МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ І СИСТЕМИ
(назва факультету)

ЕЛЕКТРОННІ ОБЧИСЛЮВАЛЬНІ МАШИНИ
(повна назва кафедри)

До захисту

Пояснювальна записка
до кваліфікаційної роботи
ОС магістр
(ступінь вищої освіти)

Тема: ДОСЛІДЖЕННЯ ТА РОЗРОБКА ФЕДЕРАТИВНОЇ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕНЬ З ВИКОРИСТАННЯМ НАСКРІЗНОГО ШИФРУВАННЯ

за освітньою програмою: " Комп'ютерна інженерія "

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

Виконав: студент: групи КС2326

(підпис студента) Данііл НАБОКОВ (Ім'я ПРІЗВИЩЕ)

Керівник: (підпис) к. т. н., доц. Олег ЄГОРОВ (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: (підпис) к. т. н., доц. Олег ЄГОРОВ (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (підпис)

Дніпро – 2025 рік

Дніпро – 2024 рік

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
UKRAINIAN STATE UNIVERSITY OF SCIENCE AND TECHNOLOGIES

COMPUTER TECHNOLOGIES AND SYSTEMS
(faculty)

ELECTRONIC COMPUTING MACHINES
(department)

Explanatory Note
to Master's Thesis
Master
(higher education degree)

on the topic: RESEARCH AND DEVELOPMENT OF A FEDERATED MESSAGING SYSTEM USING END-TO-END ENCRYPTION

according to educational curriculum «Public Administration»

in the Speciality: Computer Engineering
(speciality and its code)

Done by the student of the group: Daniil NABOKOV
(name, surname)

Scientific Supervisor: Oleg EGOROV
(position, name, surname)

Normative controller :
(position, name, surname)

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Електронні обчислювальні машини»
Рівень вищої освіти: «Другого (магістерського) рівня вищої освіти»
Освітня програма: «Комп'ютерна інженерія»
Спеціальність: 123 – Комп'ютерна інженерія
(шифр та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ЕОМ
_____ Ігор ЖУКОВИЦЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)
Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу ОС «магістр»
(ступінь вищої освіти)

студенту Набокову Даніїлу Олеговичу
(Прізвище, Ім'я По батькові)

1. Тема роботи: Дослідження та розробка федеративних систем обміну повідомлень з використанням наскрізного шифрування
Керівник роботи: Єгоров Олег Йосипович, кандидат технічних наук, доцент
(Прізвище, Ім'я, По батькові, науковий ступінь, вчене звання)

затверджені наказом від «10» 10. 2024 р. № 1268ст

2. Строк подання студентом роботи: 22.01.2025 р.
3. Вихідні дані до роботи: рекомендації щодо підвищення ефективності та безпеки федеративного обміну повідомленнями, власна система обміну повідомлень, що забезпечує масштабованість і стабільність при великих навантаженнях.

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
4.1 Аналіз архітектури та функціональних особливостей сучасних систем обміну повідомлень
4.2 Визначення вимог сучасної системи обміну повідомлень
4.3 Розробка власної системи обміну повідомлень
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Не передбачено

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав: (підпис консультанта, дата)	Завдання прийняв: (підпис студента, дата)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ. Аналіз архітектури та функціональних особливостей сучасних систем обміну повідомлень.		30%
2	Планування виконання кваліфікаційної роботи та визначення вимог системи		60%
3	Розробка власної системи обміну повідомлень		100%
4	Подання кваліфікаційної роботи до кафедри		
5	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії		

Студент _____ (підпис) Даніїл НАБОКОВ (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) Олег ЄГОРОВ (Ім'я ПРІЗВИЩЕ)

Міністерство освіти і науки України
Український державний університет науки і технологій

Відгук керівника

кваліфікаційної роботи **магістра**
(ступінь вищої освіти)

Студент групи КС2326 Набоков Даніїл Олегович
(шифр групи) (Прізвище, Ім`я, По батькові)

Тема випускної роботи: Дослідження та розробка федеративної системи обміну повідомлень з використанням наскрізного шифрування

1. Якісні відмінності кваліфікаційної роботи: робота поєднує детальний аналіз протоколів та успішну реалізацію наскрізного шифрування, демонструючи високу надійність та гнучкість; особливу увагу приділено стабільної синхронізації між серверами та захисту даних навіть за багатовузлової взаємодії, що показує глибоке розуміння принципів розподілених систем.

2. Зауваження: прототип працює стабільно, хоча окремі аспекти взаємодії між серверами вимагають додаткової оптимізації, а інтерфейс занадто спрощено. Загалом зауваження не впливає на цілісність роботи.

3. Висновок щодо дотримання академічної доброчесності: всі принципи академічної доброчесності дотримані. Використані джерела були цитовані, а власні результати представлені чітко і обґрунтовано.

Комплексна оцінка кваліфікаційної роботи: робота відповідає вимогам до кваліфікаційних робіт, має високу практичну цінність. Робота виконана на високому професійному рівні та заслуговує оцінки відмінно.

Керівник: к.н.т. доц. Олег ЄГОРОВ
(посада) (Ім'я, ПРІЗВИЩЕ) (підпис)

Дата: 22.01.2025

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра:
66 с., 10 рис., 10 табл., 64 джерел.

Об’єкт розробки – федеративна система обміну повідомлень з використанням наскрізного шифрування (end-to-end-encryption).

Мета роботи – дослідження та критичний аналіз наявних рішень у сфері федеративного обміну повідомленнями, розроблення методології та архітектурних рішень для створення власної системи, що забезпечує високий рівень безпеки та конфіденційності шляхом застосування механізмів наскрізного шифрування.

Методи дослідження – порівняльний аналіз існуючих протоколів обміну повідомленнями, математичне моделювання криптографічних механізмів, проектування розподілених систем, аналіз безпеки комунікаційних протоколів, експериментальне тестування алгоритмів шифрування.

Одержані результати – проведено комплексне дослідження сучасних підходів до федеративного обміну повідомленнями з наскрізним шифруванням; систематизовано існуючі рішення, визначено їх переваги та обмеження; проаналізовано криптографічні протоколи та механізми забезпечення приватності у розподілених комунікаційних системах; сформульовано рекомендації щодо підвищення ефективності та безпеки федеративного обміну повідомленнями; розроблено власну систему з достатнім рівнем безпеки, що забезпечує масштабованість і стабільність при великих навантаженнях.

Ключові слова: ФЕДЕРАТИВНІ СИСТЕМИ, НАСКРІЗНЕ ШИФРУВАННЯ (E2EE), DIFFIE–HELLMAN, КРИПТОГРАФІЧНІ ПРОТОКОЛИ, УПРАВЛІННЯ КЛЮЧАМИ, ОБМІН ПОВІДОМЛЕННЯМИ.

ЗМІСТ

ВСТУП

Комунікація є основою будь-якого суспільства, і без неї неможливе повноцінне функціонування жодної системи. Повідомлення стали ключовим інструментом передачі інформації, завдяки яким забезпечується взаємодія між людьми, пристроями, сервісами та цілими системами. Сучасні системи обміну повідомленнями переживають стрімку еволюцію, особливо в контексті підвищення вимог до безпеки та конфіденційності. Ця еволюція пов'язана зі зростанням обсягів цифрової взаємодії та потребою захищати особисту інформацію від несанкціонованого доступу. Це обумовлено як зростаючою складністю кіберзагроз, так і вимогами користувачів до приватності, яка має бути гарантована навіть у разі передачі даних через сторонні сервери.

Особливий інтерес становлять федеративні системи обміну повідомленнями. Вони представляють собою новий підхід до організації захищених комунікацій, коли користувачі можуть безпечно взаємодіяти через децентралізовані сервери. Федеративні системи дозволяють обмінюватися повідомленнями між різними платформами, керованими різними адміністраторами, зберігаючи автономність кожного сервера та конфіденційність даних. Для цього сервери обмінюються сертифікатами, які підтверджують автентичність даних, і забезпечують синхронізацію подій, що зачіпають інші сервери. Наприклад, коли користувач A на сервері X надсилає запит користувачу B на сервері Y, сервер X повідомляє сервер Y про зміни.

Використання шифрування в такому випадку обов’язкове, оскільки інформація може бути поширена між багатьма серверами, що може викликати проблеми з безпекою. Наскрізне шифрування гарантує, що тільки відправник і отримувач мають доступ до змісту повідомлень, навіть якщо дані проходять через кілька проміжних серверів, також слід згадати, що дані, які передаються між користувачами, шифруються так, що навіть сервери, через які вони проходять, не повинні мати доступу до їхнього змісту.

Нами було розроблено прототип (proof of concept) подібної системи обміну повідомленнями з наскрізним шифруванням, що включає як серверну, так і клієнтську частини. Сервер був реалізований на мові програмування Elixir, з використанням потужної бібліотеки Plug, що забезпечує ефективну обробку HTTP-запитів та зручну інтеграцію з іншими сервісами. Клієнтська частина була побудована на Phoenix — веб-фреймворку, також написаному на Elixir, що дозволяє створювати високопродуктивні та масштабовані веб-додатки. Такий вибір технологій був обґрунтований високою надійністю та масштабованістю цих інструментів, що є критичними для забезпечення безперебійної роботи системи в умовах великих навантажень і численних одночасних користувачів.

РОЗДІЛ 1
АНАЛІЗ АРХІТЕКТУРИ ТА ФУНКЦІОНАЛЬНИХ ОСОБЛИВОСТЕЙ СУЧАСНИХ СИСТЕМ ОБМІНУ ПОВІДОМЛЕНЬ

1.1 Аналіз становлення та розвитку систем обміну повідомленнями

Перші системи обміну повідомленнями з'явилися ще в 1960-х роках, коли комп'ютери почали використовуватися для комунікаційних цілей. Однією з перших таких систем була наявна в операційній системі CTSS (Compatible Time-Sharing System), розроблена в Массачусетському технологічному інституті (MIT). Вона дозволяла користувачам відправляти короткі текстові повідомлення один одному, використовуючи загальні обчислювальні ресурси. [1]

Електронна пошта стала першою масштабною системою обміну повідомленнями, яка отримала широке поширення. Винахід електронної пошти приписують Рейю Томлінсону, який у 1971 році створив програму для відправки повідомлень між комп'ютерами, використовуючи символ "@" для розділення імені користувача та доменного імені. Ця система стала основою для сучасної електронної пошти і використовувала протокол SMTP (Simple Mail Transfer Protocol), який дозволяв різним поштовим серверам взаємодіяти між собою. [2][3]

У 1980-х роках з’явився IRC (Internet Relay Chat) — один із перших протоколів для обміну текстовими повідомленнями в реальному часі, який відіграв важливу роль у розвитку онлайн-комунікацій. IRC дозволяє користувачам створювати канали для групових чатів або вести приватні бесіди, що давало можливість обмінюватися повідомленнями на глобальному рівні. Однак, він був порівняно простим і не включав багатьох сучасних функцій. IRC став основою для розвитку багатьох інших систем миттєвих повідомлень, але він не забезпечував федерацію між різними сервісами, тобто користувачі могли спілкуватися лише в межах одного серверу IRC. [4]

З початком 1990-х років стали з’являтися нові месенджери, які намагалися перевершити IRC, додаючи більше зручності та нових можливостей для користувачів. Одним із таких був ICQ, розроблений ізраїльською компанією Mirabilis у 1996 році. ICQ стала одним із перших месенджерів, який дозволяв користувачам не тільки обмінюватися миттєвими повідомленнями, але й використовувати список контактів, отримувати офлайн-повідомлення, а також використовувати інші функції, які згодом стали стандартом для більшості IM-сервісів, зокрема підтримка статусів користувачів і групових чатів. [5]

У кінці 1990-х років, був розроблений XMPP (Extensible Messaging and Presence Protocol) — один із перших відкритих стандартів для федеративного обміну миттєвими повідомленнями. Розроблений у 1999 році, XMPP дозволяв з’єднувати різні IM-сервіси, що відкривало нові можливості для спілкування між користувачами з різних платформ. Зокрема, XMPP забезпечував можливість створення федерацій серверів, що дозволяло інтегрувати різні системи обміну повідомленнями в одну єдину мережу. Це стало основою для створення таких популярних платформ, як Google Talk і Jabber. XMPP став першим великим кроком у розвитку відкритих стандартів у сфері миттєвих повідомлень, що дозволило подолати обмеження закритих систем, таких як ICQ, і створити більш гнучкі та взаємодіючі платформи для глобального обміну повідомленнями. [6][7]

З появою смартфонів у кінці 2000-х років відбулася революція у сфері обміну повідомленнями. Мобільні додатки для обміну повідомленнями, такі як WhatsApp, Viber, Telegram та Signal, стали надзвичайно популярними завдяки своїй зручності, швидкості та можливості надсилання мультимедійних повідомлень, що значно перевищувало функціональність традиційних SMS-повідомлень. Ці платформи також почали включати додаткові функції, як-от голосові та відеодзвінки, а також адекватне та зручне шифрування для забезпечення конфіденційності та безпеки обміну даними. Завдяки цим інноваціям вони швидко завоювали довіру користувачів і стали важливими інструментами як для особистих, так і для професійних комунікацій. З часом ці

месенджери почали розширювати свої можливості, перетворюючись на універсальні платформи для обміну інформацією, організації роботи та навіть ведення бізнесу. [8]

1.2 Розвиток і впровадження шифрування в системах обміну повідомленнями

З розвитком технологій та зростанням кіберзагроз шифрування в системах обміну повідомленнями пройшло тривалий еволюційний шлях, перетворившись із факультативного інструменту на обов'язковий компонент кожної сучасної системи. У найперших системах, таких як CTSS і ранні електронні поштові сервіси, шифрування не застосовувалося, оскільки головними цілями були забезпечення базової функціональності та спільний доступ до ресурсів. У той час ще не існувало достатньої обчислювальної потужності для впровадження ефективного шифрування, а проблема конфіденційності не була гострою через обмежену кількість користувачів.

Перший вагомий прорив у криптографії стався у 1970-х роках із початком використання асиметричного шифрування в комп’ютерних системах. Алгоритм RSA, створений у 1977 році, відкрив можливість захищеного обміну ключами, що стало основою для багатьох систем, таких як протоколи SSL/TLS, які згодом інтегрувалися у веб-браузери та поштові сервіси. Однак, впровадження цих технологій було поступовим і часто залишається на розсуд адміністратора або розробників системи. [9][10]

У 1980-х роках, коли електронна пошта почала використовуватися ширше, потреба у шифруванні стала більш нагальною. Одним із перших рішень стала спроба впровадити стандарт PEM (Privacy Enhanced Mail) у 1993 році, проте складність налаштування та відсутність підтримки обмежили його застосування. [11]

Важливий прорив у шифруванні електронної пошти відбувся у 1991 році із запуском PGP (Pretty Good Privacy), розробленого Філом Циммерманом. Цей

інструмент став революційним завдяки своїй здатності забезпечувати як шифрування повідомлень, так і їхній цифровий підпис. PGP використовував асиметричне шифрування RSA для обміну ключами та симетричне шифрування IDEA для захисту вмісту повідомлень. Однак, складність його налаштування обмежувала широке впровадження серед пересічних користувачів. [12]

ProtonMail, запущений у 2014 році, популяризував наскрізне шифрування, використовуючи асиметричну криптографію для обміну ключами, які генеруються на пристрої користувача, та симетрична для збереження повідомлень у зашифрованому вигляді, однак проблеми виникають при відправленні на зовнішні платформи, так якщо сервіс не підтримує шифрування, то лист перед відправленням буде розшифровано. [13]

У 1990-х роках, із поширенням інтернету та появою месенджерів, конфіденційність повідомлень стала викликом. ICQ, AIM і Yahoo! Messenger не забезпечували шифрування на рівні клієнта. Повідомлення передавалися у відкритому вигляді, що робило їх вразливими до перехоплення. Для з'єднання між серверами використовувався протокол HTTPS, але він не захищав повідомлення від доступу самих серверів. [14]

У 2000-х роках у протоколі XMPP (Extensible Messaging and Presence Protocol) були впроваджені механізми для забезпечення шифрування, які суттєво підвищили безпеку передачі даних. У 2004 році протокол отримав підтримку TLS (Transport Layer Security), що дозволило зашифрувати з'єднання між клієнтами та серверами, а також між серверами, зменшуючи ризик перехоплення повідомлень під час їхнього транспортування. Пізніше були інтегровані додаткові рівні безпеки, такі як автентифікація через SASL (Simple Authentication and Security Layer). [15]

Окрім того, XMPP отримав підтримку наскрізного шифрування (E2EE) через розширення OMEMO, яке базується на Signal Protocol і забезпечує динамічне оновлення ключів, гарантуючи конфіденційність навіть у децентралізованій мережі. [16]

Також XMPP підтримує OpenPGP для шифрування та підпису повідомлень, що дозволяє забезпечити високий рівень безпеки для користувачів, які хочуть використовувати асиметричне шифрування для захисту своїх повідомлень. [17]

Крім того, для деяких додатків XMPP є підтримка OTR (Off-the-Record), протоколу, який додає наскрізне шифрування та можливість "заперечуваної автентифікації", проте він вважається застарілим. [18]

Нову епоху започаткувала поява Signal Protocol у 2010-х роках. Цей протокол став революційним завдяки інтеграції Double Ratchet, що забезпечує динамічне оновлення ключів з кожним новим повідомленням. [19]

Наскрізне шифрування (E2EE) стало стандартом для мобільних додатків, таких як Signal, WhatsApp, Facebook Messenger. Signal Protocol використовує симбіоз симетричних і асиметричних алгоритмів: AES-256 для шифрування вмісту, HMAC-SHA256 для автентифікації повідомлень і X3DH (Extended Triple Diffie-Hellman) для обміну ключами. Таке поєднання гарантує безпеку навіть у випадку компрометації одного з ключів. [20]

У паралель до цього Telegram створив свій власний протокол MTProto, який не отримав широкого визнання за межами власної платформи, орієнтований на ефективність і швидкість. Він використовує симетричне шифрування AES-256, асиметричну криптографію RSA-2048 і алгоритм Diffie-Hellman для обміну ключами. Telegram дозволяє вибіркове E2EE лише для секретних чатів, залишаючи групові чати менш захищеними, що викликало критику експертів із безпеки. [21]

У федеративних системах, таких як Matrix, з'явилися більш складні виклики, пов'язані з груповими чатами та синхронізацією ключів між багатьма пристроями. Для цього був розроблені протоколи Olm та Megolm, які забезпечує наскрізне шифрування навіть у великих чатах. Megolm використовує довготривалий ключ сесії, знижуючи обчислювальні витрати, але підтримує високий рівень конфіденційності за рахунок захисту ключа розповсюдження. [22]

1.3 Виникнення та розвиток федеративних систем обміну повідомленнями

Федеративні системи - це складна мережева архітектура, яка дозволяє незалежним системам спілкуватися та взаємодіяти, зберігаючи при цьому їхню індивідуальну операційну автономію. Цей підхід докорінно змінює спосіб взаємодії цифрових комунікаційних платформ, виходячи за рамки традиційних централізованих моделей. У федеративній системі кожен сервер або вузол функціонує як незалежна одиниця з власною інфраструктурою управління, базою користувачів і політиками. Важливо, що ці вузли можуть безперешкодно обмінюватися інформацією за допомогою стандартизованих протоколів зв'язку. Такий дизайн гарантує, що користувачі не обмежені однією платформою, що дозволяє створювати ширші та гнучкіші комунікаційні мережі. [23]

Електронна пошта є однією з найуспішніших реалізацій федеративної системи зв'язку та однією з перших. На відміну від централізованих платформ обміну повідомленнями, електронна пошта дозволяє незалежним серверам з різних організацій безперешкодно спілкуватися, створюючи глобальну розподілену мережу, в якій жодна організація не контролює маршрутизацію повідомлень або взаємодію користувачів. Федеративність електронної пошти відбувається переважно через протокол SMTP, який діє як складний дипломатичний механізм міжсерверної комунікації. [3]

Коли електронний лист надсилається, сервер-відправник повинен домовитися про передачу з сервером одержувача, функціонуючи подібно до міжнародних дипломатичних каналів, що полегшують комунікацію між суверенними державами. Кожна адреса електронної пошти містить важливий елемент федерації: домен. Коли повідомлення надсилається, сервер-відправник виконує DNS-пошук для ідентифікації поштового сервера одержувача, визначаючи точну маршрутизацію через потенційно тисячі незалежних серверних інфраструктур. Цей процес демонструє основний принцип федерації - автономні системи взаємодіють за допомогою стандартизованих протоколів.

