підтримки медіа та мобільних застосунків.