КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА СИСТЕМИ ОБМІНУ МИТТЄВИМИ ПОВІДОМЛЕННЯМИ З НАСКРІЗНИМ ШИФРУВАННЯМ

Виконав студент 4-го курсу
Михайло СОКОЛОВ _____
(підпис)

Науковий керівник:
доцент, кандидат технічних наук
Євген ДЕМКІВСЬКИЙ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 10 ілюстрацій, 3 таблиці, 10 джерел посилань.

БАЗА ДАНИХ, ВЕБ-ДОДАТОК, КЛІЄНТСЬКА ЧАСТИНА, МЕСЕНДЖЕР, ОБМІН ПОВІДОМЛЕННЯМИ, СЕРВЕРНА ЧАСТИНА, ШИФРУВАННЯ.

Об’єктом роботи є процес обміну зашифрованими повідомленнями між користувачами, а також збереження цих даних на хмарному сховищі. Предметом роботи є веб-клієнт та веб-сервер для зручного та простого виконання вищеописаного процесу.

Метою роботи є розробка web-системи обміну миттєвими повідомленнями, яка буде складатись із клієнтської та серверної частин.

Інструменти розробки: мова програмування JavaScript, мова програмування Golang, фреймворк Vue.js, система керування базами даних PostgreSQL, інтегроване середовище розробки GoLand, бібліотеки криптографічних примітивів.

Результат роботи: розроблено web-проєкт із клієнтською та серверною частиною для обміну миттєвими повідомленнями з наскрізним шифруванням.

Результат роботи може використовуватись як додаток для обміну миттєвими повідомленнями або як будівельний блок для більш великих месенджерів із готовими вбудованими інструментами шифрування.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

БД – База даних;

ОС – Операційна система;

СКБД – Система керування базами даних;

API – Application Programming Interface (програмний інтерфейс додатку);

CSS – Cascading Style Sheets (мова стилю сторінок);

E2E – End-to-end (наскрізне);

HTML – HyperText Markup Language (мова розмітки web-сторінок);

HTTP – HyperText Transfer Protocol (протокол передачі даних);

JS – JavaScript (мова програмування);

JSON – JavaScript Object Notation (формат обміну даними);

Nonce – Number that can only be used once (число, яке повинно використовуватись лише один раз. Використовується в криптографічних протоколах);

SQL – Structured Query Language (мова для написання запитів);

UI – User Interface (інтерфейс користувача).

ВСТУП

Оцінка сучасного стану об'єкта розробки. У сучасному світі люди постійно обмінюються між собою інформацією в мережі. Вона може бути робочого плану, така як дати певних конференцій, або ж звичайним листуванням з колегою або другом. Для одних цілей використовують обмін листами за допомогою електронної пошти, для інших – обмін миттєвими повідомленнями за допомогою месенджерів, і друге нині стає все більш популярним для будь-яких сфер. Але, незалежно від вибору користувача, сервіс, яким він користується, повинен захищати інформацію, яка передається. В такому формулюванні найважливішим аспектом є збереження секретності інформації, або ж – її конфіденційність. Тобто, лише користувач, якому призначалась певна інформація, повинен мати можливість її прочитати і ніхто інший. Це може бути забезпечено за допомогою наскрізного шифрування. Під час такого шифрування два користувачі мають однаковий спільний ключ шифрування. Також, бажано було б, щоб така система зберігала зашифровані повідомлення у базі даних, щоб користувач не зберігав їх у себе на пристрої постійно, а завантажував коли це потрібно. Такі системи називаються «хмарними».

Актуальність роботи та підстави для її виконання. Проаналізувавши вище сказане, можна зробити висновок, що багато користувачів хотіли б мати можливість обмінюватись повідомленнями, зберігаючи при цьому їх конфіденційність. Тобто, існує потреба в створенні зручної системи обміну миттєвими повідомленнями, яка б забезпечувала хоча б конфіденційність цих повідомлень та зберігала повідомлення на стороні сервера. Такою системою може бути web-застосунок, або конкретніше – web-месенджер, який зможе забезпечити необхідну функціональність. Також, важливо відмітити, що додаток повинен ґрунтуватися на криптографічні методи, а не на обіцянки розробника, що дані надійно захищені та не будуть розповсюджуватись.

