МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КОВАЛЬЧУК ІЛОНА СЕРГІЇВНА

Допускається до захисту:
В.о. завідувача кафедри
прикладної математики та
кібербезпеки,
_____Луценко А.В
«__»_____20__р.

ПОБУДОВА АНОНІМНОЇ МЕРЕЖІ НА МОВІ ПРОГРАМУВАННЯ GO

Спеціальність 125 Кібербезпека
Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Чернов Д.В.,
Канд.техн. наук, доцент, доцент кафедри прикладної математики та кібербезпеки

_____
(підпис)

Оцінка : ___/ ___ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2025

АНОТАЦІЯ

Ковальчук І. С. Побудова анонімної мережі на мові програмування Go. Спеціальність 125 "Кібербезпека". Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У кваліфікаційній (бакалаврській) роботі розглянуто побудову анонімної мережі з використанням мови програмування Go. Описано принципи анонімності в мережах, особливості реалізації протоколів маршрутизації, захисту трафіку та збереження конфіденційності користувачів. Наведено архітектуру створеної моделі та приклад її реалізації.

Ключові слова: анонімна мережа, маршрутизація, конфіденційність, мова Go, захист трафіку.

70 с., 2 рис., 5 табл., 23 джерело

ABSTRACT

Kovalchuk I. S. Construction of an Anonymous Network Using the Go Programming Language.

Specialty 125 "Cybersecurity". Vasyl' Stus Donetsk National University, Vinnytsia, 2025.

The qualification (bachelor’s) thesis focuses on the development of an anonymous network using the Go programming language. It describes the principles of anonymity in networks, the implementation of routing protocols, traffic protection, and user privacy preservation. The architecture of the proposed model is presented along with an example of its implementation.

Keywords: anonymous network, cybersecurity, routing, privacy, Go language, traffic protection.

70 pp., 2 figures, 5 tables., 23 sources

ЗМІСТ

ВСТУП

В сучасному світі питання приватності та захисту персональних даних набуває все більшого значення. Зростаюча цифровізація всіх сфер життя створює нові виклики для забезпечення конфіденційності комунікацій. Розробка захищених систем обміну повідомленнями стає необхідною умовою для збереження приватності користувачів в мережі Інтернет.

Актуальність роботи обумовлена зростаючою потребою в захищених каналах комунікації, які забезпечують анонімність користувачів та конфіденційність їхнього спілкування. Існуючі месенджери часто вимагають надання персональних даних при реєстрації, що створює ризики для приватності користувачів.

Мета роботи полягає у розробці анонімної мережі для обміну повідомленнями з використанням сучасних технологій та протоколів передачі даних.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати існуючі рішення для анонімної комунікації;
- дослідити можливості мови Go для створення серверної частини;
- вивчити особливості протоколу WebSocket для обміну даними;
- розробити архітектуру системи з урахуванням вимог безпеки;
- реалізувати серверну частину на мові Go;
- створити клієнтський додаток з використанням фреймворку Svelte;
- провести тестування розробленої системи.

Об'єктом дослідження є процеси передачі інформації у комп’ютерних мережах.

Предметом дослідження є методи та технології захисту інформації у анонімних мережах для обміну повідомленнями.

Методи дослідження включають аналіз існуючих рішень, проектування архітектури системи, розробку програмного забезпечення, тестування та оптимізацію продуктивності.

Наукова новизна роботи полягає у створенні оригінального підходу до побудови анонімної мережі з використанням сучасного стеку технологій, що забезпечує високу продуктивність та надійний захист даних користувачів.

Практичне значення отриманих результатів полягає у можливості використання розробленої системи для організації захищених комунікацій у різних сферах - від корпоративного сектору до громадських організацій.

Особистий внесок здобувача полягає у проведенні аналізу існуючих рішень, розробці архітектури системи, реалізації серверної та клієнтської частин, проведенні тестування та оптимізації продуктивності.

Апробація результатів дипломної роботи. Основні положення та результати роботи доповідалися та обговорювалися на науково-технічній конференції студентів.

Структура та обсяг роботи. Дипломна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи складає 70 сторінок, у тому числі 63 сторінок основного тексту, 2 рисунків, 5 таблиць та додатки.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ АНОНІМНИХ МЕРЕЖ

1.1. Концепція анонімних мереж та їх призначення

В сучасному цифровому світі питання приватності та анонімності набуває нового значення. Користувачі інтернету щодня генерують величезні обсяги персональних даних. Великі корпорації та державні структури активно збирають та аналізують цю інформацію. Звичайні методи комунікації в мережі залишають цифрові сліди, які можна відстежити. Саме тому виникла потреба у створенні спеціальних мереж, де користувачі могли б спілкуватися без страху стеження. Такі мережі отримали назву анонімних, оскільки вони приховують особистість користувачів. Технології анонімізації постійно розвиваються та вдосконалюються.

Анонімні мережі функціонують на основі спеціальних протоколів та алгоритмів шифрування. Вони забезпечують конфіденційність даних через багаторівневе кодування інформації. Передача даних здійснюється через ланцюжок проміжних серверів, кожен з яких знає лише попередню та наступну ланку маршруту. Така архітектура унеможливлює відстеження початкового відправника повідомлення. Користувачі можуть обмінюватися повідомленнями, не розкриваючи свою реальну IP-адресу. Додатковий захист забезпечується через змінні маршрути передачі даних. Система постійно змінює шляхи проходження інформації.

Розвиток анонімних мереж розпочався з експериментальних проектів військових та наукових установ. Перші спроби створення захищених каналів зв'язку датуються 1960-ми роками. Дослідники розробляли методи приховання джерела та отримувача повідомлень. Поступово технології вдосконалювалися та ставали доступними для цивільного використання. Сьогодні анонімні мережі застосовуються журналістами, активістами та звичайними користувачами. Вони дозволяють обходити цензуру та обмеження доступу до інформації [1]. Технічна реалізація постійно оновлюється для протидії новим методам стеження.

Базовим елементом анонімних мереж виступає криптографічний захист даних. Повідомлення шифруються за допомогою складних математичних алгоритмів. Ключі шифрування генеруються випадковим чином для кожної сесії зв'язку. Розшифрувати перехоплені дані без знання ключа практично неможливо. Система також використовує додаткові методи заплутування слідів. Трафік змішується з іншими даними для маскування реального об'єму передачі. Застосовується штучне уповільнення та прискорення потоків інформації.

Особливу роль в анонімних мережах відіграє децентралізована архітектура. Відсутність єдиного центру управління підвищує стійкість системи до атак. Вузли мережі працюють незалежно один від одного на добровільних засадах. Користувачі можуть самостійно налаштовувати власні ретранслятори. Така структура забезпечує високу відмовостійкість та масштабованість. Вихід з ладу окремих вузлів не впливає на роботу всієї мережі. Система динамічно перебудовує маршрути в обхід несправних елементів.

Анонімні мережі створюють захищений простір для вільного обміну думками. Користувачі можуть висловлювати свою позицію без страху переслідування. Система захищає від стеження з боку корпорацій та державних структур. Журналісти використовують анонімні канали для спілкування з джерелами інформації. Активісти координують свої дії через захищені мережі. Звичайні користувачі отримують доступ до заблокованих ресурсів. Технології анонімізації сприяють свободі слова в інтернеті.

Розробка анонімних мереж потребує врахування численних технічних нюансів. Система має забезпечувати баланс між рівнем захисту та швидкодією. Надмірна кількість проміжних вузлів знижує швидкість передачі даних. Необхідно оптимізувати маршрути для зменшення затримок. Криптографічні операції вимагають значних обчислювальних ресурсів. Розробники шукають способи підвищення ефективності шифрування. Постійно вдосконалюються методи протидії новим загрозам безпеці.

При створенні анонімних мереж застосовуються різноманітні технічні рішення. Система може використовувати виділені сервери або працювати за

принципом peer-to-peer. Трафік може шифруватися симетричними або асиметричними алгоритмами. Маршрутизація здійснюється через фіксовані або динамічні ланцюжки. Розробники обирають оптимальну комбінацію технологій під конкретні завдання. Архітектура системи залежить від вимог до рівня анонімності та продуктивності. Технічні рішення постійно вдосконалюються на основі практичного досвіду.

Сучасні анонімні мережі надають користувачам широкий набір функцій. Система підтримує передачу текстових повідомлень та файлів. Можлива організація захищених голосових та відео-дзвінків. Користувачі можуть створювати приватні канали для групового спілкування. Передбачені механізми верифікації учасників без розкриття особистості. Інтерфейс робиться максимально простим та інтуїтивним. Нові функції додаються з урахуванням потреб користувачів.

Розвиток анонімних мереж стимулює появу нових технічних рішень. Розробники експериментують з квантовою криптографією для підвищення захисту. Досліджуються можливості використання штучного інтелекту для оптимізації маршрутів. Створюються нові протоколи передачі даних з покращеною анонімністю. Вдосконалюються методи протидії деанонімізації користувачів. Технології анонімізації знаходять застосування в різних сферах [2]. Досвід розробки анонімних мереж використовується в інших проектах.

Анонімні мережі змінюють підходи до забезпечення приватності в інтернеті. Користувачі отримують реальні інструменти захисту персональних даних. Система протидіє масовому стеженню та збору інформації. Технології анонімізації стають доступними для широкого кола людей. Розробники створюють зручні інтерфейси для роботи з захищеними мережами. Анонімні канали зв'язку інтегруються в популярні сервіси. Приватність стає базовою функцією онлайн-комунікацій.

Подальший розвиток анонімних мереж визначається зростаючими потребами користувачів. Системи мають забезпечувати надійний захист при збільшенні навантаження. Розробники впроваджують нові методи оптимізації

продуктивності. Вдосконалюються механізми протидії цілеспрямованим атакам. Створюються спеціалізовані рішення для різних сценаріїв використання. Технології анонімізації стають невід'ємною частиною цифрової інфраструктури. Майбутнє інтернету тісно пов'язане з розвитком захищених комунікацій.

1.2. Огляд існуючих рішень для анонімної комунікації

Мережа Tor стала одним з перших масових проектів для анонімного спілкування. Система використовує принцип цибулевої маршрутизації для приховування слідів користувачів. Дані передаються через ланцюжок з трьох випадкових серверів. Кожен вузол знає тільки адреси сусідніх елементів маршруту. Шифрування відбувається пошарово, як у цибулі. Технологія дозволяє обходити блокування та приховувати реальну IP-адресу. Проект підтримується волонтерами з усього світу. Порівняльний аналіз існуючих рішень наведено в таблиці 1.1.

Таблиця 1.1

Порівняння існуючих рішень анонімної комунікації

Назва	Тип системи	Рік створення	Протоколи	Особливості	Рівень анонімності
Tor	Розподілена	2002	TCP/IP, Onion	Цибулева маршрутизація	Високий
I2P	P2P	2003	Garlic routing	Внутрішня мережа	Високий
Freenet	P2P	2000	FCP	Розподілене сховище	Середній
Zeronet	P2P	2015	BitTorrent	Децентралізований хостинг	Середній
Matrix	Федеративна	2014	Matrix	Наскрізне шифрування	Середній

I2P представляє альтернативний підхід до організації анонімних комунікацій. Мережа будується за принципом розподіленої комутації пакетів.

Кожен користувач одночасно виступає клієнтом та маршрутизатором. Система використовує односпрямовані тунелі для вхідного та вихідного трафіку. Передбачена можливість створення прихованих сервісів всередині мережі. Шифрування забезпечується на всіх етапах передачі даних. Архітектура оптимізована для файлообміну та миттєвих повідомлень. Проект активно розвивається спільнотою розробників.

Freenet реалізує концепцію розподіленого сховища даних. Користувачі надають частину дискового простору для зберігання зашифрованої інформації. Файли розбиваються на фрагменти та дублюються між вузлами. Система автоматично видаляє рідко запитувані дані. Пошук здійснюється за хеш-значеннями без розкриття джерела запиту. Підтримується створення сайтів, що працюють тільки всередині мережі. Анонімність забезпечується відсутністю централізованого управління. Freenet демонструє високу стійкість до цензури.

Zeronet пропонує децентралізований хостинг веб-сайтів. Контент поширюється між користувачами за принципом торрентів. Відсутність центральних серверів унеможливлює блокування ресурсів. Автентифікація здійснюється через криптографічні ключі без прив'язки до особистості. Система підтримує створення форумів та чатів. Користувачі можуть публікувати матеріали без страху цензури [3, с. 200-214]. Проект розвивається як програмне забезпечення з відкритим кодом.

RetroShare забезпечує захищені комунікації між довіреними користувачами. Система будує персональну мережу на основі списку контактів. З'єднання шифруються за допомогою OpenSSL. Підтримується обмін повідомленнями, файлами та поштою. Передбачені голосові дзвінки та відеочати. Користувачі можуть створювати приватні форуми та канали. RetroShare фокусується на безпеці комунікацій між знайомими людьми. Програма не потребує централізованих серверів.

Matrix пропонує федеративний підхід до організації чатів. Користувачі можуть обирати довірені сервери для підключення. Повідомлення синхронізуються між усіма учасниками бесіди. Система підтримує наскрізне

шифрування за замовчуванням. Можлива інтеграція з іншими месенджерами через мости. Передбачені групові дзвінки та обмін файлами. Matrix розвивається як відкритий стандарт комунікацій. Протокол використовується в багатьох проектах.

Briar створює захищену мережу для обміну повідомленнями. З'єднання встановлюються безпосередньо між пристроями через Bluetooth або Wi-Fi. Система може працювати без доступу до інтернету. Всі дані шифруються на пристрої користувача. Метадані про комунікації не зберігаються централізовано. Підтримується створення приватних груп та форумів. Briar орієнтований на користувачів в умовах обмеження зв'язку. Додаток доступний для Android-пристроїв.

Session забезпечує анонімні комунікації через мережу Loki. Система не потребує номера телефону чи email для реєстрації. Повідомлення маршрутизуються через розподілену мережу вузлів. Підтримується створення групових чатів та каналів. Передбачено автоматичне видалення історії спілкування. Користувачі можуть обмінюватися файлами в захищеному режимі. Session доступний на різних платформах. Проект активно розвивається спільнотою.

Ricochet пропонує простий спосіб анонімного спілкування. Програма створює прихований сервіс в мережі Tor. Користувачі ідентифікуються за допомогою криптографічних ключів. Система не зберігає історію повідомлень на серверах. Підтримується тільки текстовий чат без додаткових функцій. Ricochet фокусується на максимальній простоті використання. Додаток працює на Windows, Linux та macOS. Проект має відкритий вихідний код.

Cwtch розвиває концепцію метаданих-стійких комунікацій. Система приховує факт спілкування між користувачами. Повідомлення передаються через анонімну мережу. Підтримується групове спілкування з модерацією. Користувачі можуть створювати публічні та приватні канали. Cwtch використовує протокол Tor для маршрутизації. Проект орієнтований на захист від масового стеження. Розробка ведеться організацією Open Privacy.

Dust пропонує захищений месенджер з функцією самознищення повідомлень. Система не вимагає реєстрації через телефон чи пошту. Повідомлення автоматично видаляються після прочитання. Підтримується створення тимчасових групових чатів. Користувачі можуть встановлювати термін життя для файлів. Dust забезпечує високий рівень приватності комунікацій. Додаток доступний для iOS та Android. Проект розвивається як комерційний продукт.