Зв'язок між серверами передбачає складні процеси автентифікації і перевірки. Сервери обмінюються обліковими даними, перевіряють ідентичність один одного та узгоджують параметри передачі.

Такі протоколи, як SPF, DKIM і DMARC, функціонують як дипломатичні повноваження, гарантуючи, що тільки авторизовані сервери можуть обмінюватися повідомленнями, і запобігаючи несанкціонованим або зловмисним передачам. [24][25][26]

У той час як електронна пошта заклала фундаментальні принципи федеративної комунікації, XMPP представляє більш просунутий і динамічний підхід до децентралізованого обміну повідомленнями. XMPP відрізняється принципово розподіленою архітектурною моделлю, яка дозволяє спілкуватися в режимі реального часу між різними незалежними мережами. На відміну від асинхронної передачі повідомлень електронної пошти, XMPP надає миттєву інформацію про присутність. [6]

Сервери можуть миттєво обмінюватися та оновлювати статус доступності користувачів, що дозволяє динамічно відстежувати зв'язок між доменами, XMPP підтримує справжній двонаправлений зв'язок зі збереженням стану. [27]

Сервери підтримують активні з'єднання, забезпечуючи миттєвий обмін повідомленнями, сповіщення в реальному часі та постійні сеанси зв'язку, які виходять за рамки простої маршрутизації повідомлень. Протокол підтримує розширені сценарії зв'язку за допомогою модульної системи розширення. Сервери можуть реалізовувати користувацькі функції, такі як багатокористувацькі чати, передача файлів та інструменти для спільної роботи, не змінюючи основну інфраструктуру. [28]

Matrix виник як інноваційний протокол зв'язку з відкритим стандартом, розроблений для усунення фундаментальних обмежень в існуючих технологіях обміну повідомленнями. На відміну від попередніх федеративних систем, Matrix принципово переосмислює комунікаційну інфраструктуру як децентралізовану, безпечну та інтероперабельну мережу. Протокол запроваджує революційний підхід до обміну повідомленнями, розглядаючи

комунікацію як спільну, синхронізовану структуру даних. Кожне повідомлення розглядається як незмінна подія, яка може бути реплікована між декількома серверами, створюючи розподілений стан розмови, який підтримує узгодженість незалежно від інфраструктури сервера. Matrix унікально синхронізує стани бесіди на декількох серверах, гарантуючи, що учасники можуть приєднуватися і переглядати історію бесіди з різних точок входу. Це кардинально відрізняється від попередніх протоколів, де контекст розмови залежав від сервера. Кожна нова подія має посилання на попередні події, створюючи ланцюжок залежностей. Це дозволяє серверам синхронізувати стан кімнати, навіть якщо повідомлення передаються через різні маршрути або з певними затримками. [29][30]

Matrix забезпечив безпрецедентну інтероперабельність, створивши протокольні мости, які дозволяють спілкуватися між принципово різними платформами обміну повідомленнями. Тепер користувачі можуть взаємодіяти між Slack, Telegram та іншими мережами, зберігаючи при цьому узгоджений досвід спілкування. На відміну від традиційних систем обміну повідомленнями, Matrix зберігає історію розмов і контекст на різних серверах і пристроях. Такий підхід гарантує, що безперервність комунікації підтримується незалежно від змін в інфраструктурі або продуктивності окремих серверів. [31]

З недавніх пір WhatsApp має наміри стати подобою федеративної системи, щоб відповідати новому закону ЄС, Акту про цифрові ринки (DMA), який набув чинності 7 березня 2024. Інтероперабельність WhatsApp знаменує важливий перехід від закритих екосистем до відкритіших комунікаційних моделей. Використовуватися буде протокол сигналу, а повідомлення будуть пакуватися в XML. [32]

1.4 Порівняння протоколів федеративних системах

У цьому дослідженні були обрані чотири різні системи обміну повідомленнями для порівняння: Email, XMPP, Matrix та Signal. Кожна з цих

систем має власні характеристики, що робить їх цікавими з технічної точки зору та корисними для порівняння в контексті безпеки, інтероперабельності, шифрування та функціональності.

- Signal: система, яка акцентує увагу на конфіденційності та безпеці. Вона підтримує наскрізне шифрування і є одним із стандартів для захищеного обміну повідомленнями. Її відкритий вихідний код і фокус на анонімності роблять її ідеальним прикладом для дослідження систем з високим рівнем безпеки.
- Email: класична система для обміну повідомленнями, яка використовується по всьому світу вже кілька десятиліть. Вона є однією з перших реалізацій федеративних систем обміну повідомленнями, що дає змогу обмінюватися листами між незалежними серверами. Вона має велику важливість через свою стародавню інфраструктуру, стандартні протоколи та широку застосовність.
- XMPP (Extensible Messaging and Presence Protocol): Цей протокол обміну повідомленнями використовує федеративну архітектуру, яка дає змогу різним серверам взаємодіяти між собою. XMPP надає гнучкість у створенні чат-систем, завдяки чому він активно використовується як для особистого, так і для корпоративного спілкування. Його відкрите походження і підтримка різноманітних розширень роблять його цікавим для порівняння.
- Matrix: Це децентралізований протокол обміну повідомленнями, який вирізняється своєю унікальною архітектурою та підтримкою наскрізного шифрування. Matrix також дозволяє інтегрувати різні платформи завдяки використанню протокольних мостів, що робить його важливим кандидатом для аналізу інтероперабельності та масштабованості.

Для проведення всебічного порівняння систем обміну повідомленнями було визначено ключові характеристики (таблиця 1.1), які відображають основні аспекти їхньої роботи.

Таблиця 1.1 – Характеристики систем обміну повідомлень

Характеристика	Опис
Шифрування	Шифрування є основним механізмом захисту даних від несанкціонованого доступу. Оскільки всі системи мають різні підходи до шифрування (наскрізне, транспортне чи комбіноване), це дозволяє порівняти їх рівень безпеки та захисту інформації.
Механізм обміну ключами	Обмін ключами є важливою частиною процесу забезпечення безпеки при використанні шифрування. Різні методи обміну ключами можуть впливати на зручність використання системи та на її вразливість до атак. Аналіз цього процесу дозволяє порівняти ефективність та зручність безпечної комунікації.
Відкритість протоколу імплементацій	Відкритість протоколу дозволяє здійснювати аудити безпеки та виявляти потенційні уразливості. Відкриті протоколи дозволяють стороннім розробникам створювати нові додатки або додаткові функції, що підвищує гнучкість і адаптивність системи.
Інтероперабельність	Інтероперабельність — це важливий аспект для систем, що повинні взаємодіяти з іншими платформами чи сервісами. Можливість інтеграції з іншими системами дозволяє користувачам обмінюватися повідомленнями між різними сервісами без обмежень, що є важливим для розвитку екосистеми.
Процес комунікації серверів	Архітектура серверів і їх взаємодія є ключовим аспектом для визначення ефективності та масштабованості системи. Різні підходи до організації комунікації між серверами впливають на швидкість доставки повідомлень, стійкість до збоїв та здатність до масштабування.
Протокол транспортування і комунікацій	Який протокол (TCP, UDP, QUIC) використовується для передачі даних між клієнтами та серверами та/або який протокол використовується для
Формат повідомлень	Формат повідомлень визначає структуру даних, що передаються через систему. Стандартизація формату повідомлень допомагає зберігати сумісність між різними платформами та забезпечувати ефективне кодування та декодування даних.
Групові чати	Групові чати є важливим елементом багатьох систем обміну повідомленнями. Порівняння того, як різні системи реалізують групові чати (кількість учасників, адміністрування, синхронізація) дає змогу оцінити їх здатність обробляти колективну комунікацію.

Продовження таблиці 1.1

Характеристика	Опис
Усунення конфліктів між серверами	Питання синхронізації стану між різними серверами в умовах асинхронної доставки повідомлень є критичним для забезпечення консистентності даних. Порівняння того, як різні системи вирішують проблему "event races", дозволяє оцінити їх стійкість та надійність.
Захищеність	Важливою характеристикою будь-якої системи є її здатність протистояти загальновідомим атакам, таким як атаки на відмову в обслуговуванні (DoS), атакуючі техніки на основі соціальної інженерії, спуфінг або перехоплення повідомлень, отримання ключів шифрування, тощо. Оцінка захисту від таких атак допомагає зрозуміти загальний рівень безпеки системи.
Аутентифікація	Механізм аутентифікації визначає, як користувачі і сервери перевіряють свою ідентичність. Це може включати традиційну перевірку пароля, двофакторну аутентифікацію або більш складні методи, такі як криптографічні сертифікати. Оцінка цього процесу дозволяє зрозуміти, наскільки система здатна запобігати несанкціонованому доступу.

Таким чином, ці характеристики дозволять всебічно порівняти (таблиця 1.2) різні системи обміну повідомленнень з точки зору безпеки, ефективності, зручності та інтеграції з іншими сервісами, та іншими наведеними характеристиками.

Таблиця 1.2 - Порівняння характеристик систем та протоколів

Характеристика	Email	XMPP	Matrix	Signal
Шифрування	Нема шифрування за замовчуванням	OMEMO, OTR (legacy), PGP	Olm, Megolm	libsignal
Механізм обміну ключами	Залежно від використо-вуємого методу шифрування	Залежно від використо-вуємого методу шифрування	Double-ratchet	Double-ratchet
Відкритість протоколу / імплементаці	Відкриті стандарти	Відкириті стандарти	Відкрий стандарт та відкриті імплементації	Відкрий стандарт та відкрита імплементація
Інтероперабельність	Лише між імплемента-ціями	Лише між імплемента-ціями	Повна, підтримує мости	Відсутня
Процес комунікації серверів різних надавачів послуг	SMTP	RFC 6120	Синхронізація подій за власним стандартом	Відсутня
Протокол	TCP	TCP	HTTPS поверх TCP	Websockets, HTTPS поверх TCP
Формат повідомлень	MIME	XML	JSON	Особливий
Групові чати	Не підтримується	ПІдтримуєтьс я, XEP-0045	Підтримується, Megolm	Підтримується
Усунення конфліктів між серверами	Нема	XEP-0198	Вирішує за допомогою графу подій	Не федерується
Захищеність	Потребує додаткових методів захисту	При виконанні стандартних	Достатня	Достатня
Аутентифікація	Залежить від постачальника	SASL, OAuth	OpenID connect, HMAC	HMAC-SHA256

1.5 Аналіз системи обміну повідомлень Signal

Signal використовує наскрізне шифрування на основі власного Signal Protocol, який вважається одним з найнадійніших у світі. Центральною частиною протоколу є алгоритм Double Ratchet (рисунок 1.1), який динамічно оновлює ключі після кожного повідомлення. Це забезпечує перфектну пряму секретність (PFS), тобто навіть у разі компрометації ключів неможливо розшифрувати попередні повідомлення. [33]

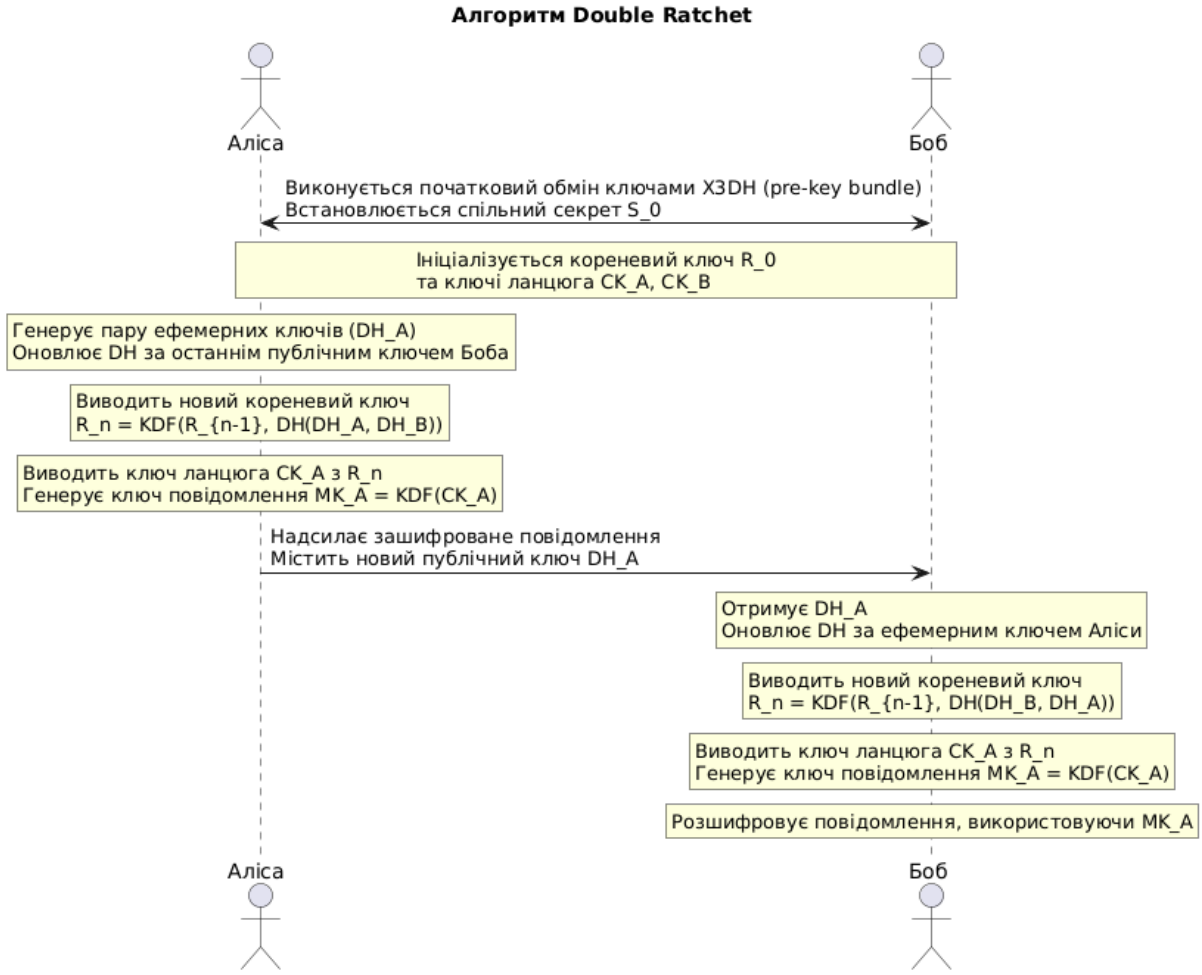


Рисунок 1.1 – Алгоритм Double Ratchet

Початковий обмін ключами між двома сторонами здійснюється через алгоритм X3DH (рисунок 1.2), із використанням попередніх ключів, зберігаючи високий рівень захисту навіть за умов активного перехоплення даних. [34]