Мета й завдання роботи. Метою роботи є розробка web-системи обміну миттєвими повідомленнями, яка буде складатись із клієнтської та серверної частин. Для досягнення цієї мети було поставлено наступні завдання:

- обрати технології, які будуть використовуватись для розробки системи;
- обрати криптографічні алгоритми та протоколи, завдяки яким буде досягнуто конфіденційність та автентичність;
- спроектувати систему та структури даних, які будуть зберігатись в БД;
- розробити клієнтську та серверну частину, використовуючи обрані технології. Важливо, щоб користувач пам'ятав зручні для нього дані, такі як логін та пароль, а не довгі сід-фрази;
- перевірити, що користувачі системи дійсно можуть обмінюватись повідомленнями та вони зберігаються на боці сервера у зашифрованому вигляді.

Об'єкт, методи й засоби розробки. Об'єктом роботи є процес обміну зашифрованими повідомленнями між користувачами, а також збереження цих даних на хмарному сховищі. Предметом роботи є веб-клієнт та веб-сервер для зручного та простого виконання вищеописаного процесу. Перед початком виконання роботи було обрано ряд інструментів, технологій та криптографічних примітивів, які будуть використані для розробки веб-проекту.

Для створення клієнтської частини було використано мову програмування JavaScript, фреймворк Vue.js для побудови UI та мови розмітки та стилів HTML і CSS. Також було застосовано ряд криптографічних бібліотек для генерації ключів, шифрування даних, цифрового підпису.

Для створення серверної частини було використано мову програмування Golang та СКБД PostgreSQL для збереження зашифрованих повідомлень та

деяких публічних даних користувачів. Як і в клієнтській частині, тут також було використано ряд криптографічних бібліотек.

Можливі сфери застосування. Розроблений застосунок може використовуватись усіма, хто бажає обмінюватись повідомленнями та зберігати їх секретність. При цьому, усі повідомлення в зашифрованому вигляді будуть знаходитись на сервері. Тобто користувачу не потрібно зберігати їх на своєму пристрої і він зможе прочитати їх в будь-який інший час. Також, розроблений застосунок може бути використаний як «будівельний блок» для розробки більш потужного додатку.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Системи обміну миттєвими повідомленнями, або простіше – месенджери, на сьогодні є досить поширеними та популярними. Вони розробляються багатьма компаніями в світі, але не всі з них надають бажану безпеку та конфіденційність. Також, більшість месенджерів не є відкритими, тобто, не мають публічно опублікованого коду. Було розглянуто декілька найпопулярніших варіантів месенджерів. Хоча велика кількість з них мають вбудоване наскрізне шифрування за замовчуванням, ці варіанти при цьому не мають хмарності додатку, тобто змушують користувача зберігати усю інформацію на своєму пристрої. У випадку втрати доступу до цього пристрою єдиний варіант відновити втрачені дані – синхронізуватись з іншим співбесідником, який має цю історію збереженою в себе. Натомість, розглянутий хмарний месенджер не надає наскрізного шифрування за замовчуванням, що не є бажаним.

1.1 Месенджер Telegram

Telegram – кросплатформений хмарний месенджер, створений Telegram Inc, який доступний на таких ОС як IOS, Android, Windows, Mac, Linux та у браузері [1]. Він підтримує наскрізне шифрування (E2E) в секретних чатах, але в звичайних використовує лише шифрування на транспортному рівні. Месенджер дозволяє обмінюватись повідомленнями, файлами, телефонувати співбесіднику по аудіо- та відео-зв’язку, тощо. Telegram є найбільш популярним месенджером в Україні та країнах СНД. Також, офіційні компоненти та протокол мають відкритий вихідний код, але серверна частина є власністю компанії та її код не розповсюджується. Для автентифікації сервера, ПЗ клієнта має вбудований публічний ключ сервера.

Переваги:

— простий та зручний інтерфейс;

— є хмарним месенджером;

— широкий функціонал;

— можливість створення чату з наскрізним шифруванням;

— відкритий вихідний код клієнту та протоколу.