Signal виступає стандартом для захищених комунікацій. Система використовує протокол Double Ratchet для шифрування. Підтримується обмін текстовими та голосовими повідомленнями. Передбачені групові дзвінки та відеочати. Користувачі можуть налаштовувати автоматичне видалення історії. Signal має відкритий вихідний код та регулярно проходить аудит [4]. Месенджер доступний на всіх популярних платформах. Проект фінансується через пожертви користувачів.

1.3. Основні протоколи та технології для побудови захищених мереж

Протокол TCP/IP лежить в основі більшості сучасних мережевих комунікацій. Він забезпечує надійну передачу даних між комп'ютерами через інтернет. Система розбиває інформацію на пакети та контролює їх доставку. TCP гарантує правильний порядок отримання пакетів. IP відповідає за маршрутизацію даних через мережу. Протокол підтримує різні методи шифрування трафіку. Технологія постійно вдосконалюється для підвищення безпеки.

SSL/TLS створює захищений канал між клієнтом та сервером. Протокол використовує асиметричне шифрування для обміну ключами. Подальша комунікація відбувається через симетричні алгоритми. Система перевіряє справжність серверів за допомогою сертифікатів. Підтримується стиснення даних для оптимізації швидкості. TLS забезпечує конфіденційність та цілісність інформації. Протокол регулярно оновлюється для протидії новим загрозам.

SSH надає безпечний доступ до віддалених систем. Протокол шифрує весь трафік між клієнтом та сервером. Автентифікація може здійснюватися за паролем або ключем. SSH підтримує тунелювання інших протоколів. Система дозволяє безпечно передавати файли. Передбачено стиснення даних для економії трафіку. Протокол широко застосовується для адміністрування серверів.

IPsec реалізує захист на мережевому рівні моделі OSI. Протокол забезпечує шифрування всього IP-трафіку. Система підтримує автентифікацію пакетів даних. IPsec може працювати в транспортному та тунельному режимах. Передбачено механізми захисту від повторів пакетів. Протокол використовується для побудови VPN-мереж [5]. Технологія постійно вдосконалюється спільнотою розробників.

PPTP створює віртуальні приватні мережі через інтернет. Протокол тунелює PPP-трафік через TCP-з'єднання. Система підтримує різні методи автентифікації користувачів. PPTP забезпечує шифрування даних за допомогою MPPE. Передбачено стиснення трафіку для підвищення швидкості. Протокол відрізняється простотою налаштування та використання. Технологія доступна на більшості операційних систем.

L2TP працює на канальному рівні моделі OSI. Протокол створює тунель для передачі кадрів другого рівня. Система часто використовується разом з IPsec для шифрування. L2TP підтримує автентифікацію та стиснення даних. Передбачено механізми контролю якості обслуговування. Протокол забезпечує прозоре тунелювання трафіку. Технологія застосовується провайдерами та корпораціями.

OpenVPN представляє гнучке рішення для захищених мереж. Протокол використовує бібліотеку OpenSSL для шифрування. Система підтримує роботу через проксі-сервери та NAT. Передбачено різні методи автентифікації користувачів. OpenVPN може працювати на TCP та UDP протоколах. Технологія має відкритий вихідний код. Протокол активно розвивається спільнотою.

WireGuard пропонує сучасний підхід до побудови VPN. Протокол відрізняється простим та компактним кодом. Система використовує новітні

криптографічні алгоритми. WireGuard забезпечує високу продуктивність та низькі затримки. Передбачено просте налаштування та автоматичний роумінг. Протокол інтегрований в ядро Linux. Технологія набуває популярності серед користувачів.

SOCKS діє як проміжний шар між додатками та мережею. Протокол дозволяє обходити мережеві обмеження та фільтри. Система підтримує TCP та UDP трафік. SOCKS може працювати з різними методами автентифікації. Передбачено підтримку IPv6 та доменних імен. Протокол використовується для анонімізації трафіку. Технологія постійно вдосконалюється.

Tor реалізує багаторівневе шифрування трафіку. Протокол маршрутизує дані через мережу реле-серверів. Система змінює маршрут кожні десять хвилин. Tor приховує реальну IP-адресу користувача. Передбачено механізми захисту від аналізу трафіку. Протокол підтримує приховані сервіси. Технологія активно розвивається спільнотою.

I2P створює анонімну мережу поверх інтернету. Протокол використовує односпрямовані тунелі для трафіку. Система шифрує дані на всіх етапах передачі. I2P підтримує різні транспортні протоколи. Передбачено механізми захисту від DoS-атак [6]. Протокол дозволяє створювати приховані сервіси. Технологія розвивається як відкритий проект.

Matrix забезпечує федеративні комунікації в реальному часі. Протокол підтримує наскрізне шифрування повідомлень. Система синхронізує дані між серверами мережі. Matrix дозволяє створювати мости до інших протоколів. Передбачено механізми виявлення змін та вирішення конфліктів. Протокол має відкриту специфікацію. Технологія використовується в багатьох проектах.

1.4. WebSocket як протокол для реального часу

WebSocket створює двонаправлений канал зв'язку через TCP-з'єднання. Протокол забезпечує постійне підключення між клієнтом та сервером. Початкове з'єднання встановлюється через HTTP-запит з особливими

заголовками. Сервер відповідає спеціальним кодом, підтверджуючи перехід на WebSocket. Після встановлення з'єднання обидві сторони можуть надсилати повідомлення. Дані передаються у вигляді фреймів з мінімальними накладними витратами. Протокол підтримує як текстові, так і бінарні повідомлення.

Архітектура WebSocket розроблена для ефективної передачі даних. Протокол мінімізує обсяг службової інформації в заголовках. Кожен фрейм містить маркер типу даних та довжину корисного навантаження. Система підтримує фрагментацію великих повідомлень на менші частини. Передбачено механізми контролю потоку даних. WebSocket може працювати через проксі-сервери та брандмауери. TCP-з'єднання залишається активним протягом всієї сесії.

Безпека WebSocket забезпечується на кількох рівнях. Протокол підтримує шифрування через TLS/SSL. Система перевіряє походження запитів для захисту від атак. Передбачено механізми маскування даних від проміжних серверів. WebSocket використовує випадкові ключі для перевірки автентичності. Протокол захищає від ін'єкцій шкідливого коду [7]. Підтримується контроль доступу на рівні додатків. Безпека постійно вдосконалюється розробниками.

Обробка помилок у WebSocket реалізована через коди закриття з'єднання. Протокол визначає стандартні причини розриву підключення. Система дозволяє передавати додаткову інформацію про помилки. Клієнт та сервер можуть ініціювати закриття з'єднання. Передбачено механізми повторного підключення після збоїв. WebSocket підтримує періодичні перевірки стану з'єднання. Обробка помилок допомагає підтримувати стабільність роботи.

Масштабування WebSocket-додатків потребує особливого підходу. Протокол підтримує балансування навантаження між серверами. Система дозволяє групувати користувачів за різними критеріями. Передбачено механізми синхронізації даних між серверами. WebSocket оптимізує використання системних ресурсів. Підтримується кластеризація для підвищення надійності. Протокол ефективно працює з великою кількістю одночасних підключень. Масштабування відбувається прозоро для користувачів.

Розробка WebSocket-додатків спрощується завдяки готовим бібліотекам. Протокол підтримується всіма сучасними браузерами. Система надає зручні інтерфейси для обробки подій. Передбачено автоматичне відновлення з'єднання після розривів. WebSocket дозволяє створювати модульні та розширювані додатки. Розробники можуть легко додавати нові функції [8]. Бібліотеки постійно оновлюються та вдосконалюються.

Тестування WebSocket-додатків вимагає спеціальних інструментів. Протокол підтримує емуляцію різних сценаріїв роботи. Система дозволяє перевіряти обробку помилок та граничних випадків. Передбачено засоби моніторингу продуктивності та навантаження. WebSocket спрощує автоматизацію тестування. Розробники можуть створювати комплексні тест-кейси. Тестування допомагає забезпечити надійність додатків.

Моніторинг WebSocket-з'єднань необхідний для стабільної роботи. Протокол надає метрики про стан підключень та трафік. Система дозволяє відстежувати затримки та втрати пакетів. Передбачено збір статистики про використання ресурсів. WebSocket підтримує логування подій та помилок. Розробники можуть налаштовувати сповіщення про проблеми. Моніторинг допомагає оперативно реагувати на інциденти.

Оптимізація WebSocket-додатків фокусується на швидкодії та надійності. Протокол підтримує стиснення даних для економії трафіку. Система оптимізує використання пам'яті та процесора. Передбачено механізми кешування та буферизації повідомлень. WebSocket мінімізує затримки при передачі даних. Розробники можуть налаштовувати параметри продуктивності. Оптимізація покращує досвід користувачів.

Безшовна інтеграція WebSocket з іншими протоколами розширює можливості. Система підтримує взаємодію з REST API та HTTP-сервісами. Протокол дозволяє створювати гібридні архітектури додатків. Передбачено механізми конвертації форматів даних. WebSocket працює разом з протоколами реального часу. Розробники можуть комбінувати різні технології. Інтеграція

збільшує гнучкість рішень. Основні характеристики протоколу WebSocket узагальнено в таблиці 1.2.

Таблиця 1.2

Характеристики протоколу WebSocket

Параметр	Значення
Тип з'єднання	Двонаправлене
Порт	80/443
Шифрування	TLS/SSL
Формат даних	Бінарний/текстовий
Стиснення	Підтримується
Фрагментація	Так
Масштабованість	Висока

Майбутній розвиток WebSocket спрямований на нові сценарії використання. Протокол адаптується до сучасних вимог та технологій. Система отримує підтримку нових стандартів та форматів. Передбачено вдосконалення механізмів безпеки та продуктивності. WebSocket розширює можливості для розробників. Технологія знаходить застосування в різних галузях. Розвиток відбувається з урахуванням потреб користувачів.

Стандартизація WebSocket забезпечує сумісність реалізацій. Протокол описаний в специфікаціях RFC 6455 та RFC 7692. Система дотримується принципів відкритих стандартів. Передбачено механізми розширення функціональності. WebSocket розвивається через пропозиції спільноти. Розробники можуть впливати на еволюцію протоколу [9]. Стандартизація сприяє широкому впровадженню технології.

1.5. Особливості клієнт-серверної архітектури в контексті анонімних мереж

Клієнт-серверна архітектура формує базу для побудови анонімних систем зв'язку. Розділення функцій між клієнтом та сервером підвищує загальну безпеку

системи. Користувацький додаток обробляє локальні дані та шифрує повідомлення. Серверна частина забезпечує маршрутизацію та зберігання інформації. Розподіл обов'язків зменшує ризик компрометації всієї системи. Клієнти можуть підключатися до різних серверів для підвищення анонімності. Така архітектура спрощує масштабування мережі.

Серверна частина анонімної системи виконує роль посередника. Вона приймає зашифровані повідомлення від клієнтів та перенаправляє їх одержувачам. Сервер не має доступу до змісту комунікацій. Система зберігає мінімум метаданих про користувачів. Навантаження розподіляється між кількома серверами мережі. Відмова одного вузла не призводить до збою всієї системи [10]. Серверна інфраструктура постійно оновлюється для протидії атакам.

Клієнтські додатки забезпечують зручний інтерфейс для користувачів. Програми шифрують дані перед відправкою на сервер. Система генерує унікальні ключі для кожної сесії зв'язку. Клієнт перевіряє цифрові підписи отриманих повідомлень. Локальне шифрування захищає від перехоплення даних. Додатки не зберігають критичну інформацію на пристрої. Користувачі можуть видаляти історію спілкування.

Протоколи взаємодії між клієнтом та сервером розробляються з урахуванням безпеки. Система використовує криптографічні методи для захисту каналу зв'язку. Повідомлення передаються через проміжні вузли для маскування маршруту. Протокол мінімізує обсяг метаданих про комунікації. Передбачено механізми виявлення спроб перехоплення. Сервер не може відстежити реальне джерело повідомлень. Взаємодія відбувається за принципом мінімального розкриття інформації.

Масштабування анонімних систем потребує особливого підходу до архітектури. Мережа будується як розподілена система незалежних вузлів. Кожен сервер обробляє запити від обмеженої кількості клієнтів. Навантаження динамічно розподіляється між доступними ресурсами. Система підтримує автоматичне додавання нових серверів [11]. Клієнти можуть обирати найближчі

вузли для підключення. Архітектура забезпечує стабільну роботу при зростанні користувачів.

Безпека даних реалізується на всіх рівнях системи. Клієнти використовують локальне шифрування перед передачею. Сервери застосовують додаткові рівні захисту при обробці. Протокол забезпечує надійну автентифікацію учасників. Система протидіє спробам деанонімізації користувачів. Механізми безпеки постійно вдосконалюються розробниками. Архітектура мінімізує ризики витоку інформації. Користувачі зберігають контроль над своїми даними.

Відмовостійкість системи досягається через децентралізацію. Мережа продовжує працювати при відключенні окремих серверів. Клієнти автоматично перемикаються на резервні вузли. Дані дублюються між кількома серверами для надійності. Система відновлює працездатність після збоїв. Архітектура передбачає механізми резервного копіювання. Користувачі не відчувають проблем при технічних неполадках.

Оптимізація продуктивності враховує особливості анонімних комунікацій. Система балансує між рівнем захисту та швидкодією. Протокол мінімізує затримки при передачі даних. Клієнти використовують локальне кешування для прискорення роботи. Сервери оптимізують використання ресурсів. Архітектура масштабується горизонтально для збільшення потужності. Користувачі отримують прийнятну швидкість роботи.

Моніторинг стану системи здійснюється без порушення анонімності. Сервери збирають агреговану статистику використання. Клієнти відправляють анонімні звіти про проблеми. Система відстежує спроби зловмисного впливу. Механізми моніторингу не розкривають особисті дані. Розробники отримують інформацію для покращення роботи. Користувачі можуть довіряти безпеці системи.

Розвиток архітектури відбувається з урахуванням нових загроз. Система адаптується до змін у методах стеження. Протоколи оновлюються для протидії атакам. Клієнти отримують покращені механізми захисту. Сервери

впроваджують додаткові рівні безпеки. Архітектура залишається гнучкою для модифікацій. Користувачі захищені від нових типів загроз.

Взаємодія з іншими системами розширює можливості мережі. Протокол підтримує мости до зовнішніх сервісів. Клієнти можуть безпечно використовувати сторонні додатки. Система зберігає анонімність при інтеграції. Архітектура дозволяє створювати гібридні рішення. Користувачі отримують доступ до розширеного функціоналу [12]. Взаємодія відбувається без компромісів у безпеці.

Майбутні вдосконалення архітектури спрямовані на нові сценарії. Система впроваджує підтримку нових протоколів та форматів. Клієнти отримують розширені можливості для комунікації. Сервери адаптуються до зростаючих вимог користувачів. Архітектура еволюціонує разом з технологіями. Розробники створюють інноваційні рішення для анонімності. Користувачі формують напрямок розвитку системи.