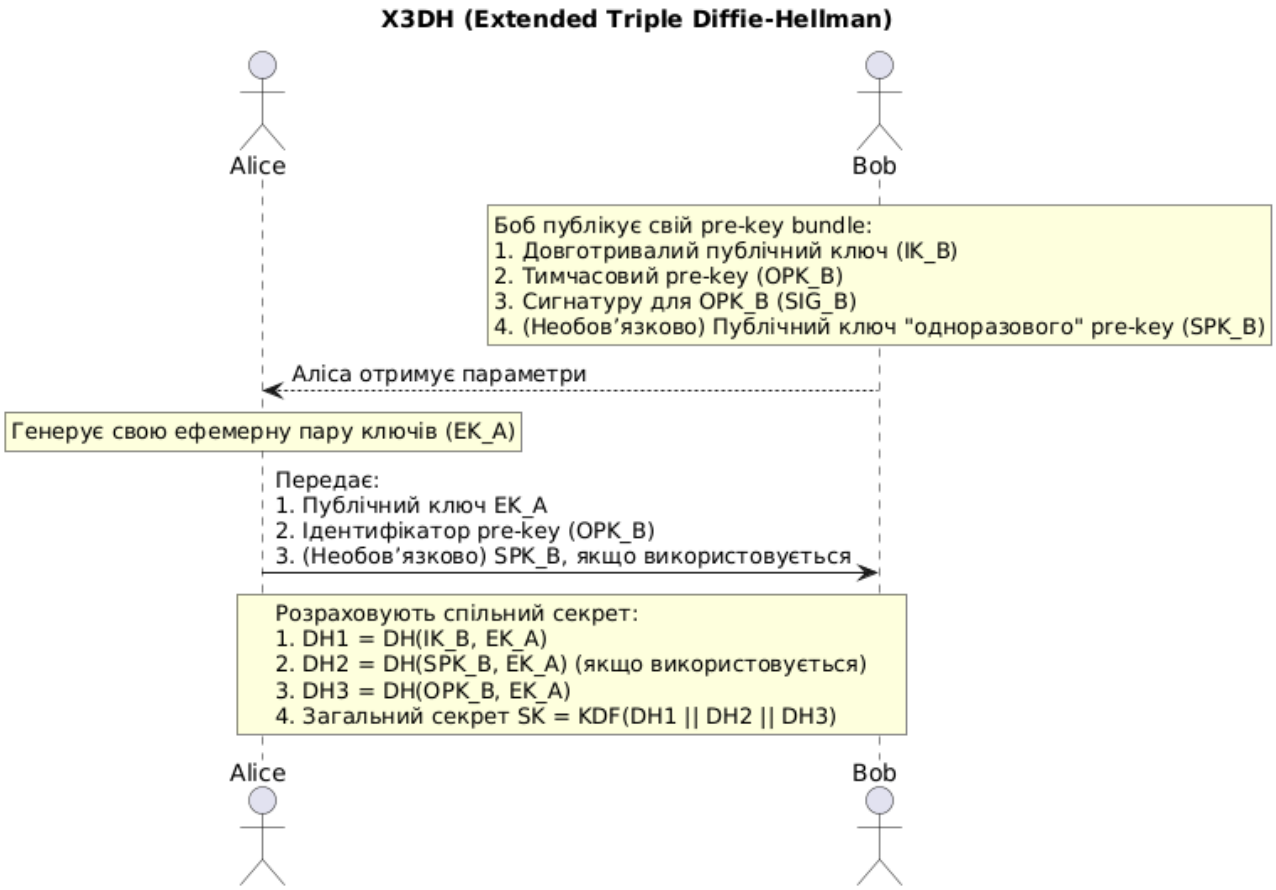


Рисунок 1.2 – Обмін ключами X3DH

Прозорість протоколу є важливою перевагою Signal: його криптографічні алгоритми та код клієнтських додатків є повністю відкритими. Це дозволяє незалежним експертам регулярно проводити аудити безпеки, виявляти потенційні вразливості та гарантувати, що протокол відповідає найсуворішим стандартам. [35]

У групових чатах реалізовано систему шифрування схожу на персональні, однак із модифікаціями, спрямованими на підвищення продуктивності. Групи використовують спільний симетричний ключ для шифрування повідомлень, який періодично оновлюється для підтримання безпеки. Всі дані про структуру групи, такі як список учасників або ролі, зберігаються у зашифрованому вигляді. Синхронізація ключів відбувається між усіма пристроями учасників автоматично, що дозволяє ефективно підтримувати актуальний стан групи навіть при великій кількості учасників. Уся інформація про групу, включаючи

список учасників і ролі адміністраторів, зберігається у зашифрованому вигляді. Синхронізація стану повідомлень і учасників груп здійснюється автоматично, забезпечуючи актуальність даних на всіх пристроях учасників. Навіть у великих групах, що налічують сотні користувачів, Signal зберігає високу продуктивність та рівень захисту. [36]

З точки зору інфраструктури, Signal є централізованою розподіленою системою. Сервіс принципово уникає підтримки інтероперабельності з іншими платформами обміну повідомленнями. Це рішення обумовлене прагненням досягти максимально високого рівня безпеки, що було б складно забезпечити у разі взаємодії з системами, які не мають аналогічних стандартів шифрування чи моделей обробки даних. На відміну від федеративних або інтегрованих рішень, таких як Matrix, Signal працює як закрита екосистема. Це означає, що всі операції — від обміну ключами до пересилання повідомлень — повністю контролюються централізованою інфраструктурою серверів Signal. Відсутність мостів або протоколів для взаємодії з іншими платформами дозволяє уникнути можливих вразливостей, які могли б з’явитися через необхідність узгодження різних криптографічних стандартів. [37]

Сервери розроблені за принципом мінімізації метаданих: вони не зберігають жодної інформації про відправника, отримувача або зміст повідомлення. Це, разом із динамічним шифруванням, гарантує, що навіть у разі компрометації серверів конфіденційність даних залишиться непорушною.[38]

Signal активно захищає своїх користувачів від численних загроз, включаючи атаки типу "людина посередині" (MITM), DoS, спуфінг та перехоплення повідомлень. Система ідентифікації ключів дозволяє користувачам перевіряти автентичність співрозмовників, уникаючи атак, пов'язаних із підміною ідентифікаторів. [39]

Ідентифікатором користувача у Signal базується на телефонних номерах, що є одночасно спрощенням і потенційним ризиком для конфіденційності. Для зменшення ризиків Signal впровадив додаткові рівні захисту, включаючи PIN-

код та функцію "Блокування реєстрації", яка унеможливлює доступ до облікового запису без підтвердження попереднього власника. [39]

Таким чином, Signal демонструє зразкове поєднання безпеки, прозорості та функціональності, що робить його одним із найкращих інструментів для конфіденційного спілкування.

1.6 Аналіз системи обміну повідомлень Email

Електронна пошта за замовчуванням не забезпечує шифрування вмісту, що означає, що листи можуть бути прочитані на серверах чи під час передачі, якщо не застосовуються додаткові заходи. Для забезпечення шифрування зазвичай використовують PGP (Pretty Good Privacy) і S/MIME (Secure/Multipurpose Internet Mail Extensions), які реалізують наскрізне шифрування (E2EE). [12][40]

PGP працює на основі асиметричної криптографії,. Відправник шифрує повідомлення відкритим ключем отримувача, тільки власник відповідного приватного ключа може його розшифрувати. PGP дозволяє зберігати ключі на локальному пристрої, використовуючи менеджери ключів (наприклад, GnuPG), також дає можливість публікувати відкриті ключі на сервера ключів. Однак метадані (адреси відправника та отримувача, тема листа) залишаються не шифрованими, що може загрожувати конфіденційності. [12]

S/MIME, натомість, покладається на цифрові сертифікати, які видаються сертифікаційними центрами (CA). Сертифікати містять інформацію про відкритий ключ користувача і підтверджують його автентичність. Шифрування здійснюється аналогічно PGP, але S/MIME більше використовується в корпоративному середовищі, оскільки спрощує управління ключами та сертифікатами через централізовану довіру до CA. Як і у випадку з PGP, S/MIME зазвичай не шифрує метадані. [40]

Сервіси, такі як ProtonMail, впроваджують повне наскрізне шифрування за замовчуванням. Тобто, повідомлення шифруються на пристрої відправника і

можуть бути розшифровані лише на пристрої отримувача. ProtonMail також шифрує метадані, наскільки це можливо. Хоча деякі аспекти, як-от адреси електронної пошти, можуть залишатися доступними заради маршрутизації. Обмеження таких сервісів полягають у тому, що наскрізне шифрування працює повноцінно лише між користувачами одного сервіс. [13]

Для листування з іншими поштовими клієнтами потрібне використання спільних стандартів. Таким чином, PGP і S/MIME є інструментами для захисту вмісту електронної пошти, але вони не забезпечують повного захисту метаданих, тоді як сервіси з E2EE, як ProtonMail, намагаються зменшити ці недоліки, хоч і залишаються залежними від специфіки своїх платформ.

У PGP і S/MIME ключі генеруються користувачем або автоматично створюються поштовим клієнтом чи сервісом. У PGP ключі створюються у вигляді пари: відкритий ключ для шифрування і приватний ключ для дешифрування. Користувач зберігає приватний ключ локально, а відкритий ключ поширює серед своїх контактів. Для поширення відкритого ключа можуть використовуватися публічні сервери ключів, наприклад, OpenPGP Keyserver, також він може передаватися вручну (через email, файл чи QR-код). У системі PGP також існує механізм довіри, так зване "веб довіри", де ваш ключ можуть підписувати інші користувачі, що підтверджує його автентичність. [41]

У S/MIME ключі базуються на сертифікатах, , наприклад, DigiCert чи GlobalSign. Для генерації сертифікату користувач подає заявку до CA, після чого йому видається сертифікат, що містить його відкритий ключ і інформацію про власника. Приватний ключ також створюється на стороні користувача й зазвичай зберігається в локальному сховищі поштового клієнта (наприклад, Outlook чи Thunderbird). Автентичність ключів S/MIME гарантується через ієрархію довіри до CA, а не через "веб довіри", як у PGP. [40]

У сучасних системах використовуються автоматизовані процеси для зручності. Наприклад, сервіси, як ProtonMail, автоматично обмінюються відкритими ключами між користувачами сервісу, спрощуючи процес. У корпоративних середовищах можливий обмін ключами через сервери каталогів,

наприклад, Active Directory Certificate Services. [42] Сучасні поштові клієнти інтегрують підтримку обміну ключами (наприклад, Thunderbird з плагіном Enigmail), дозволяючи автоматично завантажувати та перевіряти ключі. Якщо ключ передається незахищеними каналами (наприклад, через email), його можуть перехопити зловмисники. Рекомендується використовувати додаткові шифровані канали (наприклад, Signal). Важливо переконатися, що джерело ключа достовірне, щоб уникнути підробки.

Протоколи електронної пошти, такі як SMTP (для відправки), IMAP і POP3 (для отримання), є відкритими стандартами. Це сприяє їхньому поширенню та можливості аудиту на безпечність, але також створює можливість атак, якщо реалізація протоколу не захищена належним чином. [43][44]

Електронна пошта забезпечує високу інтероперабельність: Користувачі різних сервісів (Gmail, Outlook, Yahoo тощо) можуть легко обмінюватися повідомленнями. Багато CRM-системам, месенджерів та інших застосунків сервісів мають системи для взаємодії з електронною поштою.

Сервери обмінюються повідомленнями через протокол SMTP поверх TCP, який відповідає за: Доставку листів між поштовими серверами. Повідомлення про статус доставки (повідомлення про помилки, недоставлені листи). Сервери знаходяться по DNS записам. Основним стандартом стандарт повідомлень є MIME (Multipurpose Internet Mail Extensions), який дозволяє включати текст, HTML, зображення, аудіо та відео.Повідомлення також можуть бути у форматі простого тексту або HTML (наприклад, для листів із багатим форматуванням). [45][46]

Групові чати не є частиною стандартного функціоналу електронної пошти, але вони підтримуються через розсилки (mailing lists). Використовуються спеціальні сервіси, як Mailman. Учасники отримують всі повідомлення, надіслані до списку, але немає такої гнучкості, як у більш новітніх системах. [47]

Коли користувач відправляє листа, його клієнт передає повідомлення до SMTP-сервера, який доставляє його до сервера одержувача. Тут немає

одночасної конкуренції кількох вузлів за зміни однієї і тієї ж сутності. Кожен лист — це ізольована транзакція. Тому проблем несумісного стану декількох клієнтів серверів не виникає. [12]

Система електронної пошти має міри захисту, але залишається вразливою: SPF (Sender Policy Framework), DKIM (DomainKeys Identified Mail), DMARC (Domain-based Message Authentication, Reporting & Conformance) використовуються для боротьби зі спуфингом. [24][25][26]

Електронна пошта залишається базовою системою комунікації, але для забезпечення конфіденційності та безпеки користувачам часто доводиться використовувати сторонні засоби або додаткові протоколи.

1.7 Аналіз систем обміну повідомлень XMPP

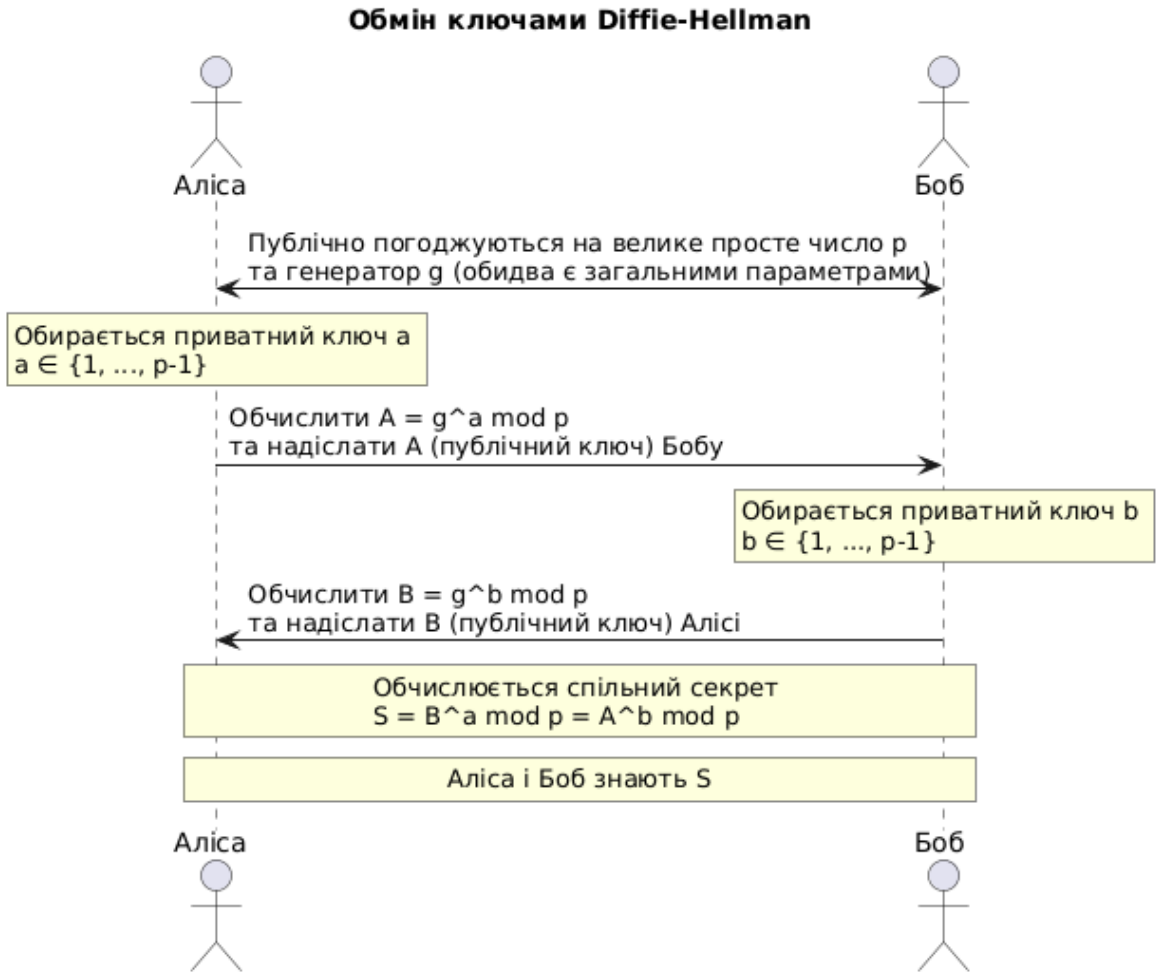
XMPP (Extensible Messaging and Presence Protocol) — це протокол, заснований на XML, для обміну повідомленнями, присутності і організації реального часу.

У XMPP використовуються кілька підходів до шифрування повідомлень, залежно від налаштувань і розширень протоколу. OMEMO (XEP-0384) це сучасний стандарт наскрізного шифрування для XMPP, який базується на Double Ratchet Algorithm (такий же, як у Signal). OMEMO забезпечує конфіденційність і прямий секрет завдяки динамічному обміну ключами, а також підтримує групові чати та синхронізацію між пристроями. OTR (Off-the-Record) старіший метод наскрізного шифрування, який забезпечує конфіденційність та аутентифікацію повідомлень. OTR добре підходить для одноразових розмов, але не підтримує історію або синхронізацію між пристроями. XMPP також може використовувати PGP (XEP-0027) для шифрування повідомлень. [16][17][18]

При використанні стандарту OMEMO обмін ключами виконується через алгоритм Double Ratchet, який автоматично обробляє обмін ключами між учасниками. Інші методи, наприклад OpenPGP, вимагають більше ручних дій з

боку користувачів, як в прикладі Email. OTR обмін ключами відбувається безпосередньо між учасниками сеансу через встановлене з'єднання. OTR генерує ключі для кожного сеансу, і вони не зберігаються для повторного використання, що можна розцінити як певну міру захисту. Однак це означає, що для кожної нової розмови ключі потрібно обмінювати заново. Спочатку ініціатор починає сесію, генеруючи тимчасову пару ключів (відкритий і закритий). Ця пара використовується для встановлення захищеного каналу з іншою стороною. [16][18]

Далі за допомогою алгоритму Diffie-Hellman (рисунок 1.3), генерується спільний секретний ключ, який дозволяє шифрувати повідомлення. Ключ генерується динамічно на основі унікальних параметрів, що ускладнює його перехоплення третіми сторонами. Завдяки цим механізмам OTR забезпечує конфіденційність і одноразовість ключів для кожної сесії.



исунок 1.3 – Обмін ключами Diffie-Hellman

XMPP є відкритим стандартом, специфікація якого розробляється та підтримується організацією IETF (Internet Engineering Task Force). Завдяки цьому протокол може бути вільно реалізований у будь-якому програмному забезпеченні без ліцензійних обмежень.