Недоліки:

— закритий вихідний код сервера;

— відсутність наскрізного шифрування за замовчуванням.

1.2 Месенджер WhatsApp

WhatsApp – найпопулярніший месенджер у світі, спочатку розроблений WhatsApp Inc., а потім придбаний компанією Meta [2]. З 2016 року має E2E шифрування, яке базується на протоколі Signal. Застосунок не має відкритого вихідного коду.

Переваги:

— простий інтерфейс;

— широкий функціонал;

— має наскрізне шифрування за замовчуванням.

Недоліки:

— закритий вихідний код;

— не є хмарним.

1.3 Месенджер Viber

Viber – VoIP-застосунок для дзвінків та обміну повідомленнями. Розроблений компанією Viber Media Inc [3]. Має E2E шифрування, але не є хмарним месенджером. Відсутність «хмарності» змушує користувачів зберігати усі повідомлення та файли на своєму пристрої, що не є зручним.

Також, застосунок не має відкритого вихідного коду ні клієнтської, ні серверної частини.

Переваги:

— простий інтерфейс;

— достатній функціонал;

— має наскрізне шифрування за замовчуванням.

Недоліки:

— закритий вихідний код;

— не є хмарним;

— за відгуками – менш зручний аніж WhatsApp та Telegram.

1.4 Месенджер Signal

Signal – кросплатформений додаток зашифрованих миттєвих повідомлень, розроблений Signal Foundation та Signal Messenger LLC [4]. Застосунок дозволяє надсилати файли, голосові нотатки, зображення, відео та текстові повідомлення. До 2015 року мав назву TextSecure та дозволяв надсилати лише текстові повідомлення. Має E2E шифрування та відкритий вихідний код.

Переваги:

— простий інтерфейс;

— широкий функціонал;

— має наскрізне шифрування за замовчуванням;

— має відкритий вихідний код.

Недоліки:

— не є хмарним.

1.5 Порівняння наявних рішень

Більшість розглянутих рішень мають простий, зрозумілий та зручний інтерфейс, що є неймовірно важливим для кінцевого користувача. Також, більшість мають E2E шифрування, але Telegram, наприклад, не має його увімкненим за замовчуванням, що не є бажаним. Не менш важливим аспектом є «хмарність» месенджеру, тобто, можливість зберігати усі зашифровані повідомлення на сервері, а не на пристрої користувача. З розглянутих варіантів, на жаль, цим володіє лише Telegram, що є зрозумілим, адже зберігати велику кількість повідомлень та файлів може бути досить дорого. Також, не усі користувачі хочуть зберігати свої повідомлення, навіть у зашифрованому вигляді на сервері.

Отже, наш застосунок повинен, як мінімум, мати зручний та зрозумілий інтерфейс, E2E шифрування та, було б бажано, щоб він був хмарним. В майбутньому може бути додана можливість вимкнути збереження повідомлень в хмарі за бажанням користувача або інші функції.

РОЗДІЛ 2 ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Майже всі розглянуті вище рішення були розроблені за допомогою відомих мов, таких як Java, C, C++, Python, тощо. Так як наш додаток має бути з відкритим вихідним кодом, орієнтація при виборі робилась на прості і швидкі мови та фреймворки, які зможе легко прочитати та зрозуміти навіть користувач, або розробник, який не має великого досвіду з програмування. Саме тому, для серверної частини було обрано мову Golang від Google, яка базується на мові C, та СКБД PostgreSQL, а для клієнтської частини було обрано багатофункціональний фреймворк Vue.js та мову JavaScript.

2.1 Мова програмування Golang

Golang (або просто Go) – компільована мова програмування, розроблена Google [5]. Go має вбудовані та зручні засоби для паралельних обчислень, які називають горутинами, та потужну бібліотеку криптографічних засобів. Підтримка мови здійснюється для усіх популярних систем, таких як Linux, Android, Mac OS X та Windows.

Метою створення Go було бажання об’єднати легкість написання коду, високу продуктивність та захищеність від помилок. Синтаксис базується на стандартних елементах мови C та Python. Код досить легко читати та сприймати, а додатки, такі як веб-сервери, потребують набагато менше коду, аніж в інших мовах. Веб-сервер написаний на Go буде приблизно в півтора, або навіть в два рази менший за сервер з таким самим функціоналом, написаний на Java або C++.