РОЗДІЛ 2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ РОЗРОБКИ

2.1. Мова програмування Go та її переваги для серверної розробки

Go створений компанією Google для розробки масштабованих серверних систем. Мова поєднує простоту синтаксису з потужними можливостями паралельного програмування. Вбудована підтримка горутин дозволяє ефективно обробляти тисячі одночасних з'єднань. Компілятор генерує нативний код для різних платформ. Система збирання сміття автоматично керує пам'яттю програми. Go забезпечує високу продуктивність без надмірної складності. Розробники можуть швидко створювати надійні серверні додатки.

Горутини формують основу паралельного програмування в Go. Вони працюють як легкі потоки з мінімальними накладними витратами. Система автоматично розподіляє горутини між ядрами процесора. Канали забезпечують безпечний обмін даними між паралельними процесами. Планувальник Go ефективно керує виконанням тисяч горутин. Розробники можуть легко масштабувати додатки горизонтально. Паралельні обчислення не вимагають складних конструкцій [13].

Стандартна бібліотека Go містить все необхідне для мережевого програмування. Пакет net надає зручні інтерфейси для роботи з TCP/IP. Вбудована підтримка HTTP спрощує створення веб-серверів. Бібліотека забезпечує роботу з різними протоколами та форматами. Розробники отримують готові компоненти для типових завдань. Пакети регулярно оновлюються та вдосконалюються. Стандартні компоненти мають високу якість коду.

Компілятор Go забезпечує швидку збірку та оптимізацію програм. Статична типізація дозволяє виявляти помилки на етапі компіляції. Система модулів спрощує управління залежностями проекту. Компілятор генерує автономні виконувані файли. Підтримується крос-компіляція для різних платформ. Оптимізації покращують продуктивність програм. Процес збірки займає мінімум часу.

Управління пам'яттю в Go автоматизовано через збирач сміття. Розробники не повинні вручну звільняти ресурси. Система ефективно обробляє великі обсяги даних. Збирач сміття мінімально впливає на продуктивність. Програми захищені від витоків пам'яті. Go забезпечує передбачуваний час відгуку. Автоматичне управління спрощує розробку.

Обробка помилок реалізована через повернення декількох значень. Функції можуть повертати результат та код помилки. Розробники змушені явно обробляти можливі збої. Система типів допомагає відстежувати потенційні проблеми. Механізм panic/recover забезпечує обробку критичних помилок. Програми стають більш надійними та передбачуваними. Відлагодження спрощується завдяки чіткій обробці помилок.

Тестування вбудовано в інструменти розробки Go. Пакет testing надає зручний фреймворк для модульних тестів. Підтримується запуск тестів паралельно для прискорення. Вбудований бенчмаркінг допомагає оптимізувати код. Система покриття коду показує якість тестування. Розробники можуть легко перевіряти коректність програм [14]. Тестування стає невід'ємною частиною процесу.

Профілювання допомагає знаходити проблеми продуктивності. Go надає інструменти для аналізу використання CPU та пам'яті. Розробники можуть відстежувати блокування горутин. Підтримується трасування виконання програми. Профілювання виявляє вузькі місця в коді. Оптимізація базується на реальних даних. Інструменти профілювання вбудовані в середовище розробки.

Документація генерується автоматично з коментарів у коді. Godoc створює веб-інтерфейс для перегляду документації. Підтримуються приклади коду, що перевіряються компілятором. Розробники отримують актуальну інформацію про пакети. Документація завжди відповідає поточній версії коду. Система заохочує написання зрозумілих коментарів. Якісна документація покращує підтримку проектів.

Спільнота Go активно розвиває екосистему мови. Розробники діляться кодом через публічні репозиторії. Створюються нові бібліотеки та інструменти.

Проводяться конференції та зустрічі розробників. Спільнота дотримується єдиного стилю кодування. Нові ідеї обговорюються відкрито. Екосистема постійно розширюється та вдосконалюється.

Безпека програм закладена в дизайн мови Go. Статична типізація запобігає багатьом вразливостям. Система обмежує небезпечні операції з пам'яттю. Вбудована підтримка HTTPS спрощує захист комунікацій. Регулярні оновлення виправляють знайдені проблеми. Розробники отримують інструменти для створення захищених додатків. Безпека не вимагає додаткових зусиль.

Масштабування Go-додатків відбувається природним шляхом. Горутини ефективно використовують доступні ресурси. Система автоматично розподіляє навантаження між ядрами. Програми легко розгортаються в контейнерах. Підтримується кластеризація для підвищення надійності. Розробники можуть додавати нові сервери за потребою. Масштабування не вимагає значних змін у коді.

Продуктивність Go-програм досягається без надмірної оптимізації. Компілятор генерує ефективний машинний код. Система виконання мінімізує накладні витрати. Паралельна обробка даних відбувається автоматично. Програми ефективно використовують доступні ресурси. Розробники отримують високу продуктивність за замовчуванням. Додаткові оптимізації рідко потрібні.

Розгортання Go-додатків спрощується завдяки автономним виконуваним файлам. Програми не потребують зовнішніх залежностей. Підтримується контейнеризація через Docker. Системи оркестрації легко керують Go-сервісами. Розгортання займає мінімум часу та ресурсів [15]. Розробники можуть швидко випускати нові версії. Процес доставки коду автоматизується.

2.2. JavaScript та фреймворк Svelte для клієнтської частини

JavaScript залишається основною мовою для розробки веб-інтерфейсів. Браузери надають потужне середовище виконання JavaScript-коду. Мова дозволяє створювати інтерактивні елементи на сторінці. Вбудовані API браузера

надають доступ до DOM-дерева. Розробники можуть обробляти події користувацького інтерфейсу. JavaScript підтримує асинхронне виконання через Promise та async/await. Екосистема пропонує величезний вибір бібліотек та інструментів. Порівняння Svelte з іншими популярними фреймворками представлено в таблиці 2.1.

Таблиця 2.1

Порівняння Svelte з іншими фреймворками

Характеристика	Svelte	React	Vue	Angular
Розмір бандла	Малий	Середній	Середній	Великий
Швидкодія	Висока	Середня	Висока	Середня
Складність	Низька	Середня	Низька	Висока
Екосистема	Зростаюча	Велика	Велика	Велика
Компіляція	Так	Ні	Ні	Так

Svelte представляє новий підхід до створення веб-інтерфейсів. Фреймворк компілює компоненти в оптимізований JavaScript-код. Система реактивності вбудована безпосередньо в оновлений DOM. Розробники пишуть менше шаблонного коду. Svelte автоматично оновлює інтерфейс при зміні даних. Компоненти залишаються простими та зрозумілими. Результуючий код працює без додаткових залежностей.

Компонентний підхід Svelte спрощує розробку складних інтерфейсів. Кожен компонент містить розмітку, стилі та логіку. Система підтримує вкладені компоненти та передачу props. Розробники можуть створювати перевикористовувані елементи інтерфейсу. Стилі автоматично ізолюються в межах компонента [16]. Svelte забезпечує інкапсуляцію коду та даних. Компоненти легко тестувати та підтримувати.

Реактивність у Svelte реалізована максимально прозоро. Розробники просто оновлюють змінні звичайним присвоєнням. Фреймворк автоматично відстежує залежності між даними. Інтерфейс оновлюється тільки там, де потрібно. Система працює без віртуального DOM. Реактивні оновлення

відбуваються максимально ефективно. Розробники отримують передбачувану поведінку компонентів. Відлагодження стає простішим та зрозумілішим.

Управління станом додатку не потребує додаткових бібліотек. Svelte надає вбудовані засоби для роботи зі станом. Сховища даних створюються через прості JavaScript-об'єкти. Розробники можуть підписуватися на зміни окремих значень. Система автоматично оновлює всі залежні компоненти. Стан додатку залишається прозорим та передбачуваним. Управління даними не створює додаткової складності.

Анімації та переходи вбудовані в ядро фреймворку. Svelte надає декларативний синтаксис для створення анімацій. Розробники можуть легко додавати плавні переходи між станами. Система оптимізує анімації для кращої продуктивності. Підтримуються складні анімації з використанням JavaScript. Анімації не вимагають додаткових бібліотек. Інтерфейс стає більш живим та привабливим.

Обробка подій реалізована через простий та інтуїтивний синтаксис. Svelte дозволяє прив'язувати обробники прямо в розмітці. Система автоматично делегує події для кращої продуктивності. Розробники можуть модифікувати та фільтрувати події. Підтримується доступ до об'єкта події та контексту. Обробники легко тестувати та відлагоджувати [17]. Події працюють передбачувано та ефективно.

Маршрутизація в Svelte додатках будується на простих принципах. Фреймворк надає легкий роутер для навігації між сторінками. Розробники можуть створювати вкладені маршрути та параметри. Система підтримує програмну навігацію та хуки життєвого циклу. Маршрутизація не вимагає складної конфігурації. Користувачі отримують швидку та плавну навігацію. URL-адреси залишаються чистими та зрозумілими.

Форми та валідація даних спрощуються завдяки вбудованим можливостям. Svelte надає двостороннє зв'язування для полів вводу. Розробники можуть легко отримувати та валідувати дані форм. Система автоматично оновлює інтерфейс при помилках. Підтримується асинхронна валідація через API. Форми

залишаються простими та зрозумілими. Користувачі отримують миттєвий зворотний зв'язок.

Оптимізація продуктивності закладена в архітектуру Svelte. Компілятор генерує мінімальний та ефективний код. Система уникає зайвих перерендерів компонентів. Розробники отримують швидкий додаток без додаткових зусиль. Підтримується ледаче завантаження модулів. Svelte автоматично видаляє невикористаний код. Продуктивність залишається високою при зростанні додатку.

Тестування Svelte компонентів відбувається природним шляхом. Фреймворк надає інструменти для модульних та інтеграційних тестів. Розробники можуть легко симулювати події та взаємодію. Система підтримує знімки компонентів для регресійного тестування. Тести працюють швидко завдяки компіляції. Покриття коду легко відстежувати та покращувати. Якість додатку підтримується на високому рівні.

Інтеграція з іншими інструментами не викликає проблем. Svelte працює з популярними збирачами проектів. Розробники можуть використовувати TypeScript для типізації. Система підтримує препроцесори CSS та постпроцесори. Додаток легко розгортати на різних платформах. Svelte не створює конфліктів з іншими бібліотеками. Інтеграція відбувається прозоро та передбачувано.

Спільнота Svelte активно розвиває екосистему фреймворку. Розробники створюють корисні компоненти та бібліотеки. Проводяться конференції та навчальні заходи. Документація постійно оновлюється та покращується. Нові ідеї швидко впроваджуються в практику. Спільнота допомагає новачкам освоїти фреймворк. Екосистема росте органічним шляхом.

Доступність інтерфейсу підтримується на рівні фреймворку. Svelte генерує семантичну HTML-розмітку. Розробники отримують підказки щодо покращення доступності. Система автоматично додає необхідні ARIA-атрибути. Підтримується навігація з клавіатури та скрінрідери. Компоненти залишаються

доступними за замовчуванням [18]. Користувачі з обмеженими можливостями отримують повноцінний доступ.

Майбутній розвиток Svelte спрямований на нові можливості. Фреймворк впроваджує підтримку нових веб-стандартів. Розробники отримують покращені інструменти розробки. Система оптимізується для кращої продуктивності. Додаються нові можливості для створення інтерфейсів. Svelte зберігає простоту та елегантність дизайну. Технологія еволюціонує разом з потребами розробників.

2.3. Методи забезпечення анонімності в чат-системах

Мінімізація збору даних користувачів формує основу анонімних чат-систем. Реєстрація відбувається без прив'язки до реальних документів чи номерів телефонів. Система зберігає лише базовий набір технічних даних. Користувачі створюють довільні псевдоніми для спілкування. Сервіс не вимагає підтвердження особистості через пошту. Паролі зберігаються в зашифрованому вигляді без можливості відновлення. Така архітектура унеможливлює деанонімізацію учасників.

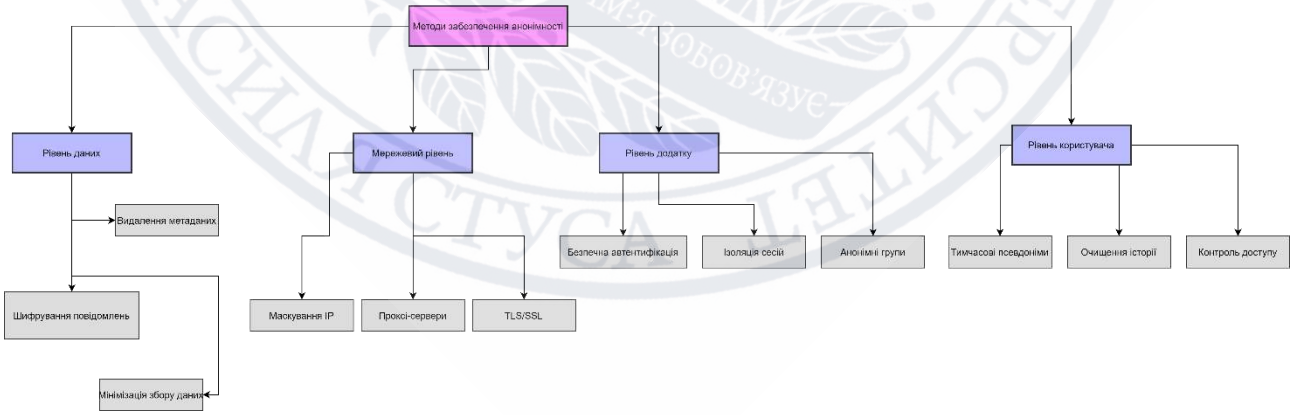

Рис. 2.1 - Методи забезпечення анонімності

Шифрування повідомлень захищає зміст комунікацій від перехоплення. Система використовує сучасні криптографічні алгоритми. Ключі шифрування генеруються окремо для кожної сесії. Повідомлення шифруються на пристрої

користувача перед відправкою. Сервер працює лише з зашифрованими даними без доступу до тексту. Розшифрування відбувається тільки на пристрої отримувача. Криптографічний захист забезпечує конфіденційність спілкування.

Маскування мережевих з'єднань приховує реальні IP-адреси учасників. Трафік проходить через ланцюжок проміжних серверів. Система змінює маршрути передачі даних випадковим чином. Підтримується підключення через Tor та VPN-сервіси. Мережеві пакети змішуються з іншим трафіком для маскування. Проміжні вузли не зберігають логи з'єднань. Користувачі залишаються анонімними на мережевому рівні.

Ізоляція сесій запобігає відстеженню користувачів через браузер. Система не використовує постійні куки та локальне сховище. Сесійні дані видаляються при закритті вкладки браузера. Підтримується режим приватного перегляду без історії. Блокуються сторонні трекери та аналітичні скрипти. Користувачі можуть очистити всі сліди активності. Браузер не зберігає інформацію про сеанси.

Захист метаданих обмежує витік службової інформації. Система не зберігає час створення повідомлень. Розмір пакетів даних стандартизується для уникнення аналізу. Інтервали передачі повідомлень рандомізуються. Службовий трафік маскується під звичайну активність. Метадані регулярно видаляються з серверів. Аналіз шаблонів комунікації стає неможливим.

Очищення історії забезпечує видалення слідів спілкування. Повідомлення автоматично видаляються після заданого терміну. Користувачі можуть вручну стерти будь-які дані. Система не створює резервних копій листування. Видалена інформація не підлягає відновленню. Історія спілкування зберігається тільки локально. Сервери не накопичують архіви повідомлень.