Сервери, що підтримують цей протокол, можуть взаємодіяти між собою за принципом федерації, дозволяючи користувачам різних платформ комунікувати без обмежень. Це дозволяє створювати децентралізовану мережу з розподіленими вузлами, забезпечуючи стійкість до збоїв і масштабованість.У XMPP сервери комунікують між собою за стандартами, визначеними протоколом. Основним стандартом є RFC 6120, який описує основи маршрутизації, встановлення з’єднань і шифрування. [48]

Для обміну повідомленнями сервери використовують потоки XML коду поверх TCP, де кожне повідомлення або присутність користувача пакуються у вигляді XML-стенза. RFC 6121 – описує обробку миттєвих повідомлень і

статусу присутності між клієнтами та серверами, а також між вузлами серверів. XMPP використовує записи DNS SRV для виявлення інших серверів у федеративній мережі. SASL (RFC 4422) – забезпечує автентифікацію для встановлення довіреного з'єднання між вузлами серверів. Він дозволяє використовувати паролі, токени або сертифікати. Також підтримується двофакторна аутентифікація та інші методи для підвищення безпеки. XEP-0368 – розширення для XMPP, яке підтримує використання TLS поверх протоколу DIRECT TCP або WebSocket. [6][48][49][50]

Федеративна модель дозволяє серверам встановлювати з’єднання один з одним, передаючи повідомлення між доменами через безпечні канали, а стандарти забезпечують сумісність і надійність роботи мережі. XMPP використовує XML як основний формат повідомлень. Ця структура дозволяє легко розширювати функціонал протоколу за рахунок додавання нових тегів і елементів. Однак це також додає складності та може впливати на продуктивність. [6]

Групові чати у XMPP реалізовані через Multi-User Chat (MUC, XEP-0045). Це дозволяє створювати чати з великою кількістю учасників, визначати ролі (адміністратор, модератор, учасник), синхронізувати історію повідомлень і підтримувати налаштування приватності. [51]

XMPP вирішує проблему конкурентних подій через механізми унікальних ідентифікаторів повідомлень, підтвердження доставки (XEP-0198). Це забезпечує консистентність даних навіть в умовах затримок або втрати з'єднання. [52]

XMPP має вбудовані механізми захисту від атак, таких як спуфінг і перехоплення повідомлень, завдяки обов’язковому використанню TLS та шифруванню. Однак стійкість до інших залежить від реалізації конкретного сервера, а не лише протоколу. Це ж можна висловити до більшості інших функціональних розширень, все залежить від реалізації сервера і клієнта, наприклад OMEMO залишається в стані експериментального розширення, тому

певні користувачі не можуть ним користуватися і не отримують усієї можливої безпеки.

1.8. Аналіз системи обміну повідомлень Matrix

Matrix використовує надійну криптографічну систему, яка спирається на передові алгоритми для забезпечення наскрізного шифрування (E2EE) та перевірки особи користувача. Два основні криптографічні алгоритми, що використовуються - Olm та Megolm. Olm, заснований на алгоритмі Double Ratchet, забезпечує безпечний зв'язок між окремими пристроями за допомогою асиметричного шифрування та обміну ключами Diffie-Hellman. Megolm, призначений для групового спілкування, - це симетричний протокол шифрування, оптимізований для ефективності та масштабованості, де для захисту повідомлень, якими обмінюються в межах групи або кімнати, використовується один ключ шифрування. [52]

Обробка обміну ключами в Matrix є фундаментальною для його криптографічної архітектури. Щоб встановити безпечний зв'язок, пристрої використовують Olm для обміну ключами сеансу через зашифровані повідомлення між пристроями. Після встановлення сеансу зв'язку за справу береться протокол Megolm, який використовує ключ сеансу для шифрування та розшифрування повідомлень. Цей сеансовий ключ повинен бути спільним для всіх пристроїв, яким потрібен доступ до зашифрованих повідомлень, включаючи пристрої, що належать відправнику і передбачуваним одержувачам. Обмін сеансовими ключами Megolm захищений за допомогою Olm-зашифрованих повідомлень, що гарантує доступ до сеансу тільки авторизованим пристроям. Матриця також дозволяє створювати резервні копії ключів, де сеансові ключі можуть бути зашифровані та збережені на сервері для відновлення, але доступні тільки якщо користувач володіє приватним ключем, пов'язаним з системою резервного копіювання. Цей механізм гарантує, що навіть якщо користувач втрачає доступ до одного пристрою, його здатність

розшифровувати минулі комунікації залишається недоторканою, якщо він зберігає свої облікові дані для резервного копіювання ключів. [52]

Пристрої в Matrix - це окремі клієнти, такі як смартфони, ноутбуки або планшети, кожен з яких представлений унікальною парою криптографічних ключів. Кожен пристрій, пов'язаний з обліковим записом Matrix, генерує власну пару ключів, яка складається з приватного ключа, що зберігається на пристрої, і публічного ключа, який передається на сервер для ідентифікації та верифікації. Для забезпечення довіри між пристроями Matrix використовує багаторівневу систему верифікації. Пристрої підписують відкриті ключі один одного за допомогою самопідписуючих ключів і майстер-ключів, створюючи ланцюжок довіри, який дозволяє користувачам перевіряти власні пристрої і встановлювати довірчі відносини з іншими користувачами. Ці криптографічні підписи є основою для визначення того, чи є пристрій автентичним і чи має він дозвіл на доступ до зашифрованих даних. [52]

Поєднуючи Olm для безпечного обміну ключами та Megolm для ефективного групового шифрування, Matrix досягає балансу між безпекою та зручністю використання. Її криптографічний дизайн підкреслює децентралізовану довіру і контроль користувача, дозволяючи користувачам перевіряти і керувати своїми пристроями, захищаючи при цьому свої комунікації від несанкціонованого доступу. Ця система ілюструє складну взаємодію симетричного та асиметричного шифрування, гарантуючи, що шифрування залишається одночасно безпечним і масштабованим у федеративній мережі.

Протокол Matrix є повністю відкритим. Код базового серверного програмного забезпечення Synapse, клієнтів і бібліотек доступний у відкритому доступі. Matrix підтримує високу інтероперабельність. За допомогою мостів він може інтегруватися з іншими платформами, такими як Slack, Telegram, WhatsApp, і навіть електронною поштою. Це робить його універсальним рішенням для комунікації між різними екосистемами. [53]

У Matrix зв'язок між серверами –- це фундаментальна основа

федеративної архітектури, що дозволяє серверам взаємодіяти і синхронізувати дані в децентралізованій мережі. Матриця обробляє цей зв'язок за допомогою протоколу Matrix, на який посилаються сервери пти обміні повідомленнями, подіями та даними між собою. Тіла подій вважаються ненадійними даними. Це означає, що будь-яка програма, яка використовує Matrix, повинна перевірити, що тіло події має очікувану форму/схему, перш ніж дослівно використовувати його вміст. Сервери матриці обмінюються двома категоріями даних: PDU та EDU. [54]

PDU (Protocol Data Unit): PDU - позначає подію в протоколі Matrix. Це фрагмент даних, наприклад, повідомлення або зміна стану, який надсилається між серверами Matrix або між клієнтами та серверами. PDU використовуються для синхронізації кімнат, надсилання повідомлень та оновлення станів кімнат. Дані події включають її тип, відправника, мітку часу, кімнату та вміст. PDU поширюються по об'єднаній мережі таким чином, що всі відповідні сервери оновлюють свій стан відповідно.

EDU (Extensible Data Unit - розширюваний блок даних): EDU - це більш легкий і гнучкий тип комунікаційної одиниці, що використовується для спеціальних взаємодій. На відміну від PDU, які є основними подіями, що представляють зміни стану кімнати або повідомлення, EDU зазвичай використовуються для позасмугового зв'язку, наприклад, для синхронізації інформації про присутність, індикаторів введення та інших метаданих. EDU дозволяють здійснювати асинхронний зв'язок поза основним потоком подій, що робить їх корисними для таких речей, як відстеження активності користувачів, але вони не несуть повної ваги подій, як PDU.

Формат повідомлень у Matrix — JSON, що робить його легко читабельним, стандартизованим і зручним для інтеграції з іншими системами. JSON дозволяє швидко обробляти повідомлення та зберігати метадані. Комунікаційним рівнем виступає HTTP. Основною причиною використання HTTP як базового протоколу є його широке розповсюдження та сумісність. Це добре встановлений стандарт з численними реалізаціями на різних платформах і

мовах програмування. Така широка підтримка забезпечує сумісність між різними системами та пристроями. Хоча новіші протоколи, такі як WebSockets, CoAP і HTTP/2, пропонують потенційні переваги з точки зору ефективності та зв'язку в реальному часі, вони можуть не мати такої широкої підтримки або бути менш розвинутими, як HTTP. Це може призвести до проблем із сумісністю та підвищеної складності у впровадженні. [55]

Matrix добре справляється з проблемою конфліктів подій завдяки використанню системи з унікальними ідентифікаторами та структури даних DAG (Directed Acyclic Graph). Це дозволяє зберігати консистентність станів навіть при різночасному надходженні повідомлень. Алгоритм визначення стану залежить від версії кімнати. За останньою версією на момент написання кваліфікаційної роботи механізм наступний: Спочатку обирається множина X конфліктних подій, зосереджуючись на power- подіях (подіях зі значним впливом, таких як зміна членства). Для кожної power-події P вона розширює X, додаючи до неї події з ланцюжка авторизації P (події, що уповноважили P), які також належать до конфліктної множини. Далі події в X сортуються за допомогою зворотного топологічного впорядкування, гарантуючи, що залежні події розглядаються першими. Потім до відсортованого списку застосовується алгоритм ітеративних перевірок автентичності, починаючи з неконфліктної моделі станів. Це генерує частково вирішений стан. Решта подій, які не були включені до кроку 1, впорядковуються на основі основного порядку, який враховує історію подій та рівні повноважень. До цих подій знову застосовуються ітеративні перевірки автентичності. Нарешті, будь-які події з моделі неконфліктних станів, які відповідають ключу невирішеної події, замінюють конфліктну подію, створюючи остаточний вирішений стан. [56]

Спочатку Matrix використовувала власну систему аутентифікації, де користувачі проходили автентифікацію за допомогою імені користувача та пароля безпосередньо на домашньому сервері Matrix. Потім домашній сервер обробляв ідентифікацію та авторизацію користувача в мережі. Однак Matrix

перейшла на використання OpenID Connect (OIDC) для автентифікації з кількох причин. [57]

Однією з головних мотивацій була стандартизація та інтероперабельність. OpenID Connect є відкритим і широко прийнятим стандартом для автентифікації, цей перехід спростив для Matrix взаємодію з іншими сервісами та системами, які вже підтримували OIDC. Іншою ключовою причиною було покращення безпеки. OpenID Connect пропонує більш надійний і безпечний механізм аутентифікації, включаючи підтримку таких функцій, як багатофакторна автентифікація (MFA), яку було б складніше впроваджувати і підтримувати в оригінальній системі Matrix. Крім того, OIDC дозволяє децентралізувати автентифікацію, тобто домашні користувачі можуть делегувати автентифікацію окремим спеціалізованим постачальникам ідентичностей, а не керувати всім процесом автентифікації самостійно.

Така децентралізація дозволяє домашнім користувачам покладатися на надійних постачальників, якими вони вже можуть користуватися, зменшуючи навантаження на підтримку інфраструктури автентифікації.

1.9 Висновки до розділу

Федеративні системи обміну повідомленнями є перспективною моделлю для майбутнього розвитку комунікаційних платформ, що відкриває нові можливості в децентралізації та безпеці. Основою їхньої архітектури є здатність різних вузлів співпрацювати, зберігаючи автономію, що дозволяє уникати залежності від єдиної централізованої інфраструктури. Цей підхід повинен спиратися на відкриті стандарти, які забезпечують узгодженість і сумісність, що зокрема відображено в розглянутих нами XMPP та Matrix. Вони не лише сприяють підвищенню стійкості до збоїв, але й дозволяють створювати інклюзивні екосистеми, де користувачі з різних платформ можуть взаємодіяти без перешкод.

Однією з ключових рис федеративних систем є їхня здатність до інтероперабельності. Завдяки відкритим стандартам та протокольним мостам, ці системи дозволяють інтегрувати розрізнені мережі, розширюючи межі спілкування. Наприклад, Matrix створює мости між Slack, Telegram та іншими платформами, забезпечуючи безперервність комунікації в умовах різноманітності протоколів. Така стратегія може стати основою для подолання фрагментації цифрового середовища, створюючи єдину екосистему, де користувачі з різних платформ можуть обмінюватися даними, залишаючись у межах своїх локальних серверів.

Одним із перспективних напрямів розвитку таких систем є нові алгоритми та механізми, які гарантуватимуть високий рівень доступності навіть за умов серверних збоїв чи інтермітуючих з'єднань. Наприклад, використання дистрибуції повідомлень через декілька реплік серверів або застосування механізмів пошуку альтернативних шляхів для доставки даних може значно підвищити надійність систем. Іншим з напрямків розвитку є полегшення користувацького досвіду, що включає розробку зручних інтерфейсів як для адміністраторів, так і для користувачів, так і полегшення. Це може включати створення інтуїтивно зрозумілих панелей управління, які спрощують процес налаштування федеративної системи. Крім того, впровадження інтелектуальних помічників для автоматичного налаштування конфігурацій та надання підказок щодо покращення безпеки або ефективності системи дозволить користувачам і адміністраторам оперативно адаптувати систему до своїх потреб без значних зусиль.

Спільним для федеративних систем є зосередженість на принципах відкритості. Відкритий вихідний код та прозорість стандартів не лише підвищують довіру користувачів, але й стимулюють інновації за рахунок участі спільнот у вдосконаленні інфраструктури. Практики, що вже знайшли втілення у федеративних системах, можуть бути перенесені й до централізованих платформ. Наприклад, принципи мінімізації метаданих, застосовувані у Signal, чи синхронізація даних через розподілені графи подій у Matrix можуть

допомогти централізованим системам підвищити безпеку та надійність. У свою чергу, підходи до інтеграції з іншими платформами, що використовуються в Matrix, можуть стати моделлю для централізованих платформ, які прагнуть створити більш відкриті екосистеми.

Таким чином, федеративні системи, з одного боку, закладають основи для нового рівня децентралізації, а з іншого — підштовхують усю галузь до переосмислення ролі відкритості, безпеки та інтероперабельності в сучасних комунікаціях. Їхній потенціал лежить у здатності об'єднувати, зберігаючи різноманіття, і створювати платформи, які не тільки відповідають сучасним викликам, але й формують майбутнє цифрового спілкування.

РОЗДІЛ 2
ПЛАНУВАННЯ ТА ОРГАНІЗАЦІЯ ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1. Опис використовуємої методології

Методологія Use Case Points (UCP) є одним із найбільш ефективних інструментів для оцінки складності та обсягу робіт у проектах розробки програмного забезпечення. Цей підхід базується на аналізі варіантів використання (use cases), які детально описують взаємодію користувачів із системою для досягнення конкретних цілей, що дозволяє формувати точне уявлення про функціональність та інтерфейси програмного продукту. У випадку розробки складних інформаційних систем, методологія UCP дозволяє на ранніх етапах планування точно визначити обсяг робіт та необхідні ресурси, що є важливим для забезпечення своєчасної реалізації проекту та управління ризиками. [58]

Вибір методології UCP для планування і оцінки часу, необхідного для реалізації системи, зумовлений її здатністю не тільки враховувати функціональні вимоги, але й інтегрувати оцінку технічної складності проекту, що є особливо важливим для проектів з великою кількістю компонентів та складними взаємодіями між ними. Оскільки кожна частина системи може впливати на загальну продуктивність та стабільність, використання UCP дозволяє отримати більш точну оцінку складності на рівні окремих функцій та їх інтеграції.

Ключовою перевагою методу UCP є його здатність враховувати не лише технічні аспекти, а й зовнішні фактори, які можуть вплинути на проект. Це включає складність взаємодії з іншими сервісами, вимоги щодо безпеки, масштабованості, гнучкості та інтеграції. Зокрема, оцінка безпеки дозволяє передбачити додаткові витрати часу та ресурсів на реалізацію захищених каналів зв’язку та управління доступом, що особливо важливо в умовах

сучасних кіберзагроз. Масштабованість та гнучкість системи враховуються для оцінки можливостей її розширення та адаптації до нових умов чи змін вимог, що підвищує точність планування.

Цей підхід дозволяє детально прогнозувати не лише загальний час розробки, а й ідентифікувати потенційні складнощі на різних етапах розробки проекту. Оцінка складності на початкових етапах допомагає виявити найбільш проблемні ділянки, що дозволяє розробити ефективні стратегії для мінімізації ризиків та забезпечення належного контролю над процесом реалізації. Завдяки цьому, використання UCP вносить значний вклад у точність прогнозування та ефективність управління проектом на всіх його етапах, від планування до завершення.

2.2. Визначення функціональних вимог системи

Система розробляється для забезпечення ефективного та безпечного обміну повідомленнями між користувачами в асинхронному та синхронному режимах з використанням наскрізного шифрування, синхронізацією даних та зручним управлінням історією повідомлень. Вона складається з таких основних компонентів (рисунок 2.1):

- Користувача
- Модулю обміну ключами (InviteModule)
- Модуля обміну повідомленнями (MessagesModule)
- Модуля наскрізного шифрування (EncryptionModule)
- Модуля синхронізації даних (SyncModule)
- Модуля для міжсерверної комунікації (NodeModule).