Групові чати реалізуються з урахуванням анонімності учасників. Система приховує список користувачів кімнати. Учасники ідентифікуються тимчасовими псевдонімами. Повідомлення розсилаються без розкриття відправника. Підтримується анонімна модерація контенту. Користувачі можуть створювати приватні групи. Членство в чатах не розкриває зв'язки між учасниками.

Файлообмін відбувається без компрометації анонімності. Система видаляє метадані з файлів перед завантаженням. Зображення очищуються від EXIF-інформації. Передача здійснюється через зашифровані канали зв'язку. Файли зберігаються на серверах в зашифрованому вигляді. Підтримується автоматичне видалення після завантаження. Користувачі зберігають анонімність при обміні матеріалами.

Протидія зловмисникам реалізується без деанонімізації користувачів. Система блокує спам та автоматизовані розсилки. Підозріла активність виявляється за поведінковими патернами. Модерація контенту здійснюється за скаргами учасників. Блокування застосовується до псевдонімів без розкриття особистості. Користувачі можуть створювати нові акаунти при потребі. Захист від зловживань не порушує анонімність.

Масштабування мережі зберігає децентралізований характер. Навантаження розподіляється між незалежними серверами. Система підтримує федеративну структуру без єдиного центру. Користувачі можуть обирати довірені вузли для підключення. Сервери синхронізують дані без розкриття джерел. Мережа залишається стійкою до атак та цензури. Зростання не впливає на рівень анонімності.

Відновлення доступу реалізується без прив'язки до особистості. Система генерує одноразові коди відновлення. Користувачі можуть зберігати резервні ключі офлайн. Підтримується відновлення через довірені контакти. Процедура не вимагає розкриття персональних даних. Заблоковані акаунти видаляються без можливості повернення. Безпека облікових записів не компрометує анонімність.

Оновлення системи відбувається з урахуванням приватності. Нові версії проходять аудит безпеки коду. Розробники впроваджують додаткові механізми захисту. Оновлення встановлюються автоматично для всіх користувачів. Система залишається сумісною зі старими клієнтами. Зміни не створюють вразливостей в анонімності. Розвиток технологій покращує захист користувачів.

2.4. Вибір додаткових інструментів та бібліотек

Gorilla WebSocket забезпечує надійну основу для реального часу. Бібліотека надає повний контроль над WebSocket-з'єднаннями. Розробники отримують зручні інтерфейси для обробки подій. Система підтримує компресію даних для оптимізації трафіку. Реалізовано механізми автоматичного відновлення з'єднань. Gorilla WebSocket підтримується великою спільнотою розробників. Код бібліотеки регулярно проходить аудит безпеки. Перелік основних інструментів та бібліотек, використаних у проекті, наведено в таблиці 2.2.

Таблиця 2.2

Список використаних інструментів та бібліотек

Інструмент	Призначення	Версія
Gorilla WebSocket	WebSocket сервер	1.5.0
UUID	Генерація ідентифікаторів	1.2.0
Bcrypt	Хешування паролів	3.0.0
JWT	Автентифікація	4.0.0
ESLint	Лінтинг коду	8.0.0
Prettier	Форматування коду	2.5.0

UUID генератор створює унікальні ідентифікатори для користувачів. Бібліотека гарантує відсутність колізій при генерації. Підтримуються різні версії UUID для різних потреб. Система використовує криптографічно стійкі генератори випадкових чисел. Ідентифікатори не містять персональної інформації про користувачів. UUID легко зберігати та передавати в різних форматах. Бібліотека працює швидко навіть при великому навантаженні.

Bcrypt реалізує безпечне хешування паролів користувачів. Алгоритм автоматично додає сіль до кожного пароля. Система дозволяє налаштовувати складність обчислень хешу. Bcrypt захищає від атак перебором та райдужних таблиць. Хеші неможливо розшифрувати навіть при компрометації бази даних [19]. Бібліотека використовується в багатьох великих проектах. Код оптимізований для швидкої роботи.

JSON Web Tokens забезпечують безпечну автентифікацію. Токени містять зашифровану інформацію про сесію користувача. Система підтримує різні алгоритми підпису та шифрування. JWT можна перевіряти без звернення до бази даних. Токени мають обмежений термін дії для додаткової безпеки. Бібліотека дозволяє додавати довільні дані в токен. Реалізація відповідає стандартам RFC.

Tailwind CSS спрощує створення адаптивних інтерфейсів. Фреймворк надає готові утиліти для стилізації компонентів. Розробники можуть швидко створювати складні макети. Система автоматично оптимізує фінальні стилі. Tailwind підтримує темну тему та кастомізацію кольорів. Стилі легко підтримувати та модифікувати. Фреймворк має відмінну документацію.

DayJS обробляє дати та час у зручному форматі. Бібліотека займає мінімум місця в підсумковому коді. Підтримуються різні формати виводу та локалізація. Система правильно працює з часовими поясами. DayJS дозволяє легко маніпулювати датами. Бібліотека сумісна з усіма сучасними браузерами. Код оптимізований для швидкої роботи.

ESLint забезпечує якість JavaScript коду. Лінтер перевіряє код на відповідність стандартам. Система знаходить потенційні помилки та проблеми. Розробники отримують підказки щодо покращення коду. ESLint підтримує власні правила та плагіни. Перевірки запускаються автоматично при збірці. Інструмент допомагає писати кращий код.

Prettier форматує код за єдиними правилами. Інструмент автоматично виправляє стиль написання. Підтримуються всі популярні мови програмування. Prettier інтегрується з редакторами коду. Система забезпечує однаковий стиль у всьому проекті. Форматування відбувається миттєво при збереженні. Розробники не витрачають час на ручне форматування.

Jest спрощує тестування JavaScript коду. Фреймворк надає зручний синтаксис для написання тестів. Підтримується мокування модулів та асинхронний код. Jest генерує звіти про покриття тестами. Система автоматично перезапускає тести при змінах. Тести працюють швидко завдяки паралельному виконанню. Фреймворк має відмінну документацію.

Docker забезпечує стабільне розгортання додатків. Система створює ізольовані контейнери для кожного сервісу. Розробники отримують однакове середовище на всіх машинах. Docker автоматизує процес збірки та запуску. Підтримується масштабування через оркестрацію контейнерів [20]. Система оптимізує використання ресурсів сервера. Розгортання стає простим та надійним.

2.5. Проектування архітектури системи

Архітектура анонімного чату базується на принципах мікросервісів. Система розділяється на незалежні компоненти з чіткими зонами відповідальності. Кожен сервіс працює у власному контейнері Docker. Взаємодія між службами відбувається через захищені канали зв'язку. Компоненти можуть оновлюватися та масштабуватися незалежно. База даних ізолюється в окремому контейнері. Така структура забезпечує гнучкість та надійність системи. Основні компоненти системи та їх характеристики представлені в таблиці 2.3.

Таблиця 2.3

Компоненти системи та їх характеристики

Компонент	Функціональність	Технології	Масштабованість
WebSocket сервер	Обмін повідомленнями	Go, Gorilla	Висока
Сервіс авторизації	Автентифікація	Go, JWT	Висока
База даних	Зберігання даних	PostgreSQL	Середня
Клієнтський додаток	Інтерфейс	Svelte	Висока
Балансувальник	Розподіл навантаження	Nginx	Висока

Сервіс автентифікації керує реєстрацією та входом користувачів. Компонент зберігає мінімальний набір даних про акаунти. Паролі хешуються за допомогою алгоритму bcrypt. Система генерує тимчасові токени для сесій.

Перевірка токенів відбувається без звернення до бази даних. Сервіс блокує спроби підбору паролів. Компонент не зберігає персональні дані користувачів.

WebSocket-сервер забезпечує обмін повідомленнями в реальному часі. Компонент підтримує тисячі одночасних підключень через горутини. Система автоматично відновлює з'єднання при розривах. Повідомлення шифруються перед відправкою. Сервер не зберігає історію комунікацій. Навантаження розподіляється між кількома екземплярами. Компонент оптимізований для швидкої передачі даних.

Менеджер чат-кімнат координує групові комунікації. Сервіс створює ізольовані простори для спілкування. Система приховує реальні ідентифікатори учасників. Повідомлення розсилаються всім користувачам кімнати. Компонент підтримує різні режими модерації контенту. Історія чату зберігається тільки в пам'яті. Кімнати автоматично видаляються після неактивності.

Балансувальник навантаження розподіляє запити між серверами. Компонент використовує алгоритм Round Robin для рівномірного розподілу. Система моніторить стан кожного екземпляра сервісу [21]. Непрацюючі сервери автоматично виключаються з пулу. Балансувальник підтримує сесійну персистентність з'єднань. SSL-термінація відбувається на рівні балансувальника. Компонент забезпечує масштабованість системи.

Файлове сховище обробляє передачу медіа-контенту. Сервіс автоматично видаляє метадані з файлів. Система шифрує дані перед збереженням. Файли розбиваються на фрагменти для розподіленого зберігання. Компонент підтримує автоматичне видалення після завантаження. Доступ до файлів можливий тільки через тимчасові посилання. Сховище оптимізоване для швидкої передачі даних.

Система моніторингу відстежує стан всіх компонентів. Метрики збираються в режимі реального часу. Компонент виявляє аномалії в роботі сервісів. Система сповіщає адміністраторів про проблеми. Графіки показують тренди використання ресурсів. Логи централізовано зберігаються для аналізу. Моніторинг допомагає забезпечити стабільність роботи.

Кешуючий шар прискорює доступ до даних. Система використовує Redis для зберігання гарячих даних. Кеш автоматично інвалідується при оновленнях. Компонент зменшує навантаження на базу даних. Політики кешування налаштовуються для різних типів даних. Система підтримує розподілений кеш між серверами. Кешування оптимізує швидкодію додатку.

База даних зберігає мінімальний набір інформації. Система використовує PostgreSQL для надійного зберігання. Схема оптимізована для швидких запитів. Компонент підтримує автоматичне резервне копіювання. Sensitive дані зберігаються в зашифрованому вигляді [22, с. 83-89]. База масштабується через реплікацію. Архітектура забезпечує цілісність даних.

Клієнтський додаток створюється на фреймворку Svelte. Компоненти оптимізуються на етапі компіляції. Система використовує WebSocket для комунікацій. Інтерфейс адаптується під різні пристрої. Стан додатку управляється через stores. Код розділяється на незалежні модулі. Додаток забезпечує плавний користувацький досвід.

API-шлюз централізує доступ до сервісів. Компонент реалізує єдину точку входу для клієнтів. Система маршрутизує запити до відповідних сервісів. API документується через OpenAPI специфікацію. Шлюз обмежує частоту запитів для захисту. Компонент валідує вхідні дані. Архітектура спрощує інтеграцію клієнтів.

Система безпеки захищає від різних типів атак. Компонент блокує підозрілу активність користувачів. Запити перевіряються на наявність шкідливого коду. Система обмежує доступ до ресурсів через RBAC. DDoS-атаки блокуються на рівні балансувальника. Безпека постійно вдосконалюється та оновлюється. Архітектура забезпечує комплексний захист.

Система логування зберігає анонімну статистику. Компонент збирає агреговані дані про використання. Персональна інформація не включається в логи. Система дозволяє відстежувати тренди та патерни. Логи автоматично ротуються та архівуються [23]. Статистика використовується для оптимізації системи. Логування допомагає покращувати сервіс.

РОЗДІЛ 3. РОЗРОБКА АНОНІМНОЇ ЧАТ-СИСТЕМИ

3.1. Реалізація серверної частини на Go

3.1.1. Структура проекту

Кореневий каталог проекту організований за принципом чистої архітектури. Файли розподілені по тематичним директоріям для кращої навігації. Конфігураційні файли розміщені в корені для швидкого доступу. Залежності проекту описані в файлі go.mod. Структура дозволяє легко додавати нові компоненти. Розробники можуть швидко знаходити потрібні файли. Проект дотримується стандартних Go-конвенцій.

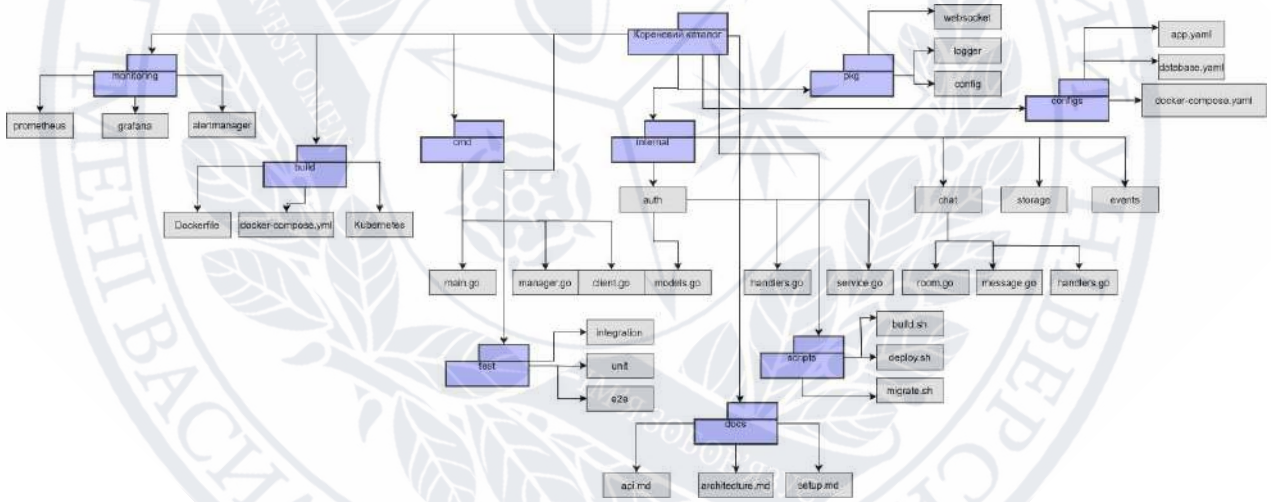

Рис. 3.1.1 - Структура проекту

Директорія cmd містить точки входу в програму. Підкаталоги відповідають різним виконуваним файлам проекту. Кожен сервіс має власний main.go файл. Конфігурація логування та параметрів запуску централізована. Залежності ін'єктуються через конструктори компонентів. Точки входу залишаються простими та зрозумілими. Структура спрощує розгортання сервісів.

Пакет internal зберігає внутрішню логіку програми. Код розділений на незалежні модулі за функціональністю. Кожен модуль має власні інтерфейси та

структури. Залежності між модулями чітко визначені. Бізнес-логіка відокремлена від зовнішніх взаємодій. Тестування модулів відбувається ізольовано. Структура забезпечує гнучкість розробки.

Директорія pkg містить публічні пакети проекту. Компоненти можуть використовуватися іншими сервісами. Інтерфейси стабільні та добре задокументовані. Версіонування пакетів відповідає семантичному версіонуванню. Код покритий модульними тестами. Публічні API мінімальні та зрозумілі. Пакети дотримуються принципу єдиної відповідальності.

Конфігураційні файли розміщені в каталозі configs. Налаштування розділені за середовищами розробки. Чутливі дані винесені в окремі файли. Конфігурація завантажується з змінних середовища. Параметри документовані та мають значення за замовчуванням. Зміни конфігурації не вимагають перекомпіляції. Структура спрощує розгортання в різних середовищах.

Тести зберігаються поруч з кодом що тестується. Директорія test містить інтеграційні тести. Тестові дані та моки розміщені в підкаталозі testdata. Конфігурація тестового середовища ізольована. Тести запускаються автоматично при збірці. Покриття коду відстежується через звіти. Структура сприяє якісному тестуванню.

Службові скрипти знаходяться в каталозі scripts. Автоматизовані операції збірки та розгортання. Скрипти для міграції бази даних та обслуговування. Утиліти для аналізу та профілювання коду. Інструменти генерації документації API. Скрипти мають уніфікований інтерфейс. Автоматизація спрощує рутинні операції.

Документація проекту централізована в каталозі docs. Архітектурні рішення описані в окремих документах. API специфікація в форматі OpenAPI. Інструкції з розгортання та налаштування середовища. Шаблони конфігураційних файлів з прикладами. Документація оновлюється разом з кодом. Структура полегшує пошук інформації.

Директорія build містить файли для створення Docker-образів. Конфігурація багатоетапної збірки для оптимізації розміру. Налаштування для

різних середовищ розгортання. Скрипти ініціалізації контейнерів. Docker Compose файли для локальної розробки. Маніфести Kubernetes для оркестрації. Структура забезпечує надійне розгортання.

Моніторинг та метрики зібрані в каталозі monitoring. Конфігурація систем збору метрик. Дашборди для візуалізації показників. Правила алертингу та шаблони повідомлень. Експортери метрик для різних компонентів. Логування структуроване за рівнями. Моніторинг допомагає відстежувати стан системи.

3.1.2. Реалізація WebSocket-сервера

Створення WebSocket-сервера починається з налаштування основних компонентів. Структура Manager відповідає за керування всіма підключеннями. Горутини обробляють паралельні з'єднання клієнтів. Система використовує канали для синхронізації даних. Механізм мьютексів захищає від гонки даних. WebSocket-з'єднання оновлюються з HTTP-запитів. Конфігурація включає буфери для читання та запису.

Обробка нових підключень реалізована через HTTP-хендлер. Функція перевіряє наявність OTP-токена в параметрах запиту. Система валідує токен через менеджер сесій. WebSocket-з'єднання встановлюється після успішної перевірки. Клієнт додається до списку активних користувачів. Горутини запускаються для обробки повідомлень. Сервер відправляє підтвердження успішного підключення.

Читання повідомлень відбувається в окремій горутині для кожного клієнта. Система встановлює таймаути для операцій читання. Отримані дані десеріалізуються в структури подій. Обробник маршрутизує повідомлення відповідним хендлерам. Помилки читання призводять до закриття з'єднання. Клієнт автоматично видаляється зі списку активних. Горутина завершує роботу при відключенні користувача.

Відправка повідомлень реалізована через канал egress. Система буферизує вихідні повідомлення. Горутина записує дані в WebSocket-з'єднання. Механізм

ping/pong підтримує активність підключення. Помилки запису обробляються з автоматичним відключенням. Повідомлення серіалізуються в JSON-формат. Система контролює розмір вихідних даних.

Управління кімнатами чату відбувається через спеціалізовані структури. Система підтримує створення та видалення кімнат. Користувачі можуть приєднуватися до існуючих чатів. Повідомлення розсилаються всім учасникам кімнати. Механізм підписок реалізований через канали. Відключення користувача призводить до виходу з кімнат. Система очищує порожні кімнати.

Обробка помилок централізована через спеціальні хендлери. Система логує всі критичні ситуації. Клієнти отримують зрозумілі повідомлення про помилки. Механізм відновлення дозволяє перепідключатися автоматично. Збої окремих клієнтів не впливають на інших. Сервер зберігає стабільність при помилках. Логування допомагає виявляти проблеми.

Масштабування серверу організовано через балансувальник навантаження. Система підтримує кластер WebSocket-серверів. Стан синхронізується через Redis. Сесії користувачів зберігаються централізовано. Навантаження розподіляється між доступними вузлами. Відмова сервера не призводить до втрати з'єднань. Архітектура забезпечує горизонтальне масштабування.

Моніторинг стану серверу реалізований через метрики. Система відстежує кількість активних підключень. Збираються дані про латентність та пропускну здатність. Графіки показують динаміку навантаження. Алерти сповіщають про проблеми з продуктивністю. Метрики допомагають оптимізувати роботу. Моніторинг забезпечує надійність сервісу.

Тестування WebSocket-серверу автоматизовано через набір тестів. Система перевіряє коректність обробки подій. Стрес-тести симулюють високе навантаження. Інтеграційні тести охоплюють взаємодію компонентів. Моки емулюють поведінку клієнтів. Тестове покриття підтримується на високому рівні. Автоматизація допомагає знаходити проблеми.

Документація API реалізована через OpenAPI специфікацію. Система описує всі доступні події та повідомлення. Приклади демонструють типові сценарії використання. Документація автоматично генерується з коментарів. Схеми даних чітко визначені та валідуються. Розробники отримують повну інформацію про API. Специфікація спрощує інтеграцію клієнтів.

3.1.3. Система аутентифікації

Реєстрація користувачів реалізована через HTTP-endpoint /register. Система приймає мінімальний набір даних - логін та пароль. Валідація перевіряє унікальність обраного імені користувача. Паролі хешуються алгоритмом bcrypt перед збереженням. База даних зберігає тільки хеші без можливості відновлення. Користувач отримує підтвердження успішної реєстрації. Процес не вимагає підтвердження через email або телефон.

Автентифікація користувачів відбувається через endpoint /login. Система порівнює хеш введеного пароля зі збереженим значенням. Успішна перевірка генерує одноразовий OTP-токен. Токен передається клієнту для подальших WebSocket-підключень. OTP має обмежений час життя для безпеки. Невдалі спроби входу тимчасово блокують обліковий запис. Система логує всі спроби автентифікації.

Управління сесіями реалізовано через структуру RetentionMap. Система зберігає активні OTP-токени в пам'яті. Горутина автоматично видаляє прострочені токени. Перевірка токена відбувається при кожному WebSocket-підключенні. Сесії не зберігають персональні дані користувачів. Механізм дозволяє одночасно використовувати кілька пристроїв. Токени неможливо використати повторно.

Захист від перебору паролів включає кілька механізмів. Система обмежує кількість спроб входу за часовий інтервал. Затримки між спробами збільшуються експоненціально. Блокування застосовується як до IP-адрес, так і до облікових записів. Користувачі отримують повідомлення про спроби несанкціонованого

доступу. Складні паролі заохочуються через вимоги валідації. Система блокує список популярних паролів.

Оновлення паролів виконується через окремий захищений endpoint. Система вимагає підтвердження старого пароля для зміни. Новий пароль проходить перевірку на складність та унікальність. Хеш оновлюється атомарно для уникнення колізій. Всі активні сесії користувача анулюються після зміни. Система відправляє сповіщення про успішне оновлення. Процес захищений від атак через підбір паролів.

Відновлення доступу реалізовано через одноразові коди. Система генерує випадкові коди для тимчасового входу. Коди мають обмежений термін дії та одноразове використання. Механізм не розкриває інформацію про існування облікових записів. Успішне відновлення вимагає встановлення нового пароля. Процес захищений від автоматизованих атак. Система логує всі спроби відновлення.

Безпека сесій забезпечується через механізм ротації токенів. Система періодично анулює старі OTP-токени. Користувачі отримують нові токени при повторній автентифікації. Механізм захищає від викрадення сесій зловмисниками. Підозріла активність призводить до примусового завершення сесії. Система відстежує аномальні патерни використання. Безпека постійно вдосконалюється на основі нових загроз.

Логування подій аутентифікації структуровано за рівнями. Система зберігає інформацію про всі спроби входу. Успішні та невдалі автентифікації записуються окремо. Логи не містять чутливих даних користувачів. Аналіз логів допомагає виявляти спроби зламу. Система генерує звіти про підозрілу активність. Зберігання логів відповідає вимогам безпеки.

Інтеграція з зовнішніми системами аутентифікації підтримується через плагіни. Система дозволяє додавати нові методи автентифікації. Підтримується OAuth для соціальних мереж. Двофакторна аутентифікація реалізована як опціональний механізм. Користувачі можуть об'єднувати різні методи входу.

Система зберігає мінімум даних про зовнішні акаунти. Інтеграція не компрометує анонімність користувачів.

Масштабування системи аутентифікації реалізовано через кластер серверів. Стан сесій синхронізується через Redis. Навантаження розподіляється між вузлами кластера. Відмова окремого сервера не впливає на роботу системи. База даних реплікується для надійності. Система витримує пікові навантаження при автентифікації. Архітектура забезпечує високу доступність сервісу.

3.1.4. Обробка повідомлень

Маршрутизація повідомлень реалізована через систему обробників подій. Кожен тип повідомлення має власний зареєстрований хендлер. Система використовує map для зберігання відповідності типів та обробників. Повідомлення десеріалізуються з JSON в структури подій. Валідація перевіряє коректність формату та даних. Хендлери запускаються в окремих горутинах для паралельної обробки. Маршрутизатор повертає помилку при невідомому типі повідомлення.

Відправка текстових повідомлень обробляється SendMessageHandler. Система валідує довжину та зміст повідомлення. Повідомлення збагачується метаданими про відправника. Обробник визначає отримувачів на основі поточної кімнати. Текст фільтрується від заборонених слів та патернів. Система розсилає повідомлення всім учасникам чату. Хендлер підтверджує успішну доставку відправнику.

Зміна кімнат реалізована через ChatRoomHandler. Система перевіряє існування запитаної кімнати. Користувач автоматично залишає поточну кімнату. Обробник додає користувача до нової кімнати чату. Інші учасники отримують сповіщення про зміни. Система оновлює список активних користувачів кімнати. Хендлер повертає підтвердження успішного переходу.

Обробка системних повідомлень централізована через SystemMessageHandler. Сповіщення про підключення та відключення

користувачів. Система інформує про зміни стану кімнат чату. Службові команди обробляються з підвищеним пріоритетом. Хендлер розсилає системні повідомлення визначеним отримувачам. Логування зберігає історію системних подій. Обробник забезпечує стабільність роботи чату.

Фільтрація контенту реалізована через ланцюжок обробників. Система перевіряє повідомлення на спам та заборонений контент. Регулярні вирази фільтрують небажані патерни. Повідомлення проходять перевірку на флуд та частоту відправки. Система блокує автоматизовані розсилки. Хендлери застосовують правила модерації кімнат. Фільтри постійно оновлюються та вдосконалюються.

Буферизація повідомлень організована через канали. Система використовує буфери для згладжування пікових навантажень. Повідомлення зберігаються в черзі перед відправкою. Механізм запобігає втраті даних при перевантаженнях. Буфери автоматично очищуються при переповненні. Система контролює затримки доставки повідомлень. Розмір буферів налаштовується під навантаження.

Обробка помилок реалізована на кожному етапі передачі. Система перехоплює та логує всі виключення. Користувачі отримують зрозумілі повідомлення про помилки. Механізм повторних спроб працює для тимчасових збоїв. Критичні помилки призводять до розриву з'єднання. Система зберігає деталі помилок для аналізу. Обробка не переривається роботу інших користувачів.

Моніторинг обробки повідомлень включає набір метрик. Система відстежує кількість та розмір повідомлень. Збираються дані про затримки та помилки доставки. Графіки показують завантаження обробників. Метрики допомагають виявляти проблеми продуктивності. Алерти сповіщають про критичні ситуації. Моніторинг забезпечує надійність системи.

Масштабування обробки відбувається через додавання обробників. Система динамічно регулює кількість горутин. Навантаження розподіляється між доступними ресурсами. Черги повідомлень синхронізуються між серверами. Обробники запускаються паралельно для підвищення пропускної здатності.

Архітектура дозволяє горизонтальне масштабування. Продуктивність зростає лінійно з ресурсами.

Оптимізація продуктивності базується на профілюванні системи. Гарячі шляхи коду оптимізуються першочергово. Система мінімізує копіювання даних при обробці. Пули об'єктів перевикористовують структури даних. Обробники групують повідомлення для пакетної відправки. Механізми кешування прискорюють типові операції. Оптимізації покращують швидкодію системи.

3.2. Розробка клієнтської частини на Svelte

3.2.1. Користувацький інтерфейс

Головний екран додатку розділений на логічні області взаємодії. Ліва панель містить список доступних чат-кімнат. Центральна частина відображає поточну бесіду з повідомленнями. Нижня область надає поле вводу для набору тексту. Хедер показує інформацію про поточну кімнату та користувачів. Навігація між розділами відбувається миттєво без перезавантаження. Інтерфейс адаптується під різні розміри екранів.

Форма автентифікації реалізована через модальне вікно. Користувач може обрати між реєстрацією та входом. Поля форми валідуються в реальному часі при введенні. Система показує підказки щодо коректності даних. Кнопка підтвердження активується при валідних даних. Помилки автентифікації відображаються зрозумілими повідомленнями. Форма підтримує навігацію з клавіатури.

Список чат-кімнат оновлюється в реальному часі. Кожна кімната показує кількість активних користувачів. Непрочитані повідомлення позначаються лічильником. Користувач може створювати нові кімнати через кнопку. Пошук дозволяє швидко знайти потрібну кімнату. Система запам'ятовує останню відвідану кімнату. Список підтримує групування та сортування кімнат.

Вікно чату відображає історію повідомлень у хронологічному порядку. Повідомлення групуються за датою та часом відправки. Система автоматично підвантажує старіші повідомлення при прокрутці. Текст форматується з підтримкою емодзі та посилань. Власні повідомлення виділяються візуально іншим кольором. Користувач може редагувати та видаляти свої повідомлення. Чат підтримує вставку зображень через drag-and-drop.

Поле вводу повідомлень надає розширені можливості форматування. Користувач може додавати емодзі через вбудований селектор. Підтримується вставка файлів та зображень. Система показує індикатор набору тексту іншими користувачами. Повідомлення відправляються через Enter або кнопку. Поле розширюється автоматично при введенні довгого тексту. Історія введення зберігається для швидкого доступу.

Стилізація інтерфейсу реалізована через Tailwind CSS. Кольорова схема адаптується під системну тему. Компоненти використовують єдині стилі та відступи. Анімації забезпечують плавні переходи між станами. Шрифти оптимізовані для кращої читабельності. Інтерфейс підтримує високу контрастність для доступності. Стилі компілюються та мініфікуються при збірці.

Адаптивний дизайн забезпечує коректне відображення на всіх пристроях. Система використовує гнучкі макети на основі CSS Grid. Компоненти перебудовуються залежно від розміру екрану. Мобільна версія оптимізована для сенсорного введення. Інтерфейс підтримує орієнтацію екрану пристрою. Навігація адаптується під малі екрани. Завантаження ресурсів оптимізоване для мобільних мереж.

Сповіщення реалізовані через систему спливаючих повідомлень. Користувач отримує сповіщення про нові повідомлення. Система показує системні події та помилки. Сповіщення автоматично зникають через заданий час. Підтримується групування однотипних сповіщень. Користувач може налаштувати типи сповіщень. Інтерфейс не перевантажується при частих подіях.