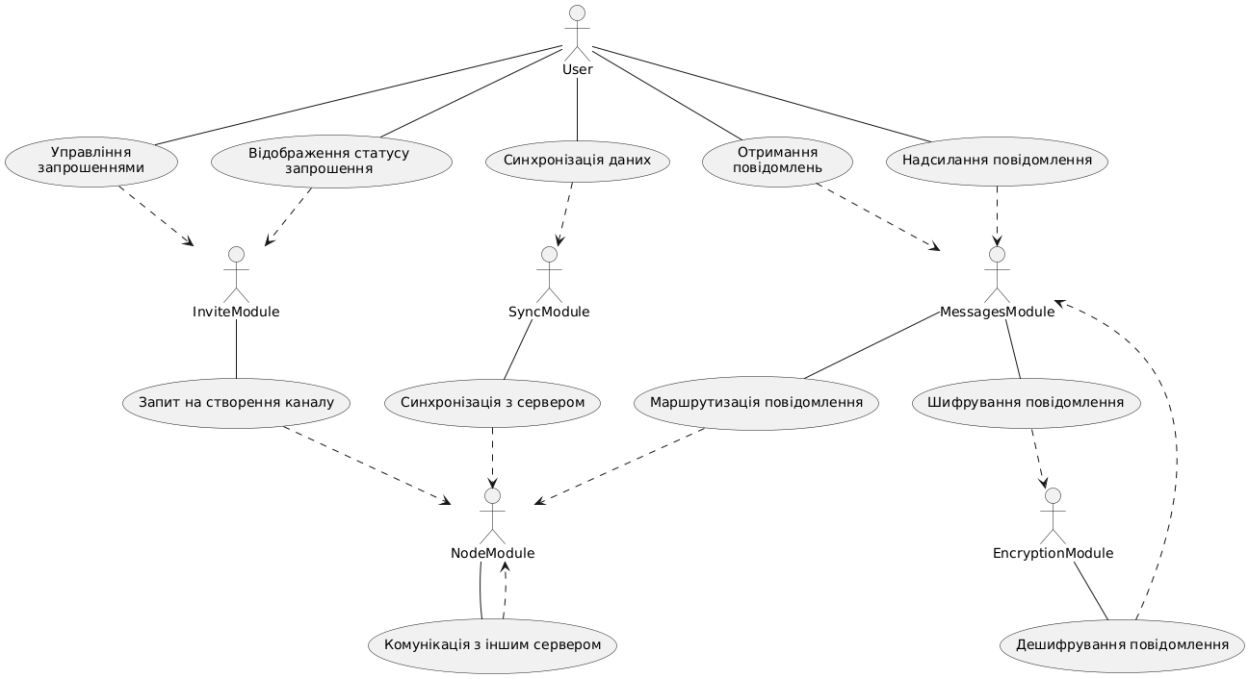


Рисунок 2.1 – Модель акторів

Користувач: є центральною частиною системи, виконуючи основні дії, такі як реєстрація, надсилання та отримання повідомлень, редагування чи видалення надісланих даних, переглядати історію повідомлень, створювати нові канали зв’язку. Користувач взаємодіє з іншими компонентами через зрозумілий інтерфейс, що забезпечує доступність усіх функцій.

MessagesModule: відповідає за управління повідомленнями. Він забезпечує миттєву доставку повідомлень із мінімальною затримкою.

EncryptionModule: забезпечує наскрізне шифрування повідомлень для збереження конфіденційності користувацьких даних. Цей модуль виконує шифрування на стороні клієнта, щоб сервери не мали доступу до вмісту повідомлень. Ключі шифрування генеруються та зберігаються виключно на пристроях користувачів, гарантуючи захист інформації навіть у разі компрометації серверів.

SyncModule: виконує синхронізацію даних між серверами та пристроями користувачів у режимі реального часу. Він дозволяє отримувати нові повідомлення та оновлення на будь-якому з пристроїв, на якому користувач

авторизований. У разі підключення нового пристрою або відновлення доступу історія повідомлень синхронізується повністю.

NodeModule: надає можливість серверам комунікувати між собою та синхронізувати стан, пересилати повідомлення користувачів та запити на встановлення каналу, якщо користувачі знаходяться на різних серверах

Користувач:

- Реєстрація/Авторизація в системі.
- Надсилання/Отримання повідомлень.
- Перегляд історії повідомлень.
- Ініціація запиту на створення нових каналів зв’язку.
- Управління запрошеннями для додавання до каналів зв’язку.
- Генерація ключів шифрування

InviteModule:

- Ініціація запиту на створення нових каналів зв’язку.
- Відображення статусу запрошення
- Управління запрошеннями для додавання до каналів зв’язку.

MessagesModule:

- Надсилання/Отримання повідомлень.
- Збереження статусу доставки та прочитання.
- Управління чергами повідомлень у разі тимчасової недоступності.

EncryptionModule:

- Генерація ключів шифрування
- Виконання наскрізного шифрування повідомлень.
- Захист конфіденційності ключів шифрування.

SyncModule:

- Синхронізація даних між пристроями користувача.
- Завантаження історії повідомлень на нових пристроях.
- Обробка конфліктів синхронізації даних.
- NodeModule:
- Комунікація між серверами для маршрутизації повідомлень.

- Синхронізація стану каналів зв’язку.
- Маршрутизація запитів між користувачами, які знаходяться на різних серверах.

Надсилання повідомлення:

Гарантія успіху: користувач успішно надіслав повідомлення. Тригер: користувач обрав контакт або канал для надсилання. Передумова: користувач зареєстрований та авторизований.

Основний сценарій:

1. Система запитує текст повідомлення.
2. Користувач вводить текст та натискає "Надіслати".
3. Система шифрує повідомлення, передає його через сервери, підтверджує доставку.

Розширення:

2а* Користувач залишив поле повідомлення порожнім.

2а.1. Система виводить повідомлення про помилку.

2а.2. Користувач вводить текст, перехід до П2.

Перегляд історії повідомлень

Гарантія успіху: користувач успішно переглянув історію повідомлень. Тригер: користувач відкрив вкладку історії повідомлень. Передумова: користувач авторизований у системі.

Основний сценарій:

1. Система запитує історію повідомлень із сервера або локальної копії.
2. Відображає повідомлення у зручному форматі.

Розширення:

2а* Дані недоступні через збій синхронізації.

2а.1. Система інформує про помилку та намагається синхронізувати дані повторно.

Встановлення нового каналу зв'язку

Гарантія успіху: канал зв'язку створено між користувачами.

Тригер: користувач ініціює створення каналу.

Передумова: користувачі зареєстровані в системі.

Основний сценарій:

1. Користувач ініціює запит на запрошення іншого користувача
2. Система надсилає запрошення іншому користувачеві, навіть якщо він знаходиться на іншому сервері.
3. Інший користувач приймає запрошення.
4. Користувачі отримують спільні ключі

Розширення:

2a Система не може знайти користувача

2a* Система змінює статус запрошення

2b Система не може зв’язатися з віддаленим сервером

2b* Система відкладає запрошення на певний час

2.3. Виділення нефункціональних вимог системи

Для розробки системи важливо визначити нефункціональні вимоги, оскільки вони забезпечують реалізацію необхідних характеристик системи, таких як продуктивність, безпека та масштабованість, що впливає на ефективність, стабільність та здатність системи задовольняти потреби користувачів у реальних умовах.

Ось кілька основних:

- Інтероперабельність: Система повинна дозволяти різним сервісам і платформам обмінюватися повідомленнями без значних проблем з сумісністю, або бути простою в інтеграції через певні мости.
- Система має підтримувати масштабування для великих обсягів даних і користувачів без втрати продуктивності або доступності.

- Важливим є забезпечення шифрування повідомлень, аутентифікації користувачів, авторизації доступу до інформації та захисту від атак.
- Система повинна гарантувати доставку повідомлень, навіть якщо один з вузлів або підсистем вийшов з ладу.
- Система повинна мати можливість адаптуватися до змін у вимогах, наприклад, додавання нових каналів комунікації або змінення політик обміну повідомленнями.

2.4. Оцінювання тривалості та планування розробки

На цьому етапі здійснюється оцінка часу, необхідного для розробки програмних процесів, спрямованих на підвищення ефективності та швидкості роботи продукту. Актори (таблиця 2.1) поділяються на простих і складних залежно від їх функціонального навантаження. Простий актор має меншу вагу, яка становить 2, тоді як складний актор має вагу 3.

Табл 2.1 – Актори

Актор	Тип актора	Вага
Користувач	Простий	2
InviteModule	Складний	3
MessageModule	Складний	3
EncryptionModule	Простий	2
SyncModule	Складний	3
NodeModule	Складний	3

Розрахунок нескоректованої оцінки акторів (UAW, Unadjusted Actor Weight) базується на підсумовуванні ваг усіх акторів системи. Розрахунок нескоректованої оцінки акторів: UAW = 2 * 2 + 3 * 4 = 16

Некоригована оцінка варіантів використання (UUCP) — метрика, яка використовується для оцінки загальної складності та обсягу роботи, необхідної

для реалізації функціональності програмного забезпечення. Для того щоб порахувати цю метрику необхідно визначити складність кожного варіанту використання (простий, середній, складний), від цієї складності залежить вага, так для простого, середнього та складного ваги відповідно становлять 5, 7, 10. Розрахунок аналогічний як для UAW.

Табл 2.2 – Варіанти використання

Варіант використання	Тип використання	Вага
Надсилання повідомлення	Простий	5
Перегляд історії повідомлень	Простий	5
Встановлення нового каналу зв’язку	Складний	10
Синхронізація даних	Середній	7
Шифрування повідомлень	Середній	7
Обробка міжсерверної комунікації	Складний	10

Розрахована вага нескоректованої оцінки варіантів використання: UUCP = 5 * 2 + 7 * 2 + 10 * 2 = 44.

Завдяки повторному використанню, нові проекти можуть бути реалізовані швидше, що скорочує час розробки продукту. Показник повторного використання коду у нашому проекті становить 60%, це не є випадковим, а базується на аналізі схожиш проектів, нашої архітектури та підходів до розробки.

При оцінці складності нашого проекту за допомогою методики Use Case Points (UCP), важливо враховувати не лише функціональні вимоги, але й технічні аспекти (таблиця 2.3), які можуть впливати на процес розробки. Технічний фактор (Technical Complexity Factor) дозволяє врахувати ці аспекти і скоригувати загальну оцінку проекту відповідно до їхнього впливу. Обчислюється за допомогою врахування 13 технічних факторів, кожен з яких оцінюється за шкалою від 0 до 5

Таблиця 2.3 – Технічні фактори

Фактор	Опис	Вага	Обгрунтування	Оцінка
T1	Розподілена система	2	Висока складність через протоколи федерації та управління розподіленими станами.	5
T2	Низька затримка	3	Високий трафік і вимоги до реального часу вимагають оптимізації паралельності та пропускної здатності.	5
T3	Ефективність для користувача	2	Оновлення в реальному часі та чутливий інтерфейс користувача є критичними для зручності.	4
T4	Внутрішня обробка	1	Вимагає надійного аналізу та обробки форматів, специфічних для федерації.	3
T5	Повторне використання	1	Заохочує довгострокову розширюваність і повторне використання коду	3
T6	Простота розгортання	0.5	Для простоти налаштування потрібні скрипти встановлення	1
T7	Простота використання	1	Зручність важлива для прийняття, але менш критична, ніж механіка федерації.	3
T8	Переносимість	1	Гнучкість розгортання вимагає контейнеризації та інструментів оркестрації.	3
T9	Простота змін	1	Оскільки розробка повинна буде розвиватися, то повинен бути закладений гарний архітектурний фундамент	3
T10	Паралельна обробка	1	Вибрана мова програмування надає легкі конструкції для виконання паралельних дій	1
T11	Функції безпеки	1	Високий пріоритет для забезпечення цілісності повідомлень, конфіденційності та захисту від атак.	5
T12	Інтеграція з інструментами або API сторонніх розробників.	1	Необов'язково, але корисно для розширюваності.	1
T13	Особливі потреби в навчанні	0.5	Сервіс має бути простим у використанні	1

Технічний фактор (TCF) є показником, який враховує технічну складність проекту. Він розраховується за формулою: TCF = 0.6 + (0.01 * коефіцієнт технічних вимог).

Розрахований технічний фактор (TCF) = 0.6 + (0.01 * 16) = 1.16

Зовнішній фактор (ECF) враховує вплив зовнішніх факторів на проект, таких як інтеграція з іншими системами, вимоги до безпеки, гнучкість, масштабованість та інші аспекти.

Розрахований зовнішній фактор (ECF) = 1.4 + (-0.03 * 14.5) = 0.995

Розрахунок ECF здійснюється за формулою: ECF = 1.4 + (-0.03 * коефіцієнт зовнішніх факторів). Оцінка за методологією Use Case Points (AUCP) розраховується як добуток основної кількості Use Case Points на TCF та ECF, також ми врахуємо відсоток перевикористання.

Табл 2.4 – Зовнішні фактори

Фактор	Опис	Вага	Обґрунтування оцінки	Оцінка
F1	Знайомство з процесом розробки	1	Знання процесу дозволяє уникнути помилок та знижує час адаптації.	2
F2	Мотивація	1	Мотивація команди впливає на продуктивність. Висока мотивація скорочує час на виконання завдань.	2
F3	Досвід роботи над подібними проектами	0.5	Попередній досвід знижує криву навчання і зменшує кількість помилок у реалізації.	3
F4	Доступність інфраструктури	1	Повна готовність інфраструктури пришвидшує старт проєкту. Затримки у її налаштуванні збільшують ризики.	2
F5	Якість технічної документації	1	Добре структурована документація знижує ризики неправильного розуміння вимог.	2
F6	Залучення розробників	1	Повня зайнятість	2
F7	Стабільність вимог	2	Можуть бути незначні зміни	1
F8	Складність мови програмування	1	Відсутня	0

Розрахована оцінка (AUCP) = 44 * 1.16 * 0.995 * (1 - 0.6) = 27.7

Кількість робочих годин визначається множенням оцінки AUCP на кількість робочих годин, необхідних для одного Use Case Point (15 годин на один пункт в нашому випадку). Виходячи з AUCP (412 годин), маємо 11 тижнів, що приблизно сходиться з термінами виконанням кваліфікаційної роботи. Розподіливши час отримаємо

Табл 2.5 – Розподіл часу

Етап виконання	Час на виконання
Визначення теми кваліфікаційної роботи та аналіз предметної області	4 дні
Аналіз аналогів	1 тиждень
Планування роботи	5 днів
Проектування системи	1 тиждень
Розробка серверної частини	3 тижні
Розробка клієнтської частини	3 тижні
Тестування і експлуатація системи	1 тиждень
Оформлення роботи	2 тижні

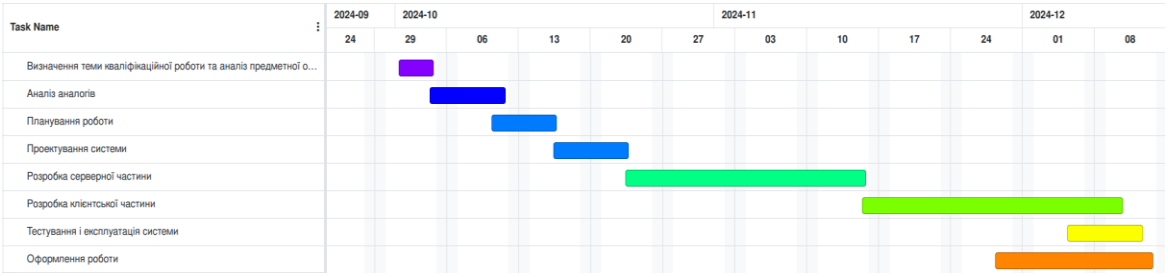


Рисунок 2.2 – Діаграма Ганта

2.5. Аналіз можливих ризиків та викликів під час розробки

На цьому етапі ми визначили можливі ризики (таблиця 2.6), які можуть виникнути під час виконання проекту, визначивши методи їх усунення, потенційно підвищивши шанси на успішне завершення проекту.

Таблиця 2.6 – Ризики

Ризик	Ймовірність	Вплив	Шлях усунення
Помилки в реалізації комунікації	Висока	Високий	Проводити модульне тестування компонентів.
Проблеми інтеграції між вузлами	Висока	Середній	Створити чітку документацію API та інтеграцій; регулярно тестувати вузли на сумісність.
Вразливість до атак	Середня	Високий	Реалізувати наскрізне шифрування з самого початку; впроваджувати secure-by-design принципи в архітектуру
Непередбачені затрати часу через нові вимоги	Висока	Високий	Визначити мінімальний набір функціональності (MVP) і зосередитися на ньому; додаткові вимоги впроваджувати лише після завершення основного функціоналу.
Не впоратися зі складністю проєкту	Середня	Високий	Розбити проєкт на менші, керовані етапи (ітерації). Визначити пріоритети через створення MVP. Проводити регулярний аудит прогресу

Аналіз ризиків проєкту показує, що найбільшу загрозу становлять потенційні помилки в реалізації комунікації, які мають високу ймовірність та критичний вплив на систему. Це вимагає впровадження суцільного модульного тестування всіх комунікаційних компонентів.

Окрему увагу слід приділити безпековим аспектам - вразливість до атак може бути мінімізована через впровадження наскрізного шифрування та принципів secure-by-design. Управлінські ризики, пов'язані з непередбаченими витратами часу та складністю проєкту, можливо контролювати через чітке визначення мінімально життєздатного продукту (MVP) та розбиття проєкту на менші, контрольовані ітерації. Загальний висновок полягає в необхідності гнучкої методології розробки, яка пріоритезує безпеку, тестування та поетапну реалізацію з чіткими контрольними точками, що дозволить випереджати потенційні ризики замість реагування на них post-factum.

2.6. Висновки до розділу

У цьому розділі були визначені функціональні та нефункціональні вимоги до системи, проведено аналіз їхньої складності та впливу на розробку. Сформульовано технічні та зовнішні фактори, які впливають на оцінку часу розробки, що дозволило розрахувати необхідний обсяг робіт та розробити детальний план виконання проєкту.

Було оцінено основні ризики, які можуть виникнути під час реалізації системи, зокрема помилки в комунікації, проблеми інтеграції вузлів, вразливість до атак та управлінські складнощі. Для мінімізації цих ризиків запропоновано стратегії, такі як впровадження модульного тестування, застосування наскрізного шифрування, чітка документація API та фокус на розробці MVP.

Загальний підхід до реалізації проєкту базується на поетапній розробці з використанням гнучкої методології, що дозволить ефективно реагувати на зміни вимог, забезпечувати безпеку та масштабованість системи. Таким чином, обґрунтовані вимоги та планування роботи створюють основу для успішного виконання проєкту в зазначені терміни.

РОЗДІЛ 3
РОЗРОБКА ВЛАСНОЇ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕНЬ

3.1. Проектування архітектурної частини

Системна архітектура — це концептуальна модель, яка описує структуру, функціональність і взаємодію компонентів системи. Вона визначає зв’язки між цими компонентами, їхню спільну роботу, виконувані функції, а також спосіб взаємодії з користувачами та іншими системами. Федеративна система обміну повідомленнями (рисунок 3.1) базується на архітектурі клієнт-сервер, оскільки кожен клієнт взаємодіє з сервером для доступу до своїх даних і відправки повідомлень. Крім того, інтероперабельність серверів, керованих різними постачальниками, підтримує децентралізовану комунікацію між користувачами системи. В своїй суті федеративна система залишається розподіленою. [23]

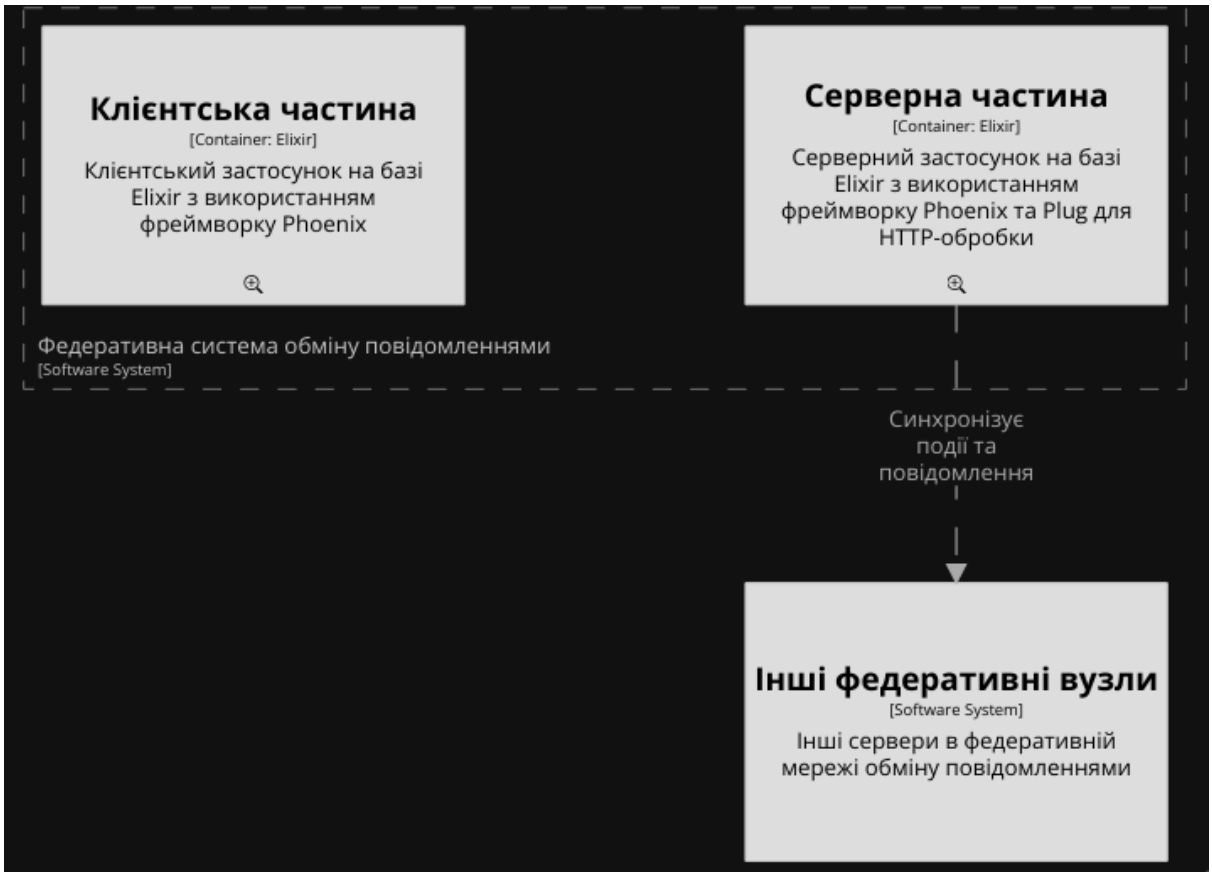


Рисунок 3.1 – Модель контейнерів створюваної системи

На серверній стороні (рисунок 3.2) система реалізована за допомогою HTTP-сервера, розробленого за допомогою Elixir та бібліотеки Plug. Для управління даними та їх зберігання використовується розподілена база даних Mnesia, відома своєю продуктивністю та ефективністю в управлінні складними операціями, а також легко розгортається, оскільки постачається разом з Erlang. [59][60][61]

Для клієнта додаток побудовано на Elixir з використанням фреймворку Phoenix, оскільки клієнтська частина зможе мати певну логіку як спільну з сервером. Для локального зберігання даних використовується SQLite3, що дозволяє користувачам отримувати швидкий доступ до необхідних даних, зменшуючи при цьому обчислювальне навантаження на сервери. [62][63]

Вибір мови обумовлений її простотою та ефективністю в умовах розподілених систем, стресостійкістю та можливостями в масштабуванні.

Рисунок 3.2 – Модель компонентів сервера

Розподілена структура системи забезпечує безперебійний зв'язок між серверами різних постачальників, зменшуючи залежність від єдиної точки відмови та підвищуючи безпеку і конфіденційність даних користувачів.

Для оптимізації розробки та організації система використовує архітектуру OTP (Open Telecom Platform), яка є потужним методологічним підходом до створення розподілених, масштабованих та відмовостійких інформаційних систем. Вона забезпечує комплексне структурування та чітке розмежування рівнів бізнес-логіки, компонентів та інтерфейсних рішень через унікальну

концепцію супервізійних дерев та модульну парадигму побудови програмних застосунків. Принципова особливість архітектурного підходу OTP полягає в імплементації ієрархічних механізмів моніторингу, контролю та відновлення процесів, що надає системі високий рівень відмовостійкості та надійності функціонування. Модульна структура забезпечує пряму можливість динамічного оновлення або вдосконалення окремих компонентів без порушення загальної архітектурної цілісності, що є критично важливим фактором для забезпечення довгострокової адаптивності та гнучкості інформаційної системи.

Для тестування компонентів використовуються сучасні інструменти, такі як ExUnit для Elixir, що гарантує надійність і точність системи. [64]

Ця архітектура закладає міцний фундамент для розробки програмного забезпечення, дозволяючи ефективно використовувати ресурси та створювати продукт високої якості. Дизайн підтримує гнучкість і масштабованість, що дозволить ефективно адаптувати систему до динамічних вимог користувачів.

3.2. Опис сховища даних

Проектування сховища даних полягає в організації та структуризації зберігання даних для підвищення ефективності їх аналізу та використання. Це передбачає вибір відповідних технологій та інструментів для збереження даних, створення схем їх структури, встановлення взаємозв'язків між таблицями, організацію доступу та впровадження методів для аналізу та використання інформації. Такий підхід покращує швидкість і ефективність роботи з даними, спрощує їх аналіз та використання, а також забезпечує високу якість і надійність даних.

Як вже було сказано перед цим, для серверу було обрано Mnesia, яка є noSQL базою даних. Тим не менш вона підтримує транзакції, має нативний інтерфейс для мов на базі Beam, була спроектована для умов розподілених

систем та забезпечує механізми для досягнення цілісності даних та відновлення у разі помилок.

У запропонованій федеративній системі обміну інформації було реалізовано наступні моделі та їх зв'язки (таблиця 3.3):

- Модель Вузла Системи та Модель Запрошення мають зв'язок один до багатьох, оскільки з одного серверу може надходити багато запрошень
- Модель Вузла та Модель Повідомлення мають такий же зв'язок один до багатьох по тій же причині
- Модель Запрошення та Модель Повідомлення мають зв'язок один до багатьох, оскільки запрошення створює новий чат і виступає його ідентифікатором, а в одному чаті може бути багато повідомлень
- Модель Вузла Системи та Модель Відкладенної Події мають зв'язок один до багатьох, оскільки у черзі для синхронізації може бути відкладено багато подій з одного серверу.
- Модель Повідомлення та Модель Вузла Системи підтримують зв'язок багато до одного, оскільки повідомлення може мати лише один сервер-джерело, проте з одного сервера може прийти багато повідомлень.

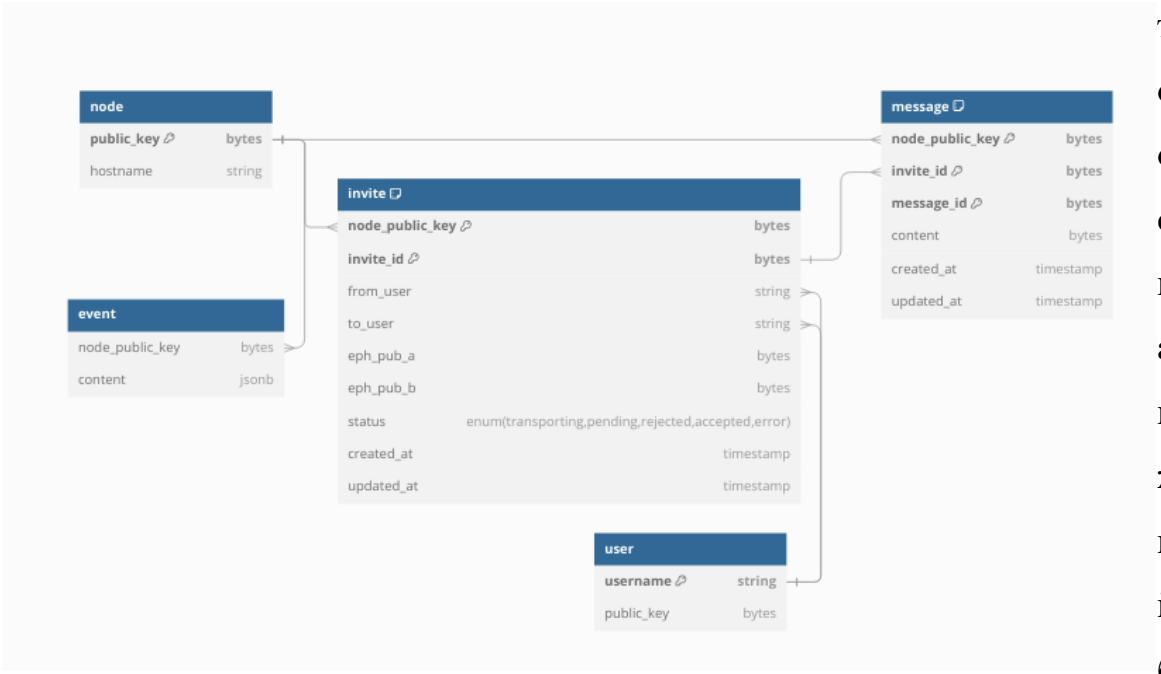


Рисунок 3.3 – Структура бази даних сервера

Структура бази даних для клієнта складається з двох основних таблиць: chats та messages. Таблиця chats зберігає інформацію не тільки про сам чат, але й про запрошення, що ініціює чат. Вона містить ідентифікатор чату, дані про користувача, з яким ведеться чат, а також тимчасові ключі для шифрування і статус чату. Крім того, зберігаються поля для відслідковування часу створення та оновлення чату, що дозволяє коректно управляти життєвим циклом чату та запрошення. Таблиця messages зберігає повідомлення, пов'язані з конкретним чатом, і містить ідентифікатор повідомлення та мітку часу його надходження. Взаємозв'язок між таблицями забезпечується через chat_id, що є зовнішнім ключем у таблиці messages та зв'язує повідомлення з відповідним чатом.

3.3. Специфікація API серверу

У цьому розділі описано реалізацію серверного API, яке забезпечує взаємодію між клієнтами та сервером у рамках розробленого федеративного месенджера. Вибір саме HTTPS для комунікаційного рівня був стимульований простотою розробки, наявними інструментами та фактом того, що такий же вибір свого часу зробили розробники Matrix. У Matrix обрали HTTPS з кількох ключових причин: по-перше, цей протокол забезпечує шифрування всього трафіку за допомогою TLS (Transport Layer Security), що запобігає можливості перехоплення чи модифікації даних третіми сторонами. По-друге, HTTPS є добре задокументованим стандартом із широкою підтримкою інструментів для реалізації, тестування та обслуговування, що значно спрощує розробку. Нарешті, використання HTTPS дозволяє забезпечити сумісність із існуючою інфраструктурою веб-технологій, що робить інтеграцію простішою та ефективнішою

API.User.Router (таблиця 3.1) обробляє операції, пов'язаних з управлінням обліковими записами користувачів. Всі запити, що стосуються реєстрації,

перевірки та синхронізації користувачів, обробляються через цей маршрутизатор.

Таблиця 3.1 – Кінцеві точки Api.User.Router

Метод	Шлях	Опис
POST	/user/register	Реалізує функцію реєстрації нового користувача. На вхід отримує JSON об’єкт з полем username, у відповідь сервер повертає зв’язку ключів ed25519. Може бути опціонально вимкнено адміністратором
POST	/user/check	Використовується для перевірки існування користувача та провіряється підпис. Отримує ідентифікатор користувача та підпис, на що сервер відповідає успішним статусним кодом, або відповіддю про невдалу провірку
POST	/user/sync	Відповідає за синхронізацію даних користувача, зокрема, відправку чи отримання оновленої інформації. Його застосування важливе для підтримки актуальності даних у розподіленій системі.

Маршрутизатор API.Node.Router (таблиця 3.2) відповідає за взаємодію між вузлами федерації. Запити, що стосуються отримання інформації про вузли, перевірки їх статусу та реєстрації нових вузлів, обробляються через цей маршрутизатор.

Таблиця 3.2 – Кінцеві точки API.Node.Router

Метод	Шлях	Опис
GET	/node/info	Надає загальну інформацію про вузол мережі, його статус, публічний ключ для підписів та доменне ім’я. Він не потребує додаткових параметрів у запиті
GET	/node/version	Служить для отримання поточної версії програмного забезпечення вузла. Цей запит також не вимагає передачі даних у тілі
POST	/node/events	Реалізує функцію передачі подій між вузлами. У тілі запиту надсилаються дані про події у вигляді масивів, кожен елемент якого має відповідну структуру до якоїсь події, наприклад нового повідомлення або запрошення. При конфлікті не намагається синхронізувати проблемні події, проте повертає які саме були проігноровані
POST	/node/register	Використовується для реєстрації нового вузла у мережі. У запиті передаються необхідні дані для ідентифікації вузла. Отримуючий сервер потім надсилає запит на доменне ім’я, провіряючи всі дані. При помилці не намагається вирішити конфлікт.

Маршрутизатор API.Message.Router (таблиця 3.3) забезпечує передачу та синхронізацію повідомлень між клієнтами та сервером.

Таблиця 3.3 – Кінцеві точки API.Message.Router

Метод	Шлях	Опис
POST	/message	Відправляє повідомлення певному користувачеві
POST	/message/fetch	Служить для отримання поточних даних про окреме повідомлення
POST	/message/sync	Служить для синхронізації повідомлень клієнта з сервером. Надає масив зі змінами різних запрошень

Маршрутизатор API.Invitation.Router (таблиця 3.4) відповідає за управління запрошеннями між користувачами. Запити для створення, отримання та синхронізації запрошень обробляються через цей маршрутизатор.

Таблиця 3.4 – Кінцеві точки API.Invitation.Router

Метод	Шлях	Опис
POST	/invite/create	Створює нове запрошення на обмін ключами з іншим користувачем. Очікує в тілі публічні ключі та ідентифікатор іншого користувача, якщо користувач знаходиться на іншому сервері, то створює запис у чергу для синхронізації.
POST	/invite/fetch	Служить для отримання поточних даних про окреме запрошення.
POST	/invite/reject	Відхиляє запрошення, якщо відправник знаходиться на іншому сервері, то створює запис у чергу для синхронізації, очікує
POST	/invite/accept	Відхиляє запрошення, якщо відправник знаходиться на іншому сервері, то створює запис у чергу для синхронізації
POST	/invite/sync	Служить для синхронізації запрошень клієнта з сервером. Надає масив зі змінами різних запрошень

Такий поділ на маршрутизатори дозволив зберегти логічну ізольованість функціоналу, спростити тестування. Крім того, така архітектура дозволяє легко

масштабувати серверні компоненти за необхідності, оскільки кожен маршрутизатор може бути виділений на окремі мікросервіси чи сервери без значних змін в інших частинах системи. Для кожного POST запиту надається підпис для підтвердження його походження.

3.4. Опис використовуємих алгоритмів шифрування

Після успішної реєстрації для користувача генерується пара ключів Ed25519: приватний ключ підпису та публічний ключ підпису. Ed25519 – це сучасний алгоритм цифрового підпису на основі еліптичних кривих Curve25519. Він характеризується високою швидкістю генерації та перевірки підписів, високим рівнем безпеки та стійкістю до різних видів атак, включаючи атаки по побічних каналах. Приватний ключ підпису зберігається у користувача, забезпечуючи його особисту ідентифікацію, тоді як публічний ключ підпису зберігається на сервері та асоціюється з ім’ям користувача. Цей публічний ключ використовується для перевірки автентичності повідомлень, підписаних відповідним приватним ключем.

Для встановлення захищеного з'єднання та обміну повідомленнями між користувачами використовується процес запрошення. Користувач A, який бажає запросити користувача B, спочатку генерує ефемерну пару ключів X25519: ефемерний приватний ключ та ефемерний публічний ключ. X25519 – це функція обміну ключами, побудована на основі еліптичної кривої Curve25519. Її головна функція – безпечне встановлення спільного секретного ключа між двома сторонами по незахищеному каналу зв'язку. Ефемерність ключів означає, що вони генеруються для кожної окремої сесії та не використовуються повторно, що значно підвищує безпеку. Користувач A створює повідомлення-запрошення, яке містить його ефемерний публічний ключ, ім’я користувача B (того, кого запрошують) та своє ім’я користувача (хто запрошує). Потім користувач A підписує це повідомлення своїм приватним

ключем підпису Ed25519, отриманим під час реєстрації. Підписане повідомлення відправляється на endpoint сервера /invite/create.