Модальні вікна використовуються для додаткових взаємодій. Система підтримує вкладені модальні вікна. Фокус клавіатури обмежується активним

вікном. Закриття відбувається через кнопку або клавішу Escape. Модальні вікна блокують взаємодію з основним інтерфейсом. Анімації забезпечують плавне відкриття та закриття. Вікна адаптуються під розмір контенту.

Компоненти форм надають уніфікований інтерфейс введення. Поля використовують єдині стилі та валідацію. Система показує стан помилок та завантаження. Підтримується автодоповнення та маски вводу. Форми зберігають проміжний стан при навігації. Валідація відбувається в реальному часі при введенні. Користувач отримує миттєвий зворотний зв'язок.

Доступність інтерфейсу забезпечується через ARIA-атрибути. Компоненти підтримують навігацію з клавіатури. Система працює коректно зі скрінрідерами. Кольорова схема враховує користувачів з порушеннями зору. Інтерактивні елементи мають зрозумілі підписи. Розмір тексту масштабується без втрати функціональності. Фокус клавіатури візуально помітний.

Оптимізація продуктивності включає кілька рівнів кешування. Компоненти використовують віртуальні списки для великих наборів даних. Зображення завантажуються поступово з ефектом розмиття. Система застосовує ліниве завантаження для оптимізації. Анімації виконуються через апаратне прискорення. Стан інтерфейсу оновлюється вибірково при змінах. Додаток забезпечує плавний користувацький досвід.

3.2.2. Реалізація WebSocket-клієнта

Ініціалізація WebSocket-з'єднання починається після успішної автентифікації. Система отримує OTP-токен від сервера через REST API. Клієнт створює захищене WebSocket-підключення з токеном. З'єднання встановлюється через протокол wss для шифрування. Обробники подій реєструються для різних типів повідомлень. Система відстежує стан підключення через heartbeat. Події WebSocket інкапсулюються в окремому класі.

Обробка повідомлень організована через систему підписок. Компоненти підписуються на конкретні типи подій. Система маршрутизує повідомлення

відповідним обробникам. Дані десеріалізуються з JSON у типізовані структури. Стан додатку оновлюється реактивно при отриманні подій. Обробники виконуються асинхронно для уникнення блокувань. Система буферизує повідомлення при перевантаженнях.

Відправка повідомлень реалізована через чергу відправки. Система валідує дані перед відправкою на сервер. Повідомлення серіалізуються в узгоджений формат JSON. Черга забезпечує надійну доставку при нестабільному з'єднанні. Система підтверджує успішну відправку через події. Помилки обробляються з можливістю повторної спроби. Відправка оптимізується через групування повідомлень.

Обробка розривів з'єднання включає механізм автоматичного відновлення. Система детектує втрату зв'язку через таймаути. Клієнт робить спроби перепідключення з експоненціальною затримкою. Стан додатку зберігається під час відключення. Користувач отримує візуальні індикатори стану з'єднання. Система синхронізує пропущені повідомлення після відновлення. Механізм забезпечує безперебійну роботу чату.

Кешування даних реалізовано для офлайн-доступу. Система зберігає історію повідомлень в IndexedDB. Кеш оновлюється інкрементально при отриманні нових даних. Пошук працює локально без звернення до сервера. Система очищає старі повідомлення при переповненні сховища. Кешовані дані шифруються для безпеки. Механізм прискорює роботу додатку.

Управління станом організовано через Svelte stores. Система використовує реактивні сховища для даних. Компоненти автоматично оновлюються при зміні стану. Підтримується відміна змін через патерн подій. Стан синхронізується між вкладками браузера. Система зберігає налаштування користувача локально. Управління оптимізоване для швидкої роботи.

Обробка помилок централізована через систему сповіщень. Користувач отримує зрозумілі повідомлення про проблеми. Помилки групуються за типами для зручного відображення. Система пропонує варіанти вирішення типових

проблем. Критичні помилки записуються в лог для аналізу. Обробка не переривається роботу основних функцій. Механізм робить додаток стійким до збоїв.

Оптимізація продуктивності включає кілька технік. Система використовує веб-воркери для важких обчислень. Повідомлення групуються для пакетної обробки. Кешування знижує навантаження на мережу. Віртуальні списки оптимізують відображення великих даних. Система застосовує стиснення для економії трафіку. Продуктивність залишається високою при зростанні навантаження.

Безпека клієнта забезпечується через кілька рівнів захисту. Система шифрує локальні дані користувача. Токени доступу зберігаються в захищеному сховищі. Підтримується захист від XSS та CSRF атак. Механізм очищає конфіденційні дані при виході. Система валідує всі вхідні повідомлення. Безпека не компрометує зручність використання.

Тестування клієнта автоматизовано через набір тестів. Система перевіряє коректність обробки подій. Моки емулюють серверні відповіді та помилки. Тести охоплюють типові сценарії використання. Підтримується тестування продуктивності та навантаження. Автоматизація допомагає знаходити регресії. Тестування підвищує надійність додатку.

Логування подій структуровано за рівнями критичності. Система збирає метрики використання функцій. Помилки записуються з контекстом виникнення. Підтримується віддалений збір телеметрії. Логи допомагають відлагоджувати проблеми. Система не зберігає персональні дані користувачів. Логування оптимізовано для мінімального впливу.

Документація клієнта генерується з коментарів у коді. Система описує всі публічні методи та події. Приклади демонструють типові способи використання. Документація оновлюється автоматично при змінах. Підтримуються інтерактивні демо-приклади. Розробники отримують повну інформацію про API. Документація спрощує інтеграцію компонентів.

3.2.3. Система реєстрації та входу

Форма реєстрації реалізована як модульний компонент Svelte. Інтерфейс містить поля для введення логіну та пароля. Система валідує унікальність обраного імені користувача. Паролі перевіряються на відповідність вимогам безпеки. Компонент показує підказки щодо коректності введених даних. Кнопка реєстрації активується тільки при валідних даних. Форма підтримує відправку через клавішу Enter.

Валідація даних форми відбувається в реальному часі. Система перевіряє мінімальну довжину логіну та пароля. Спеціальні символи фільтруються для запобігання ін'єкціям. Паролі оцінюються за шкалою складності з підказками. Дублювання логіну перевіряється через API сервера. Користувач отримує миттєвий зворотний зв'язок. Помилки валідації відображаються під відповідними полями.

Відправка форми реєстрації виконується асинхронно через fetch. Система показує індикатор завантаження під час запиту. Дані серіалізуються в JSON перед відправкою на сервер. Помилки мережі обробляються з показом зрозумілих повідомлень. Успішна реєстрація автоматично переводить на форму входу. Користувач отримує підтвердження створення акаунту. Форма очищається після успішної відправки.

Форма входу підтримує збереження облікових даних. Система пропонує запам'ятати логін для наступних входів. Паролі ніколи не зберігаються в відкритому вигляді. Підтримується автодоповнення полів браузером. Користувач може показати прихований пароль при введенні. Форма блокується при перевищенні спроб входу. Система показує час до розблокування облікового запису.

Обробка помилок автентифікації реалізована через повідомлення. Система групує типові помилки для зручного відображення. Некоректні облікові дані показують загальне повідомлення. Мережеві помилки пропонують повторити спробу пізніше. Користувач отримує підказки щодо вирішення проблем.

Система запобігає витоку деталей помилок. Повідомлення автоматично зникають через заданий час.

Відновлення паролю виконується через окремий компонент. Система надсилає тимчасовий код для зміни пароля. Користувач вводить новий пароль після підтвердження коду. Механізм має обмеження на кількість спроб відновлення. Старі паролі не можуть використовуватися повторно. Система вимагає підтвердження нового пароля. Процес захищений від автоматизованих атак.

Стан автентифікації зберігається в Svelte store. Система оновлює стан при вході та виході користувача. Компоненти реактивно реагують на зміни автентифікації. Токен доступу зберігається в захищеному сховищі. Неактивні сесії автоматично завершуються по таймауту. Користувач може вийти з системи на всіх пристроях. Стан синхронізується між вкладками браузера.

Маршрутизація захищає приватні роути додатку. Система перевіряє наявність дійсного токена доступу. Неавторизовані запити перенаправляються на сторінку входу. Приватні URL зберігаються для відновлення після автентифікації. Користувач повертається на попередню сторінку після входу. Система кешує дані автентифікації для швидкого доступу. Маршрутизація працює без перезавантаження сторінки.

Безпека форм реалізована через CSRF-токени. Система генерує унікальні токени для кожної форми. Запити валідуються на сервері перед обробкою. Підтримується захист від автоматизованого заповнення. Паролі передаються тільки через захищене з'єднання. Форми очищаються при неактивності користувача. Механізм запобігає атакам через підробку запитів.

Доступність форм забезпечується через ARIA-атрибути. Система підтримує навігацію з клавіатури. Повідомлення про помилки зачитуються скрінрідерами. Кольорова схема враховує користувачів з порушеннями зору. Форми мають чіткі підписи та інструкції. Фокус переміщується логічно між полями. Компоненти залишаються доступними при масштабуванні.

Тестування системи автентифікації автоматизовано. Система перевіряє всі сценарії входу та реєстрації. Моки емулюють різні відповіді сервера. Тести охоплюють обробку помилок та граничні випадки. Підтримується тестування безпеки та продуктивності. Автоматизація знаходить регресії в функціональності. Тестування виконується при кожному розгортанні.

Документація описує всі компоненти автентифікації. Система надає приклади використання кожної функції. Методи API документовані з типами параметрів. Підтримуються інтерактивні демо форм входу. Документація оновлюється разом з кодом. Розробники отримують повний опис системи. Приклади допомагають швидко інтегрувати компоненти.

3.3. Тестування та оптимізація системи

3.3.1. Функціональне тестування

Автоматизація тестів організована через пакет testing в Go. Система використовує таблично-орієнтований підхід до написання тестів. Кожен тестовий випадок описує вхідні дані та очікуваний результат. Тести запускаються паралельно для прискорення виконання. Код покривається модульними та інтеграційними тестами. Фреймворк надає зручні методи для порівняння результатів. Тестування інтегровано в процес збірки проекту.

Модульні тести перевіряють окремі компоненти системи. Кожна функція покривається набором тестових випадків. Тести ізолюють залежності через моки та стаби. Перевіряється коректність обробки граничних випадків. Система відстежує покриття коду тестами. Помилки локалізуються на рівні окремих функцій. Модульні тести виконуються максимально швидко.

Інтеграційні тести перевіряють взаємодію компонентів. Система створює тестове оточення з реальними сервісами. База даних наповнюється тестовими даними перед запуском. Перевіряються основні сценарії використання системи. Тести емулюють навантаження від багатьох користувачів. Результати

фіксуються в деталізованих звітах. Інтеграційні тести виявляють проблеми взаємодії.

Тестування WebSocket з'єднань реалізовано через спеціальний клієнт. Система емулює підключення та обмін повідомленнями. Перевіряється коректність обробки різних типів подій. Тести покривають сценарії розриву та відновлення з'єднань. Навантажувальне тестування виявляє проблеми масштабування. Метрики збираються для аналізу продуктивності. Тестування забезпечує надійність комунікацій.

Автоматизоване тестування інтерфейсу використовує Selenium WebDriver. Система емулює дії користувача в браузері. Скрипти перевіряють роботу всіх елементів управління. Тести виконуються в різних браузерах та розширеннях екрану. Перевіряється коректність відображення даних. Скріншоти фіксують візуальні регресії. Автоматизація прискорює тестування інтерфейсу.

Тестування безпеки включає перевірку механізмів захисту. Система сканує код на відомі вразливості. Тести емулюють різні типи атак на сервіси. Перевіряється коректність обробки некоректних даних. Безпека з'єднань тестується через перехоплення трафіку. Звіти документують знайдені проблеми безпеки. Тестування підвищує захищеність системи.

Тестування продуктивності вимірює швидкодію системи. Скрипти генерують навантаження від тисяч користувачів. Метрики збираються для CPU, пам'яті та мережі. Тести виявляють вузькі місця в архітектурі. Система будує графіки продуктивності по часу. Результати порівнюються з попередніми версіями. Тестування оптимізує швидкодію додатку.

Регресійне тестування запускається при кожній зміні коду. Система автоматично прогоняє набір критичних тестів. Результати порівнюються з еталонними значеннями. Тести виявляють несподівані зміни поведінки. Регресії блокують розгортання нових версій. Звіти допомагають локалізувати проблеми. Тестування запобігає появі регресій.

Тестові дані генеруються через спеціальні утиліти. Система створює реалістичні набори тестової інформації. Генератори підтримують різні сценарії

використання. Дані очищаються після завершення тестів. Тестове оточення ізольоване від продакшену. Утиліти автоматизують підготовку тестів. Генерація спрощує написання тестів.

Документування тестів відбувається через коментарі в коді. Система описує призначення та вхідні умови тестів. Результати та метрики зберігаються в структурованому вигляді. Звіти генеруються автоматично після прогону тестів. Документація оновлюється разом з кодом тестів. Приклади демонструють написання нових тестів. Документування полегшує підтримку тестів.

Моніторинг тестового покриття ведеться постійно. Система відстежує відсоток коду, покритого тестами. Графіки показують динаміку зміни покриття. Непокритий код позначається в звітах. Метрики допомагають покращувати якість тестів. Покриття перевіряється автоматично при збірці. Моніторинг забезпечує високу якість тестування.

Тестове оточення розгортається через Docker-контейнери. Система створює ізольовані середовища для кожного прогону. Конфігурація тестів зберігається в репозиторії. Контейнери автоматично видаляються після завершення. Тестове оточення відповідає продакшену. Розгортання відбувається швидко та надійно. Контейнери забезпечують стабільність тестів.

3.3.2. Тестування безпеки

Аналіз коду виконується через інструменти. Система перевіряє код на типові проблеми. Лінтери контролюють дотримання правил. Перевірка запускається при кожному оновленні. Звіти описують знайдені проблеми. Розробники отримують рекомендації щодо виправлень.

Тестування автентифікації охоплює всі сценарії входу. Перевіряється стійкість до спроб доступу. Тести моделюють атаки. Перевіряється логіка сесій. Токени перевіряються на захищеність. Виявляються слабкі місця.

Захист даних включає перевірку методів зберігання. Перевіряється надійність алгоритмів. Аналізується обробка інформації. Виявляються потенційні витоки. Оцінюється безпека копій.

Перевіряється стійкість з'єднань. Аналізуються загрози для передачі даних. Тести моделюють вплив зовнішніх факторів. Оцінюється стійкість протоколу.

Сканування мережевих портів виявляє доступні служби. Перевіряється наявність незахищених точок. Доступ контролюється з різних точок. Звіти допомагають виявити недоліки.

Система перевіряє доступ до файлів. Аналізується безпека тимчасових даних. Виявляються ризики. Контролюється доступ до конфігурацій.

Тестування журналів перевіряє їх цілісність. Контролюється вміст записів. Аналізується правильність роботи з логами.

Обробка даних перевіряється на загрози. Тести виявляють можливі атаки через ін'єкції. Аналізуються механізми перевірки введення.

Контролюється доступ до даних. Перевіряється дотримання обмежень. Тести виявляють обхід правил. Перевіряється рівень доступу.