Після відправлення запрошення сервери синхронізуються між собою, обмінюючись інформацією про нові запрошення. Під час синхронізації сервери перевіряють цифрові підписи один одного, підтверджуючи автентичність та цілісність даних, якими вони обмінюються. Сервер, на якому знаходиться користувач Б, отримує інформацію про запрошення, адресоване йому. Вузол довіряє, що сервер А вже перевірив підпис користувача А на повідомленні-запрошенні, тому повторна перевірка підпису користувача на цьому етапі не потрібна. Замість цього сервер Б перевіряє, чи отримане ним повідомлення про запрошення є автентичним та не було підроблено під час передачі між серверами, перевіряючи підпис сервера А. Якщо синхронізація пройшла успішно та повідомлення про запрошення автентичне, воно передається користувачу В.

Отримавши запрошення, користувач В також генерує свою ефемерну пару ключів X25519. Віддає її серверу, після чого стається повторна синхронізація серверів. Після цього обидва користувачі мають змогу обчислити спільний секретний ключ, використовуючи свої ефемерні приватні ключі та ефемерні публічні ключі один одного. Завдяки математичним властивостям X25519, обидва користувачі отримують ідентичний спільний секрет, навіть якщо обмін публічними ключами відбувається по незахищеному каналу. З цього спільного секрету за допомогою функції отримання ключа (KDF) з використанням хеш-функції Blake2 генерується ключ для шифрування повідомлень. Blake2 – це криптографічна хеш-функція, що забезпечує високу швидкість та безпеку. KDF використовується для перетворення спільного секрету на криптографічно сильний ключ шифрування, придатний для використання з алгоритмом AES.

Для шифрування самих повідомлень використовується симетричний алгоритм шифрування AES256 у режимі GCM. AES (Advanced Encryption Standard) є стандартом де-факто для симетричного шифрування. AES256 означає використання ключа довжиною 256 біт, що забезпечує високий рівень

криптографічної стійкості. Режим GCM (Galois/Counter Mode) не тільки шифрує повідомлення, забезпечуючи конфіденційність, але й додає тег автентифікації, що гарантує цілісність даних та захист від підміни. Після шифрування повідомлення за допомогою AES256-GCM, воно відправляється на сервер, де потім передається отримувачу. Отримувач, маючи спільний секрет та згенерований з нього ключ, може розшифрувати повідомлення та перевірити його цілісність. Весь цей процес зображений на рисунку 3.4

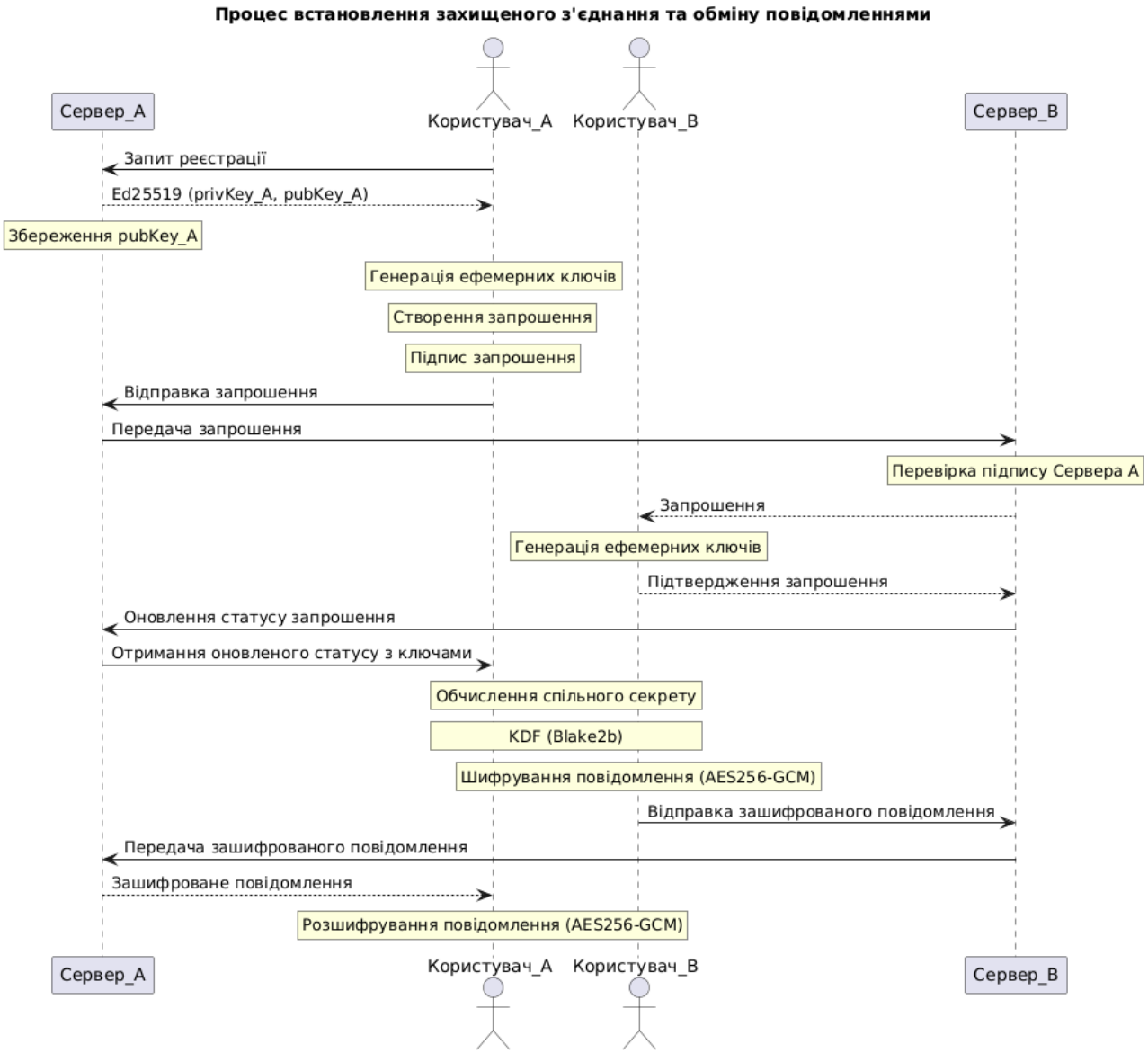


Рисунок 3.4 – Процес встановлення захищеного з’єднання та обміну інформацією

Вибір криптографічних алгоритмів Ed25519 для підпису, X25519 для обміну ключами та AES256-GCM для шифрування повідомлень був зумовлений їх високою ефективністю, стійкістю до атак та здатністю забезпечити конфіденційність і цілісність даних. Вони дають змогу гарантувати безпечний обмін інформацією в умовах розподіленої системи, де кожен компонент забезпечує свою частину захисту. Вони є доступними в стандартній криптографічній бібліотеці Erlang, яку можна безпосередньо використовувати в Elixir. Це дозволяє ефективно інтегрувати ці алгоритми в систему, не потребуючи додаткових зовнішніх бібліотек або складних налаштувань.

3.5. Опис користувацького інтерфейсу

Інтерфейс користувача (UI) нашої системи обміну повідомленнями був розроблений з акцентом на простоту і зручність використання, дотримуючись принципу мінімалізму, але при цьому задовольняючи функціональні вимоги програми. Дизайн включає традиційний двопанельний макет, який зазвичай використовується в додатках для обміну повідомленнями, щоб підвищити зручність користування та простоту навігації. Ліва панель слугує списком чатів, що відображає імена контактів у чіткій та організованій формі. Ця панель займає меншу частину екрана, щоб основна увага залишалася зосередженою на вмісті чату, який відображається з правого боку. Мінімізуючи розмір списку чату, ми максимізуємо доступний простір для активних розмов, забезпечуючи більш захоплюючий користувацький досвід.

Щоб зберегти сумісність з технічними обмеженнями системи, деякі функції були навмисно опущені. Наприклад, в інтерфейсі немає аватарів поруч з іменами контактів, оскільки система не підтримує візуальні елементи такого роду. Це рішення відповідає основній меті - зменшити складність та забезпечити легкість і функціональність дизайну.

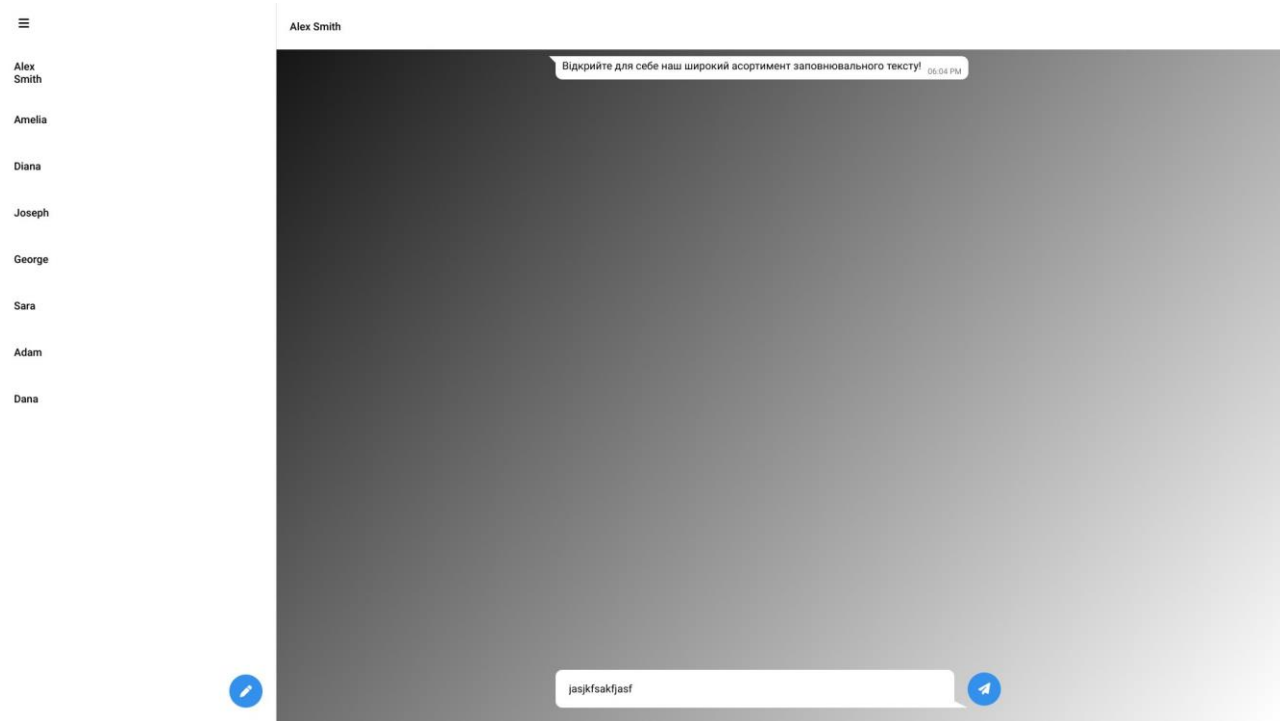


Рисунок 3.5 – Користувацький інтерфейс

Область вмісту чату, що займає більшу частину екрану, призначена для сприяння безперешкодному спілкуванню. Користувачі можуть переглядати повну історію розмов з контактом в цьому розділі і створювати нові повідомлення в тому ж інтерфейсі. Поле для введення тексту розташоване в нижній частині екрану, поруч із простою кнопкою надсилання, що створює інтуїтивно зрозумілий потік взаємодії.

За допомогою спливаючого вікна, захованого за кнопкою в лівій частині, користувачі можуть запрошувати нових контактів приєднатися до них, спрощуючи процес розширення своєї мережі. Крім того, спливаюче вікно дозволяє користувачам переглядати отримані запрошення та керувати ними. Ця функція подвійного призначення легко інтегрована в інтерфейс користувача, зберігаючи мінімалістичну естетику і пропонуючи при цьому важливі соціальні функції. Консолідуючи ці функції в одному доступному компоненті, дизайн гарантує, що користувачі можуть легко виконувати ці дії, не захаращуючи основний інтерфейс.

Загальний дизайн відображає ретельний баланс між формою та функціями. Зосереджуючись на основних функціях і спрощуючи процес взаємодії, система

обміну повідомленнями забезпечує зручний користувацький досвід, пристосований до технічних можливостей платформи.

3.5. Тестування серверу та клієнту

Ми провели обширний і ретельний процес тестування за допомогою ExUnit [64], щоб забезпечити надійність і стійкість нашої системи. Стратегія тестування передбачає систематичну розбивку додатку на логічно згруповані компоненти, кожному з яких були присвоєні певні теги, що відображають їх функціональність і залежності. Наприклад, ми позначили компоненти, що використовують бібліотеку Plug, тегом plug, що відображає їхню роль в обробці HTTP-запитів і відповідей. Компоненти, які взаємодіяли з базами даних, наприклад, ті, що використовували Mnesia або Ecto, були позначені тегами mnesia та ecto відповідно, що підкреслювало їхню важливість у зберіганні, пошуку та управлінні транзакціями. Крім того, для модулів, які потребують взаємодії із зовнішніми серверами для коректної роботи, ми використовували тег federated, щоб підкреслити їхню потребу в міжсерверній взаємодії та залежність від федеративних архітектур.

Щоб перевірити цілісність і функціональність цих компонентів, ми розробили і впровадили великий набір тестів. Кожен тест був написаний з точністю, щоб охопити широкий спектр сценаріїв, включаючи граничні ситуації, обробку помилок і продуктивність за різних умов. Успіх цих зусиль очевидний з результатів, оскільки всі тести пройшли успішно, продемонструвавши, що система поводиться так, як очікувалося, у всіх тестових сценаріях.

Окрему увагу ми приділили навантажувальному тестуванню, результати якого відображені на графіках (рисунок 3.6). Тестування передбачало запуск 32 вузлів, кожен з яких обслуговував до 30 тисяч автоматизованих користувачів, що постійно відправляли повідомлення користувачам сусідніх вузлів. Система продемонструвала стабільну пропускну здатність на рівні приблизно 1200

запитів на секунду (rRPS), при цьому кількість невдач залишалася майже нульовою протягом більшості часу тестування. Це підтверджує високу стійкість і ефективність системи під значним навантаженням. Середній час відповіді залишався низьким, що свідчить про швидку обробку запитів для більшості користувачів. Однак ближче до завершення тестування спостерігалося зростання часу відповіді у верхніх 5% випадків, що, ймовірно, є наслідком досягнення максимальної пропускної здатності системи.

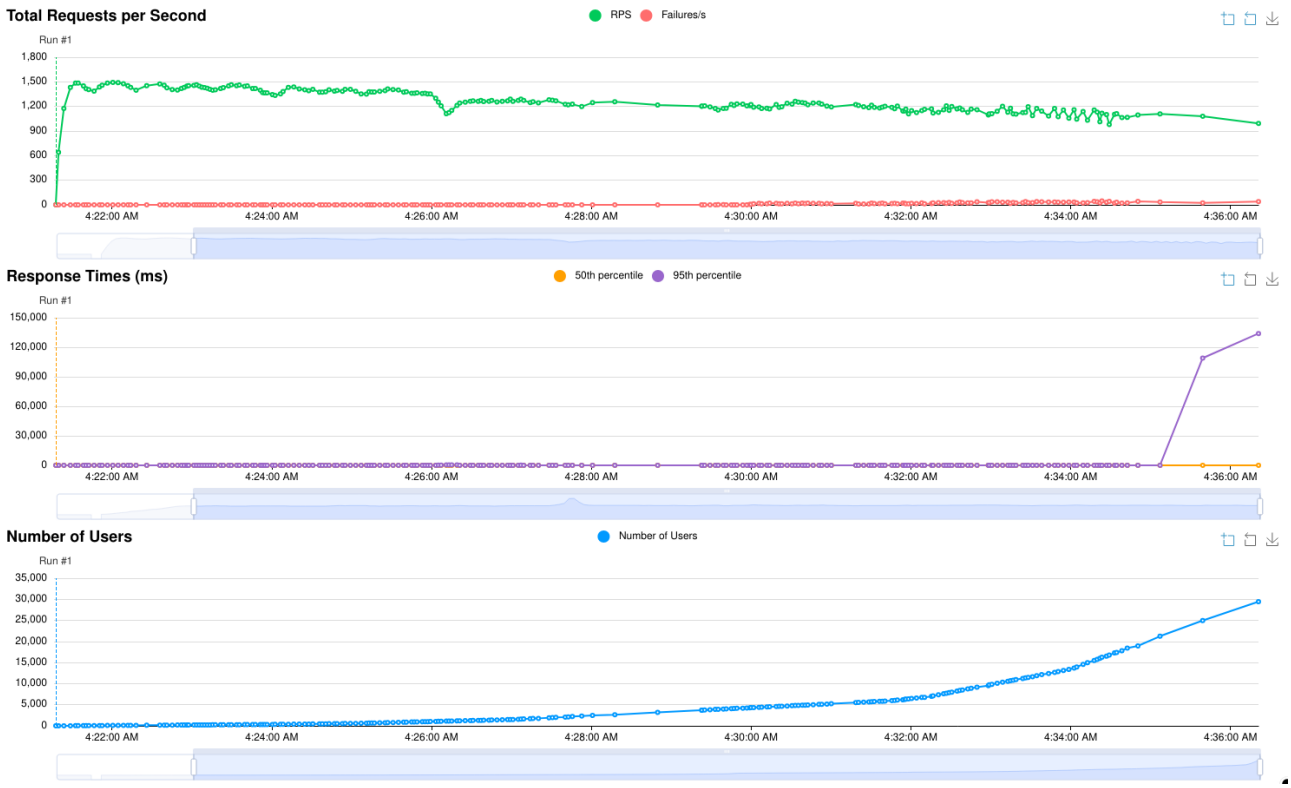


Рисунок 3.6 — Графіки навантажувального тестування

Крім того, наш фокус на досягненні всебічного покриття дозволив нам досягти загального тестового покриття 70%, гарантуючи, що більша частина кодової бази системи була виконана під час тестування. Поєднання організованої системи тегування, великої кількості тестів і значного покриття підкреслює наше прагнення створити надійну, зручну в обслуговуванні і добре протестовану систему, яка відповідає високим стандартам якості і продуктивності.