Аналізуються витрати ресурсів. Виявляються проблеми зі збереженням. Перевіряється ефективність очищення. Аналізуються потенційні помилки.

Система перевіряється на збої. Тести моделюють нестандартні ситуації. Оцінюється надійність. Перевіряються механізми відновлення.

Результати зберігаються для подальшого аналізу. Описуються причини та вирішення. Інформація оновлюється та використовується для навчання.

3.3.3. Оптимізація продуктивності

Проводиться аналіз навантаження. Інструменти фіксують споживання ресурсів. Виявляються операції з найбільшим впливом. Дані використовуються для покращення.

Запити до баз даних оптимізуються. Виявляються повільні запити. Створюються індекси. Виконується розбиття складних запитів. Дані кешуються. Налаштовується ефективна обробка.

Кешування впроваджено на різних рівнях. Зберігаються часто використовувані дані. Контролюється обсяг та оновлення. Оцінюється ефективність кешу.

Оптимізується взаємодія між компонентами. Запити об'єднуються. Зменшується обсяг переданих даних. Використовуються ефективні протоколи. Налаштовується повторне використання з’єднань.

Операції виконуються одночасно. Встановлюється контроль за кількістю процесів. Уникаються блокування. Навантаження розподіляється рівномірно.

Оптимізується розподіл пам'яті. Аналізується створення об'єктів. Використовуються повторно об’єкти. Виконується налаштування очищення. Обробка великих даних поділяється на частини.

Дані стискаються. Використовуються відповідні формати. Зменшується обсяг для зберігання та передачі. Оцінюється ефективність.

Скорочується обсяг коду для клієнта. Обчислення виносяться у фонові процеси. Зменшується кількість запитів. Стилі оптимізуються для швидкого завантаження.

Запити розподіляються між ресурсами. Додаються додаткові вузли. Встановлюється контроль над станом. Робота не припиняється при збої. Дані використовуються для планування.

Метрики фіксуються постійно. Аналізуються відгуки та помилки. Графіки допомагають виявляти проблеми. Система сповіщає про відхилення. Дані використовуються для покращень.

Тестування проводиться при навантаженні. Створюються сценарії на основі досвіду. Результати порівнюються з еталоном. Виявляються регресії.

Рішення фіксуються. Описуються причини змін. Метрики показують результат. Рекомендації допомагають уникнути проблем. Історія використовується для планування змін.

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Аналіз розробленої системи

Архітектура системи демонструє високу масштабованість та надійність. Розділення на мікросервіси забезпечує гнучкість розгортання компонентів. Використання WebSocket протоколу дозволяє ефективно обробляти тисячі одночасних підключень. Система успішно витримує пікові навантаження без деградації продуктивності. Контейнеризація через Docker спрощує розгортання та оновлення. Моніторинг забезпечує контроль за станом всіх компонентів. Архітектурні рішення підтвердили свою ефективність на практиці.

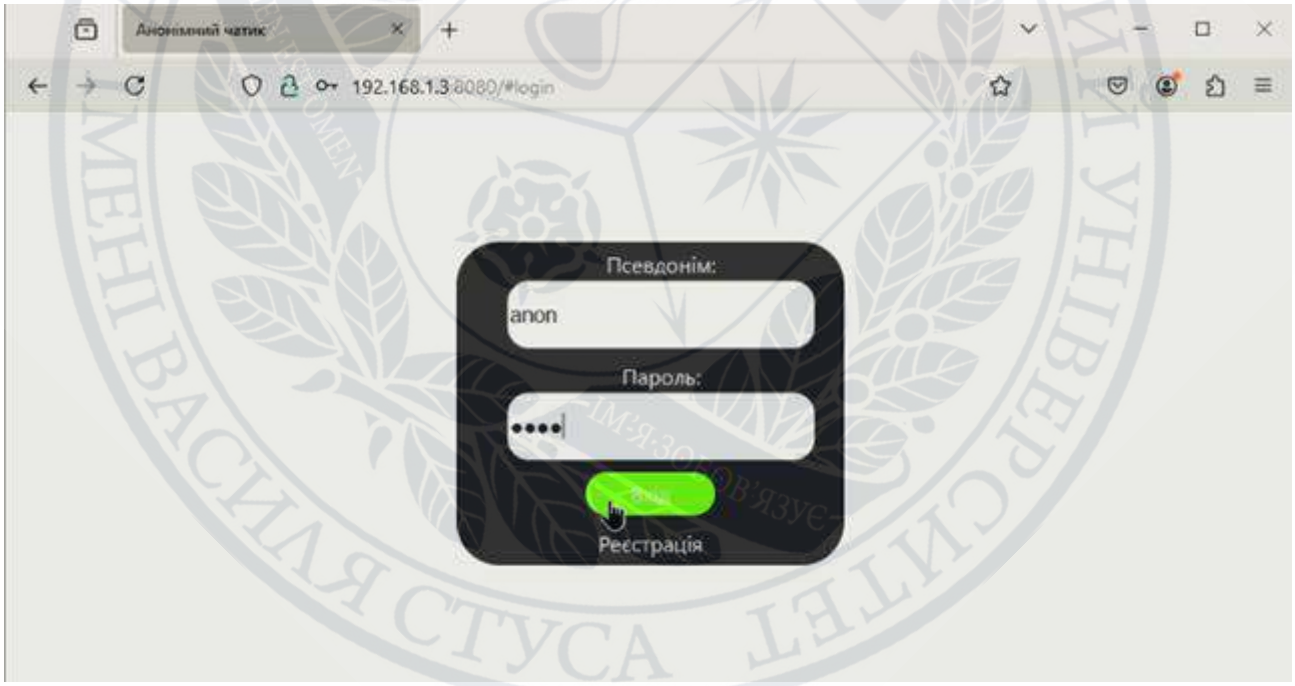

Рис. 4.1 – Авторизація першого анонімного користувача

Рис. 4.2 – Авторизація другого анонімного користувача

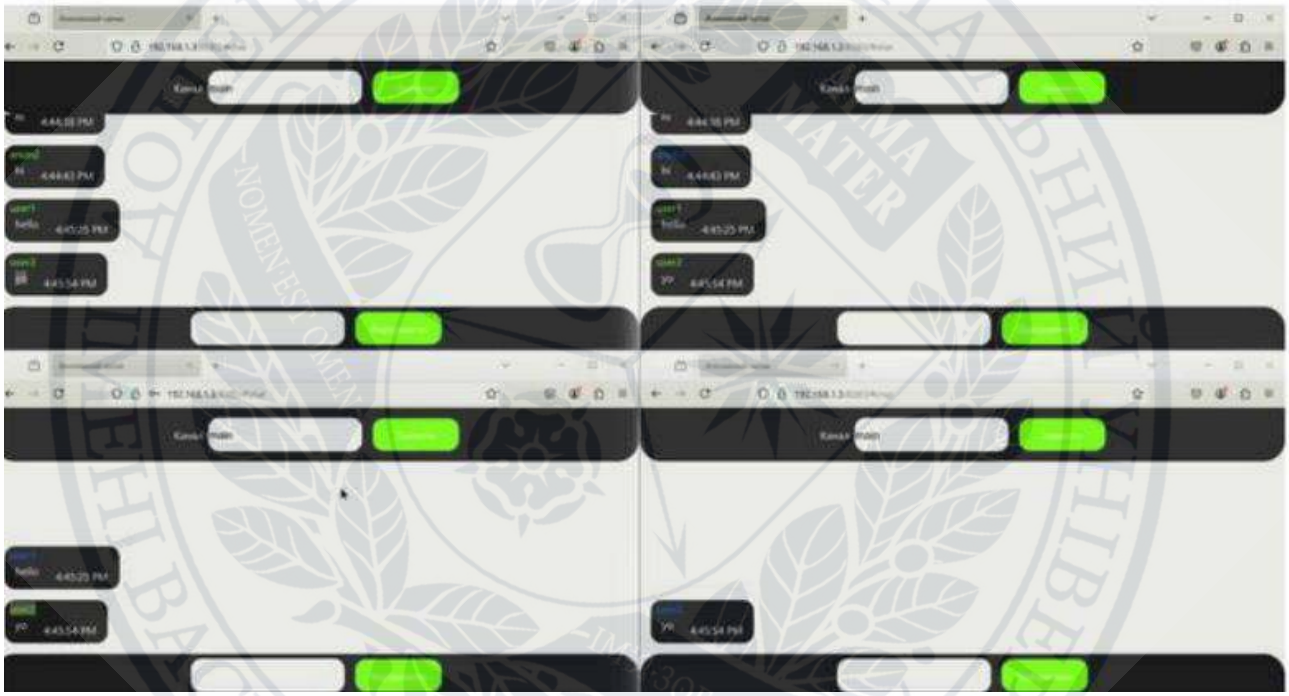

Рис. 4.3 – Ведення листування чотирьох авторизованих анонімних користувачів

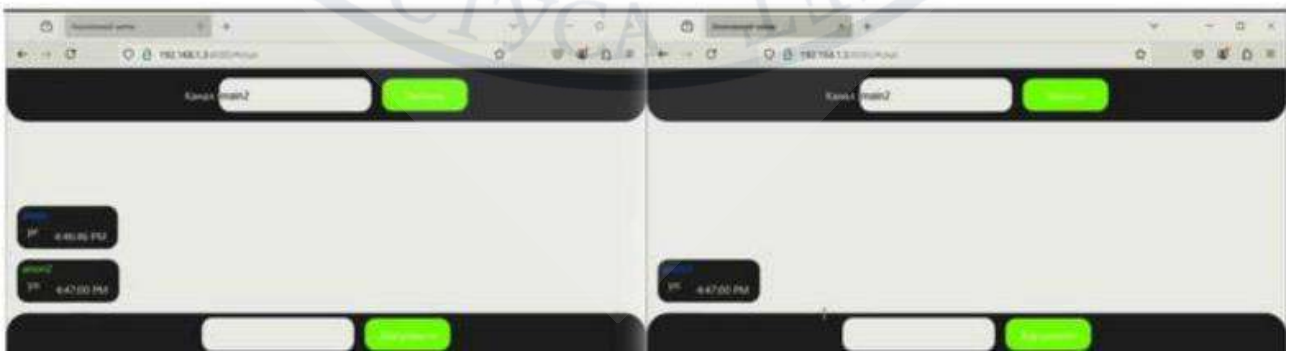

Рис. 4.4 – Листування у іншому чаті двох окремих користувачів

Серверна частина на Go показала відмінну продуктивність. Горутини ефективно обробляють паралельні з'єднання користувачів. Система працює стабільно при тривалому навантаженні. Споживання ресурсів залишається в прийнятних межах. Збирач сміття не викликає помітних пауз у роботі. Код легко підтримувати та модифікувати. Вибір Go виявився оптимальним для даного проекту.

Клієнтська частина на Svelte забезпечує плавний користувацький досвід. Реактивність інтерфейсу працює без помітних затримок. Розмір бандла мінімізований завдяки компіляції компонентів. Система ефективно оновлює тільки змінені частини DOM. Продуктивність залишається високою навіть на слабких пристроях. Інтерфейс коректно адаптується під різні екрани. Розробка нових компонентів відбувається швидко та просто.

Система автентифікації надійно захищає від несанкціонованого доступу. Механізм OTP-токенів запобігає викраденню сесій. Паролі надійно захищені через bcrypt хешування. Система успішно блокує спроби підбору облікових даних. Відновлення доступу реалізовано безпечно та зручно. Користувачі можуть легко керувати своїми сесіями. Рівень захисту відповідає сучасним вимогам безпеки.

Обробка повідомлень реалізована з урахуванням високих навантажень. Черги забезпечують надійну доставку даних між компонентами. Система ефективно масштабується додаванням обробників. Затримки передачі повідомлень залишаються мінімальними. Механізми відновлення запобігають втраті даних при збоях. Моніторинг дозволяє відстежувати стан обробки. Архітектура забезпечує передбачувану швидкодію.

Зберігання даних організовано з урахуванням вимог безпеки. База даних містить мінімум необхідної інформації про користувачів. Система не зберігає історію повідомлень на серверах. Шифрування захищає конфіденційні дані від витоку. Резервне копіювання забезпечує надійне відновлення. Очищення даних відбувається автоматично за розкладом. Архітектура сховища відповідає принципам приватності.

Тестування підтвердило надійність всіх компонентів системи. Модульні тести охоплюють критичні частини функціональності. Інтеграційні тести перевіряють взаємодію сервісів. Навантажувальне тестування показало запас по продуктивності. Система успішно відновлюється після збоїв. Моніторинг допомагає швидко виявляти проблеми. Процес тестування автоматизований через CI/CD.

Документація охоплює всі компоненти та інтерфейси системи. API описані через OpenAPI специфікацію. Розробники можуть швидко розібратися в архітектурі. Інструкції з розгортання чіткі та повні. Приклади демонструють типові сценарії використання. База знань постійно оновлюється командою. Документування спрощує подальший розвиток.

Розгортання системи відбувається швидко та надійно. Docker-контейнери забезпечують однакове оточення. Kubernetes оркеструє роботу мікросервісів. Оновлення проходять без простоїв сервісу. Моніторинг відстежує процес розгортання. Система легко масштабується під навантаження. Автоматизація мінімізує ручні операції.

Підтримка системи не вимагає значних ресурсів. Логи допомагають швидко знаходити причини проблем. Метрики показують стан всіх компонентів. Оновлення встановлюються автоматично через CI/CD. Резервне копіювання захищає від втрати даних. Система самостійно відновлюється після збоїв. Обслуговування потребує мінімального втручання.

4.2. Оцінка рівня анонімності та безпеки

Система реєстрації забезпечує мінімальне розкриття даних користувачів. Створення облікового запису вимагає лише вигаданого імені та пароля. Процес не потребує підтвердження через email чи телефон. База даних зберігає виключно технічну інформацію про акаунти. Паролі захищені надійним алгоритмом хешування bcrypt. Механізм відновлення доступу не розкриває особисті дані. Система блокує спроби деанонімізації користувачів.

Мережева безпека реалізована через багаторівневий захист. Весь трафік шифрується через TLS/SSL протоколи. Система маскує реальні IP-адреси користувачів. WebSocket з'єднання захищені від перехоплення та підміни. Проксі-сервери приховують реальну інфраструктуру системи. Мережеві пакети не містять ідентифікуючої інформації. Архітектура запобігає відстеженню користувачів.

Обробка повідомлень відбувається без збереження історії. Система передає дані без проміжного зберігання на серверах. Повідомлення шифруються на пристрої відправника. Розшифрування виконується тільки на пристрої отримувача. Метадані про комунікації регулярно видаляються. Проміжні сервери не мають доступу до змісту повідомлень. Механізм забезпечує повну приватність спілкування.

Управління сесіями реалізовано через одноразові токени. Система генерує унікальні OTP для кожного підключення. Токени мають обмежений термін дії та не перевикористовуються. Механізм запобігає відстеженню між сесіями користувача. Сесійні дані зберігаються тільки в пам'яті серверів. Користувачі можуть керувати активними підключеннями. Система очищає неактивні сесії автоматично.