3.6. Розгортання серверу

Наша стратегія розгортання була прикладом добре структурованого та сучасного процесу DevOps, підкреслюючи безперешкодну інтеграцію методів розробки та експлуатації для створення високопродуктивного, безпечного та зручного в обслуговуванні застосунку. Розмістивши наше рішення на Debian, ми використали його стабільність і сумісність із сучасними інструментами для створення надійного виробничого середовища. Центральним елементом нашого підходу було використання Docker, який уможливив контейнеризацію додатку та його залежностей, забезпечуючи відтворюваність на етапах розробки, тестування та виробництва. Ця парадигма контейнеризації не тільки спростила управління середовищем, але й втілила принципи інфраструктури як коду, що є наріжним каменем філософії DevOps.

Безпека була фундаментальним фактором при розгортанні нашої системи. Щоб захистити цілісність і конфіденційність даних, ми використовували сертифікати SSL/TLS від Let's Encrypt, а Certbot автоматизував процеси видачі та поновлення сертифікатів. Ця автоматизація зменшила ручне втручання, що відповідає принципам ефективності та масштабованості DevOps, забезпечуючи при цьому відповідність сучасним стандартам безпеки.

Nginx виконував роль зворотного проксі в нашій архітектурі, виконуючи такі важливі функції, як маршрутизація трафіку до відповідних контейнерів, забезпечення HTTPS-з'єднань та оптимізація продуктивності за допомогою балансування навантаження. Його гнучкість дозволила нам впроваджувати модульні конфігурації, задовольняючи складні вимоги до маршрутизації, зберігаючи при цьому низьку затримку і високу надійність.

Важливим аспектом нашої реалізації була інтеграція GitHub Actions для безперервної інтеграції та безперервного розгортання (CI/CD). Цей фреймворк автоматизації дозволив створити єдиний конвеєр для побудови, тестування та розгортання додатку безпосередньо з нашого репозиторію GitHub. Автоматизувавши ці процеси, ми мінімізували людську помилку, скоротили час

розгортання та сприяли розвитку культури постійного вдосконалення - ознаки ефективних практик DevOps.

Таким чином, розроблена нами інфраструктура розгортання є прикладом цілісного застосування принципів DevOps, інтегруючи передові технології та автоматизацію для досягнення безпечної, масштабованої та ефективної операційної структури. Цей підхід не тільки відповідає сучасним передовим практикам, але й забезпечує основу для майбутньої масштабованості та інновацій, підкреслюючи синергію між технологічним прогресом і методологічною суворістю.

ВИСНОВКИ

У дипломній роботі на основі проведених досліджень здійснено узагальнення та практичне вирішення актуального питання створення федеративної системи обміну повідомленнями з використанням наскрізного шифрування. Основні висновки та рекомендації роботи такі:

1. Проведене дослідження сучасних федеративних систем обміну повідомленнями дозволило ідентифікувати ключові тенденції, сильні сторони та виклики, які стоять перед цими технологіями. Федеративні системи вирізняються своєю здатністю підтримувати децентралізовану архітектуру, що дозволяє різним серверам працювати автономно та взаємодіяти через стандартизовані протоколи. Ці системи забезпечують високу гнучкість, інтероперабельність та масштабованість, що є важливими характеристиками для сучасних комунікаційних платформ. Однак аналіз показав, що, попри переваги, федеративні системи стикаються з численними викликами. Синхронізація стану між серверами, оптимізований обмін подіями та підтримання стандартів між різними реалізаціями протоколів як приклад. Закриті екосистеми, такі як Signal, забезпечують найвищий рівень безпеки завдяки наскрізному шифруванню, використанню сучасних криптографічних протоколів та прозорості їх реалізації. Проте вони не підтримують інтеграцію з іншими платформами, що обмежує їх використання у федеративних мережах. Таким чином, дослідження підтвердило, що сучасні федеративні системи мають значний потенціал для розвитку, однак потребують подальшої роботи над вирішенням питань масштабованості, ефективності роботи в умовах групових сценаріїв і уніфікації стандартів обміну ключами. Розробка нових архітектур і підходів, які поєднують безпеку закритих систем та інтероперабельність федеративних мереж, є перспективним напрямком для подальших досліджень.

2. Наскрізне шифрування (E2EE) є фундаментальним елементом безпеки у федеративних системах обміну повідомленнями, особливо у середовищах, де передача даних відбувається через інші сервери, а сама архітектура передбачає

децентралізацію. Аналіз сучасних підходів до шифрування підтвердив, що саме E2EE дозволяє гарантувати конфіденційність повідомлень у таких умовах. Федеративні системи побудовані на принципах децентралізації, що передбачає взаємодію незалежних серверів. Наприклад, повідомлення від користувача A на сервері X до користувача B на сервері Y передається через кілька проміжних серверів або вузлів. Цей процес вимагає дуплікації даних для синхронізації між серверами, що створює додаткові точки вразливості. У такій моделі сервери виконують роль посередників, через які проходять зашифровані дані. Однак самі сервери не мають доступу до змісту повідомлень, оскільки розшифрувати їх можуть лише кінцеві користувачі. Алгоритми Double Ratchet і X3DH, що лежать в основі Signal Protocol, є прикладами ефективного вирішення цієї проблеми. Вони забезпечують, що навіть у разі компрометації одного з серверів жодна проміжна точка не може отримати доступ до змісту повідомлення. Це стає можливим завдяки постійному оновленню ключів після кожного повідомлення, що унеможливлює розшифрування перехоплених даних навіть за наявності старих ключів. У випадку групових чатів, які потребують синхронізації даних між багатьма учасниками, виклик стає ще складнішим. Протокол Megolm, що використовується в Matrix, вирішує цю проблему через централізоване використання симетричного ключа, який періодично оновлюється, згаданий Signal використовує схоже рішення. Проте ризики залишаються у разі компрометації цього ключа, що потребує подальших оптимізацій. Таким чином, наскрізне шифрування в умовах дуплікації даних і передачі через проміжні сервери є не лише необхідним, але й ключовим елементом захисту. Воно забезпечує конфіденційність повідомлень навіть у децентралізованих системах, де кожен сервер потенційно може бути вразливим до зовнішніх атак або несанкціонованого доступу. Подальший розвиток алгоритмів управління ключами та їх адаптація до особливостей федеративних архітектур є пріоритетним завданням для сучасних дослідників безпеки.

3. У рамках дипломної роботи нами було розроблено і протестовано федеративну систему обміну повідомленнями, що використовує власні рішення

на основі перевірених криптографічних алгоритмів для забезпечення наскрізного шифрування (E2EE). При розробці системи були враховані важливі аспекти безпеки, зокрема, необхідність забезпечення конфіденційності та цілісності даних на всіх етапах передачі. Для забезпечення безперебійної роботи в федеративній мережі, де дані передаються через численні сервери, нами була враховані архітектурні рішення, що дозволяють ефективно синхронізувати інформацію між серверами та підтримувати стабільний обмін повідомленнями. Прототип також підтримує високий рівень захисту даних завдяки застосуванню наскрізного шифрування, що запобігає доступу сторонніх осіб до змісту повідомлень. У процесі тестування було перевірено не тільки функціональність системи, але й її здатність ефективно працювати при великих навантаженнях, зокрема при одночасному підключенні численних користувачів. Результати показали, що система здатна забезпечити належний рівень працездатності навіть при великій кількості оброблених повідомлень та підключених клієнтів. Таким чином, розроблена система є стабільним і функціональним прототипом, який може бути використаний для подальших досліджень або інтеграції в комерційні рішення. Система забезпечує надійний захист даних, високу продуктивність і стабільність, що робить її підходящою для реальних застосувань у федеративних мережах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Van Vleck T. The history of electronic mail. *Multics*. URL: https://www.multicians.org/thvv/mail-history.html (date of access: 13.12.2024).
2. B. Postel J. RFC 788: simple mail transfer protocol. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc788 (date of access: 13.12.2024).
3. RFC 1869: SMTP service extensions / J. Klensin et al. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc1869 (date of access: 13.12.2024).
4. RFC 1459: internet relay chat protocol. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc1459 (date of access: 13.12.2024).
5. Knight S. What ever happened to ICQ?. *Techspot*. 2024. URL: https://www.techspot.com/article/1771-icq/ (date of access: 13.12.2024).
6. Saint-Andre P. RFC 6121: extensible messaging and presence protocol (XMPP): instant messaging and presence. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc6121 (date of access: 13.12.2024).
7. Broersma M. Jabber numbers overtake ICQ. *ZDNET*. 2003. URL: https://www.zdnet.com/article/jabber-numbers-overtake-icq/ (date of access: 13.12.2024).
8. Botlhale Village. The evolution of instant messaging: from simple text to multifunctional platforms - botlhale village. *Botlhale Village*. URL: https://botlhalevillage.co.za/the-evolution-of-instant-messaging-from-simple-text-to-multifunctional-platforms/ (date of access: 13.12.2024).
9. Gardner M. A new kind of cipher that would take millions of years to break. *MATHEMATICAL GAMES*. 1977. Vol. 237, no. 2. P. 120–125. URL: https://www.jstor.org/stable/24954008.
10. Cobb M. What is the RSA algorithm? Definition from SearchSecurity. *Search Security*. URL: https://www.techtarget.com/searchsecurity/definition/RSA (date of access: 13.12.2024).
11. Linn J. RFC 1421: privacy enhancement for internet electronic mail: part I: message encryption and authentication procedures. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc1421 (date of access: 13.12.2024).
12. Atkins D., Stallings W., Zimmermann P. RFC 1991: PGP message exchange formats. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc1991 (date of access: 13.12.2024).
13. ProtonMail. Proton Mail encryption explained. *Proton*. URL: https://proton.me/support/proton-mail-encryption-explained (date of access: 13.12.2024).
14. McCullagh D. Yahoo, ICQ chats still vulnerable to government snoops. *CNET*. URL: https://www.cnet.com/tech/tech-industry/yahoo-icq-chats-still-vulnerable-to-government-snoops/ (date of access: 13.12.2024).
15. Saint-Andre P., Cridland D., Meijer R. XEP-0001: XMPP extension protocols. *XMPP*. URL: https://xmpp.org/extensions/xep-0001.html (date of access: 13.12.2024).

16. XEP-0384: OMEMO Encryption / A. Straub et al. *XMPP*. URL: https://xmpp.org/extensions/xep-0384.html (date of access: 13.12.2024)
17. Schmaus F., Schürmann D., Breitmoser V. XEP-0373: OpenPGP for XMPP. *XMPP*. URL: https://xmpp.org/extensions/xep-0373.html (date of access: 13.12.2024).
18. Whited S. XEP-0364: current off-the-record messaging usage. *XMPP*. URL: https://xmpp.org/extensions/xep-0364.html.
19. Billy Perrigo. The inside story of how Signal became the private messaging app for an age of fear and distrust. *Yahoo! Tech*. 2020. URL: https://www.yahoo.com/tech/inside-story-signal-became-private-150114933.html (date of access: 13.12.2024).
20. Toulas B. Meta rolls out default end-to-end encryption on Messenger, Facebook. *BleepingComputer*. URL: https://www.bleepingcomputer.com/news/security/meta-rolls-out-default-end-to-end-encryption-on-messenger-facebook/ (date of access: 13.12.2024).
21. Jakobsen J., Orland C. On the CCA (in)security of MTProto. 2015. URL: https://eprint.iacr.org/2015/1177.pdf (date of access: 13.12.2024).
22. End-to-End Encryption implementation guide. *Matrix.org*. URL: https://matrix.org/docs/matrix-concepts/end-to-end-encryption/ (date of access: 13.12.2024).
23. Rubin J. Federated systems. *MIT - Massachusetts Institute of Technology*. URL: https://www.mit.edu/~jlrubin/public/pdfs/federation.pdf (date of access: 13.12.2024).
24. Wong M., Schlitt W. RFC 4408: sender policy framework (SPF) for authorizing use of domains in e-mail, version 1. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc4408 (date of access: 13.12.2024).
25. DKIM https://web.archive.org/web/20201219210800/http://www.dkim.org/
26. DMARC https://datatracker.ietf.org/doc/html/rfc7489
27. Hancke P., Cridland D. XEP-0288: bidirectional server-to-server connections. *XMPP*. URL: https://xmpp.org/extensions/xep-0288.html (date of access: 13.12.2024).
28. Saint-Andre P., Cridland D., Meijer R. XEP-0001: XMPP extension protocols. *XMPP*. URL: https://xmpp.org/extensions/xep-0001.html (date of access: 13.12.2024).
29. About matrix. *Matrix.org*. URL: https://matrix.org/foundation/about/ (date of access: 17.12.2024).
30. Rooms & events. *Matrix.org*. URL: https://matrix.org/docs/matrix-concepts/rooms_and_events/ (date of access: 13.12.2024).
31. Bridging. *Matrix.org*. URL: https://www.matrix.org/docs/communities/bridging/ (date of access: 13.12.2024).
32. Brouwer D. Making messaging interoperability with third parties safe for users in Europe. *Engineering at Meta*. URL:

https://engineering.fb.com/2024/03/06/security/whatsapp-messenger-messaging-interoperability-eu/ (date of access: 13.12.2024)
33. Moxie Marlinspike, Trevor Perrin. The Double Ratchet Algorithm. *Signal Messenger*. URL: https://signal.org/docs/specifications/doubleratchet/ (date of access: 13.12.2024).
34. Marlinspike M., Perrin T. The X3DH key agreement protocol. *Signal Messenger*. URL: https://signal.org/docs/specifications/x3dh/ (date of access: 13.12.2024).
35. Mozilla. Signal privacy & security. **Privacy Not Included*. URL: https://foundation.mozilla.org/en/privacynotincluded/signal (date of access: 13.12.2024).
36. O'Leary J. Technology preview: Signal private group system. *Signal Messenger*. URL: https://signal.org/blog/signal-private-group-system/ (date of access: 13.12.2024).
37. Marlinspike M. Reflections: the ecosystem is moving. *Signal Messenger*. URL: https://signal.org/blog/the-ecosystem-is-moving/ (date of access: 13.12.2024).
38. Lund J. Technology preview: Sealed sender for Signal. *Signal Messenger*. URL: https://signal.org/blog/sealed-sender/ (date of access: 17.12.2024).
39. O'Leary J. Improving registration lock with secure value recovery. *Signal Messenger*. URL: https://signal.org/blog/improving-registration-lock/ (date of access: 13.12.2024).
40. Schaad J., Ramsdell B., Turne S. RFC 8551: secure/multipurpose internet mail extensions (S/MIME) version 4.0 message specification. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc8551 (date of access: 13.12.2024).
41. Schaad J., Ramsdell B., Turner S. OpenPGP HTTP keyserver protocol. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/draft-gallagher-openpgp-hkp (date of access: 13.12.2024).
42. What is active directory certificate services?. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows-server/identity/ad-cs/active-directory-certificate-services-overview (date of access: 13.12.2024).
43. Melnikov A., Leiba B. RFC 9051: internet message access protocol (IMAP) - version 4rev2. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc9051 (date of access: 13.12.2024).
44. Myers J., Rose M. RFC 1939: post office protocol - version 3. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc1939 (date of access: 13.12.2024).
45. Freed N., Borenstein N. RFC 2045: multipurpose internet mail extensions (MIME) part one: format of internet message bodies. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc2045 (date of access: 13.12.2024).
46. Freed N., Borenstein N. RFC 2046: multipurpose internet mail extensions (MIME) part two: media types. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc2046 (date of access: 13.12.2024).

47. GNU mailman. URL: https://www.gnu.org/software/mailman/ (date of access: 13.12.2024).
48. Saint-Andre P. RFC 6120: extensible messaging and presence protocol (XMPP): core. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc6120 (date of access: 13.12.2024).
49. Melnikov A., Zeilenga K. RFC 4422: simple authentication and security layer (SASL). *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc4422 (date of access: 13.12.2024).
50. Burtrum T. XEP-0368: SRV records for XMPP over TLS. *XMPP*. URL: https://xmpp.org/extensions/xep-0368.html (date of access: 13.12.2024).
51. Saint-Andre P. XEP-0045: Multi-User Chat. *XMPP*. URL: https://xmpp.org/extensions/xep-0045.html (date of access: 13.12.2024).
52. Evans S. Matrix cryptographic key infrastructure. *Sumner Evans*. URL: https://sumnerevans.com/posts/matrix/cryptographic-key-infrastructure/ (date of access: 13.12.2024).
53. Bridging. *Matrix.org*. URL: https://www.matrix.org/docs/communities/bridging/ (date of access: 13.12.2024).
54. Server-Server API. *Matrix Specification*. URL: https://spec.matrix.org/v1.3/server-server-api/#pdus (date of access: 13.12.2024).
55. Matrix specification. *Matrix Specification*. URL: https://spec.matrix.org/v1.3/ (date of access: 13.12.2024).
56. Room version 11. *Matrix Specification*. URL: https://spec.matrix.org/v1.8/rooms/v11/ (date of access: 13.12.2024).
57. Hodgson M. Matrix 2.0 is here!. *Matrix.org*. URL: https://matrix.org/blog/2024/10/29/matrix-2.0-is-here/ (date of access: 13.12.2024).
58. Cohn M. Estimating With Use Case Points. URL: https://www.cs.cmu.edu/~jhm/DMS%202011/Presentations/Cohn%20-%20Estimating%20with%20Use%20Case%20Points_v2.pdf.
59. The Elixir programming language. *The Elixir programming language*. URL: https://elixir-lang.org/ (date of access: 13.12.2024).
60. Plug. *HexDocs*. URL: https://hexdocs.pm/plug/readme.html (date of access: 13.12.2024).
61. Overview – mnesia. *Erlang/OTP*. URL: https://www.erlang.org/doc/apps/mnesia/mnesia_overview.html (date of access: 13.12.2024).
62. *Phoenix Framework*. URL: https://www.phoenixframework.org/ (дата звернення: 13.12.2024).
63. *SQLite Home Page*. URL: https://www.sqlite.org/index.html (дата звернення: 13.12.2024).
64. ExUnit. *HexDocs*. URL: https://hexdocs.pm/ex_unit/1.17.3/ExUnit.html (date of access: 13.12.2024).