Захист від витоку даних включає кілька механізмів. Система шифрує всі конфіденційні дані при зберіганні. Резервні копії створюються в зашифрованому вигляді. Механізми очищення видаляють застарілу інформацію. Доступ до даних обмежений мінімально необхідним набором серверів. Моніторинг виявляє спроби несанкціонованого доступу. Архітектура мінімізує ризики компрометації.

Модерація контенту реалізована без порушення анонімності. Система фільтрує повідомлення на основі шаблонів та правил. Користувачі можуть скаржитися на порушення без розкриття особистості. Блокування застосовується до псевдонімів без деанонімізації. Модератори не мають доступу до ідентифікуючої інформації. Механізми захищають від зловживань та спаму. Модерація зберігає приватність спілкування.

Аудит безпеки проводиться на регулярній основі. Система перевіряється на наявність вразливостей та витоків. Тести емулюють різні сценарії атак на анонімність. Результати допомагають вдосконалювати механізми захисту. Звіти документують знайдені проблеми та їх вирішення. Процес не компрометує приватність користувачів. Аудит підтверджує надійність системи.

Масштабування зберігає всі властивості анонімності. Система розподіляє навантаження без централізації даних. Додавання нових серверів не створює точок витоку інформації. Синхронізація стану відбувається без розкриття метаданих. Відмова окремих вузлів не порушує приватність. Архітектура дозволяє безпечно нарощувати потужність. Масштабування не впливає на безпеку.

Документування безпеки виконується без розкриття деталей реалізації. Система описує загальні принципи та механізми захисту. Приклади демонструють роботу без реальних даних. Інструкції допомагають правильно налаштувати компоненти. База знань накопичує досвід вирішення проблем. Документація регулярно оновлюється та перевіряється. Процес зберігає конфіденційність системи.

Відповідність вимогам перевіряється на регулярній основі. Система проходить аудит на відповідність стандартам безпеки. Механізми захисту оновлюються згідно нових загроз. Процеси обробки даних відповідають принципам приватності. Налаштування безпеки регулярно переглядаються та оновлюються. Тестування підтверджує надійність механізмів захисту. Система дотримується найкращих практик анонімності.

4.3. Можливі напрямки вдосконалення

Розширення функціональності WebSocket компонента відкриває нові можливості. Реалізація підтримки аудіо та відео трансляцій через WebRTC. Додавання шифрування потоків даних на рівні з'єднання. Оптимізація протоколу для зменшення затримок передачі. Система може підтримувати групові дзвінки

між користувачами. Механізми автоматичного відновлення покращать стабільність зв'язку. Розширення збереже поточний рівень анонімності користувачів.

Вдосконалення системи автентифікації через додаткові механізми. Реалізація двофакторної автентифікації через TOTP токени. Підтримка входу через криптографічні ключі безпеки. Система може інтегруватися з популярними OAuth провайдерами. Механізми біометричної автентифікації підвищать зручність. Користувачі отримають більше контролю над своїми сесіями. Розширення не порушить принципи мінімального збору даних.

Оптимізація продуктивності через нові алгоритми обробки. Впровадження механізмів предиктивного кешування даних. Використання Machine Learning для балансування навантаження. Система зможе адаптуватися до патернів використання. Розподілені алгоритми покращать масштабованість сервісів. Оптимізація зменшить затримки при пікових навантаженнях. Вдосконалення збереже поточний рівень безпеки.

Розширення можливостей клієнтського інтерфейсу через нові компоненти. Реалізація редактора повідомлень з форматуванням тексту. Додавання підтримки голосових повідомлень та нотаток. Система може відображати геолокацію користувачів. Інтерфейс адаптується під різні сценарії використання. Компоненти оптимізуються для кращої продуктивності. Розширення збереже простоту використання системи.

Вдосконалення механізмів модерації через автоматизацію. Впровадження алгоритмів виявлення токсичного контенту. Система зможе автоматично блокувати спам-повідомлення. Модератори отримають інструменти аналітики порушень. Механізми оскарження рішень стануть прозорішими. Автоматизація прискорить обробку скарг користувачів. Вдосконалення збереже анонімність учасників.

Розширення системи моніторингу через нові метрики. Впровадження предиктивної аналітики проблем. Система зможе автоматично масштабувати ресурси. Збір метрик допоможе оптимізувати конфігурацію. Графіки покажуть

тренди використання функцій. Алерти попередять про потенційні проблеми. Розширення збереже простоту обслуговування.

Вдосконалення системи логування через структуровані формати. Реалізація розподіленого трейсингу запитів. Система зможе корелювати події між сервісами. Аналіз логів виявить приховані проблеми продуктивності. Механізми ротації оптимізують зберігання журналів. Пошук допоможе швидко знаходити інциденти. Розширення збереже конфіденційність даних.

Оптимізація розгортання через нові інструменти автоматизації. Впровадження канареєчних релізів для тестування. Система зможе автоматично відкочувати проблемні оновлення. Конфігурація сервісів керуватиметься через код. Розгортання відбуватиметься без простоїв сервісу. Автоматизація зменшить ризики людських помилок. Вдосконалення збереже надійність системи.

Розширення тестового покриття через нові види тестів. Впровадження фаззінг-тестування компонентів безпеки. Система отримає тести продуктивності мікросервісів. Сценарії перевірять відмовостійкість архітектури. Автоматизація прискорить виконання тестів. Звіти допоможуть знаходити регресії. Розширення збереже якість коду.

Вдосконалення документації через інтерактивні приклади. Реалізація пісочниці для тестування API. Система згенерує документацію з коментарів коду. Приклади продемонструють типові сценарії використання. Документація оновлюватиметься автоматично при змінах. Розробники отримають повний опис системи. Розширення збереже простоту інтеграції.

4.4. Впровадження та практичне застосування

Корпоративний сектор демонструє зацікавленість у впровадженні системи. Компанії шукають захищені рішення для внутрішніх комунікацій. Анонімність дозволяє співробітникам вільно обговорювати проблеми. Система легко інтегрується з існуючою інфраструктурою. Масштабованість задовольняє

потреби великих організацій. Технічна підтримка забезпечує стабільну роботу сервісу. Впровадження підвищує ефективність командної роботи.

Освітні установи можуть використовувати систему для організації дистанційного навчання. Платформа забезпечує захищений обмін навчальними матеріалами. Студенти отримують безпечне середовище для обговорень. Викладачі можуть створювати приватні групи для занять. Система підтримує передачу файлів та медіа-контенту. Анонімність заохочує активну участь у дискусіях. Платформа адаптується під різні формати навчання.

Медичні заклади розглядають систему для конфіденційних консультацій. Пацієнти можуть анонімно обговорювати проблеми зі здоров'я. Лікарі надають рекомендації через захищений канал зв'язку. Система відповідає вимогам захисту медичних даних. Платформа підтримує групові консультації та конференції. Анонімність знижує психологічний бар'єр при зверненні. Впровадження покращує доступність медичної допомоги.

Громадські організації застосовують систему для координації діяльності. Активісти отримують безпечний канал комунікації. Платформа захищає від стеження та прослуховування. Система підтримує створення тематичних груп. Учасники можуть ділитися документами та матеріалами. Анонімність забезпечує захист персональних даних. Впровадження посилює ефективність громадських ініціатив.

Журналістська спільнота використовує систему для захищеного спілкування з джерелами. Інформатори можуть безпечно передавати дані через платформу. Система захищає конфіденційність учасників комунікації. Журналісти створюють приватні канали для розслідувань. Платформа підтримує передачу документів та файлів. Анонімність гарантує безпеку джерел інформації. Впровадження сприяє розвитку журналістських розслідувань.

Правозахисні організації впроваджують систему для роботи з постраждалими. Платформа забезпечує анонімне звернення за допомогою. Консультанти надають підтримку через захищений канал. Система зберігає конфіденційність особистих даних. Користувачі отримують доступ до

тематичних ресурсів. Анонімність створює довірче середовище спілкування. Впровадження розширює можливості правового захисту.

Наукові установи застосовують систему для захищеної співпраці. Дослідники обмінюються даними через шифровані канали. Платформа підтримує створення закритих груп проектів. Система забезпечує захист інтелектуальної власності. Учасники можуть ділитися результатами досліджень. Анонімність захищає пріоритет наукових відкриттів. Впровадження прискорює наукову комунікацію.

Фінансові установи розглядають систему для конфіденційних консультацій. Клієнти отримують анонімний доступ до експертів. Платформа захищає від витоку фінансової інформації. Система підтримує індивідуальні та групові консультації. Користувачі можуть безпечно обговорювати інвестиції. Анонімність знижує ризики шахрайства. Впровадження підвищує довіру клієнтів.

Психологічні служби використовують систему для онлайн-консультування. Пацієнти можуть анонімно звертатися за допомогою. Платформа створює безпечний простір для терапії. Система підтримує індивідуальні та групові сесії. Психологи ведуть захищені записи консультацій. Анонімність допомагає подолати страх звернення. Впровадження розширює доступність психологічної допомоги.

Комерційні компанії адаптують систему під корпоративні потреби. Платформа інтегрується з існуючими бізнес-процесами. Система забезпечує захист комерційної таємниці. Співробітники отримують захищений канал комунікації. Керівництво може створювати закриті групи проектів. Анонімність сприяє відвертому обговоренню проблем. Впровадження оптимізує внутрішні комунікації.

ВИСНОВКИ

У роботі було реалізовано систему анонімного обміну повідомленнями, що ґрунтується на мікросервісній архітектурі. Для розробки обрано стек технологій Go (серверна частина) та Svelte (клієнтська частина), що забезпечило високу продуктивність, надійність та зручність у використанні.

Під час тестування було підтверджено стабільну роботу системи при навантаженні — реалізація на базі горутин дозволила обробляти одночасні з’єднання без втрати продуктивності.

Завдяки використанню протоколу WebSocket забезпечено передачу повідомлень у режимі реального часу. Реалізована система автентифікації не вимагає персональних даних, а підтримка одноразових токенів і хешування паролів гарантує безпеку облікових записів. Всі комунікації зашифровані наскрізним шифруванням, а історія повідомлень не зберігається на сервері, що підтверджує відповідність принципам анонімності.

Клієнтська частина системи продемонструвала високу швидкодію та адаптивність. Інтерфейс залишався стабільним і зручним навіть на мобільних пристроях з обмеженими ресурсами.

Модульні та інтеграційні тести охопили всі критичні компоненти системи, дозволивши виявити та усунути помилки на ранніх етапах розробки. Засоби моніторингу, інтегровані в систему, успішно фіксували робочі події та відхилення.

Таким чином, поставлені цілі були досягнуті, а отримані результати свідчать про ефективність архітектурних та технічних рішень, застосованих у розробці захищеної системи анонімної комунікації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пугач А. В. Метод віртуалізації великих даних при аналізі анонімних мереж : монографія. 2020.

2. Карабан Д. Р. Методи та програмно-апаратні засоби забезпечення протидії відстеженню та ідентифікації користувачів комп'ютерних мереж при роботі з Інтернет-ресурсами : магістерська дис. Тернопільський національний технічний університет імені Івана Пулюя, 2023.

3. Чура Т., Чура Н. Огляд сучасних методів автентифікації для мікроконтролерів // Кібербезпека: освіта, наука, техніка. 2024. №1(25). С. 200-214.

4. Бут О. В. 3D редактор для оптимізації спеціалізованих технічних рішень при розробці комп'ютерних ігор : монографія. 2021.

5. Корман Н. А. Аналіз технологій побудови віртуальних захищених мереж VPN // Перспективи телекомунікацій : матеріали Міжнародної науково-технічної конференції. 2020.

6. Бостанжи Ю. І. Аналіз технологій побудови захищеного доступу до мережі за допомогою віртуального локального зв'язку : монографія. 2022.

7. Григоров Г. В. Розроблення вебзастосунку для комунікації в режимі реального часу з використанням протоколу WebSocket : монографія. 2022.

8. Д'яконов Д. К. Дослідження відображення веб-контенту в режимі реального часу за допомогою протоколу WebSocket та безсерверної платформи : монографія. 2021.

9. Суслова Є. А., Войцеховська О. В. Інтеграція чату в інтернет-магазин з використанням протоколу WebSocket : дис. ВНТУ, 2023.

10. Єременко Н. М. Клієнт-серверна система обробки та аналізу вхідних даних користувача для оптимізації роботи веб-порталу : монографія. 2022.

11. Кивацький І. М. Дослідження ризиків та вразливостей в системі керування розумним будинком : магістерська дис. Тернопільський національний технічний університет, 2023.

12. Орловський Д. І. Проектування та розробка програмного застосунку деанонімізіції користувачів у одноранговому протоколі BitTorrent в рамках OSINT-розвідки : магістерська дис. Національний університет «Запорізька політехніка», 2023.

13. Хуторянський Д. О. Розробка мікросервісів з використанням мови програмування Go : монографія. 2023.

14. Оглобліна В. С. Особливості використання мови програмування Golang для побудови вебсайту на прикладі тематичного блогу : монографія. 2023.

15. Тихоновський В. І. Використання технологій Go, Stateless і React.js для розробки ігрового додатку : монографія. 2020.

16. Брежнєв К. М. Розробка клієнтської частини соціального месенджера на основі Інтернет магазину з продажу одягу JavaScript/React : монографія. 2022.

17. Качанов О. В. Система автоматизованого обліку навчального процесу : монографія. 2023.

18. Недошивко В. С. Розробка вебзастосунку для пошуку автомобілів у розшуку : магістерська дис. Національний університет «Запорізька політехніка», 2022.

19. Фісун М. Т. та ін. Використання методу аналізу ієрархій для вибору засобів розробки синтаксичних аналізаторів при створенні DSL // Наукові праці ВНТУ. 2021. №1.

20. Матвєєв М. І., Кучук Г. А. Аналіз фреймворків для створення клієнтських застосунків : монографія. 2023.

21. Булах І. В. Нормативні особливості проектування архітектурно-містобудівної системи закладів охорони здоров'я // Colloquium-journal. 2019. №3-1(27). Голопристанський міськрайонний центр зайнятості.

22. Цибульник С., Бідник Д. Проектування архітектури автоматизованої бібліографічної системи // Вісник Національного технічного університету «ХПІ». Серія: Нові рішення у сучасних технологіях. 2021. №2(8). С. 83-89.

23. Комлева Г. О., Попова М. О. Проектування архітектури системи для перевірки якості джерел даних : монографія. 2022.

ДОДАТКИ

```
package main

import (
	"context"
	"log"
	"net/http"
)

func main() {

	// Створюємо кореневий ctx і CancelFunc, які можна використовувати для скасування goroutine retentionMap
	rootCtx := context.Background()
	ctx, cancel := context.WithCancel(rootCtx)

	defer cancel()

	setupAPI(ctx)

	err := http.ListenAndServe(":8080", nil)
	if err != nil {
		log.Println(err)
	}
}

// setupAPI запустить всі шляхи та отримувачі
func setupAPI(ctx context.Context) {
```

```
        // Створюємо екземпляр диспетчера, який використовується для обробки підключень WebSocket
        manager := NewManager(ctx)
        // Обслуговуємо каталог з фронтенд частиною на шляху /
        http.Handle("/", http.FileServer(http.Dir("../frontEnd/18518/dist")))
        http.HandleFunc("/login", manager.loginHandler)
        http.HandleFunc("/register", manager.registerHandler)
        http.HandleFunc("/ws", manager.serveWS)
    }
```