Як було зазначено, Go має вбудовані засоби для паралельних обчислень – горутини. Мова створювалась з оглядкою саме на багатониткове програмування та ефективну роботу на багатоядерних системах. Горутини є набагато простішими за аналоги потоків в інших мовах, але, при цьому, не є повільнішими за них. Іноді горутини називають «легкими потоками», що

теоретично не є стовідсотково вірним, але, все ж таки таке словосполучення може мати місце.

Хоча синтаксис і схожий на C та Python, але авжеж мова також має відмінності. Go не потребує крапки з комою в кінці рядка, як в C та C++, та дужок після операторів if або for, як в Python. Також, мова не має оператору while, замість нього можна використовувати оператор for з трошки іншим синтаксисом. Go дозволяє досить легко працювати з динамічними масивами – слайсами, та має низку інших вбудованих типів, таких як хеш-таблиці та канали, які використовуються для спілкування горутин між собою. На відміну від C та C++, Go має вбудоване автоматичне прибирання сміття (Garbage Collector), який працює за алгоритмом Mark and sweep. Також, нещодавно було додано підтримку шаблонних функцій (templates).

Мову було обрано через зручність, стислість та орієнтованість на клієнт-серверну архітектуру і багатоникове програмування. Код є більш стислим та зрозумілим, що для нас є неабияким плюсом. Також, стандартна бібліотека Go має потужні криптографічні алгоритми. Ця мова є основною під час розробки серверної частини у роботі.

2.2 Веб-фреймворк Vue.js

Vue.js (або просто Vue) – веб-фреймворк написаний на JavaScript для створення користувацьких інтерфейсів з відкритим вихідним кодом [6]. Перша версія була опублікована 2013 року Еваном Ю (You), який тоді працював у Google. Причиною створення була відсутність готових рішень для швидкої побудови складних користувацьких інтерфейсів. React тоді був на початковій стадії розробки, а інші фреймворки, такі як Angular або Backbone.js були занадто складними. Як і Go, розробник Vue хотів створити просте, але при цьому потужне рішення для розробників.

Більшість вважають Vue простішим для вивчення за інші фреймворки, такі як React та Angular. Vue є реактивним, тобто представлення у вигляді Model-View-Controller змінюється, коли змінюється модель. Це дозволяє створювати веб-сторінки, які будуть динамічно мінятись, але при цьому не будуть вимагати оновлення сторінки. Фреймворк підтримує використання JavaScript, HTML, CSS та, за бажання, можливе використання TypeScript. Також, фреймворк підтримує шаблони на основі HTML, CSS переходи, анімації та роутинг.

Vue-файл розділений на три блоки: блок шаблонів – в ньому знаходиться HTML код компоненти , блок скрипту – в ньому зберігаються локальні дані та знаходяться функції, написані мовою JavaScript або TypeScript та стилів – для задання CSS стилів. Також, фреймворк має низку не обов'язкових бібліотек для роутингу (vue-router), сховища (vuex) та інші. На офіційному сайті є досить велика та добре оформлена документація фреймворку. Він надає потужний функціонал, який дозволяє швидко та якісно створювати різні веб-рішення.

2.3 СКБД PostgreSQL

PostgreSQL (іноді називають просто Postgre або Postgres) – об'єктно реляційна система керування базами даних (СКБД), розроблена Майклом Стоунбрейкером [7]. На відміну від інших СКБД, таких як Oracle, PostgreSQL не має єдиної компанії яка відповідає за її оновлення та має відкритий вихідний код. Базується на мові SQL і підтримує більшість можливостей стандарту SQL від 2016 року. Postgres підтримує управління базою даних через запити, інтерфейс командного рядка та спеціальне ПЗ – pqAdmin. Інтерфейс програми зображений нижче на рис. 1. Можемо побачити в лівій частині існуючі бази даних, в яких є каталоги, публікації, схеми, тощо.

Створивши БД можна відкрити інструмент запитів, який дозволить отримувати дані з таблиць.

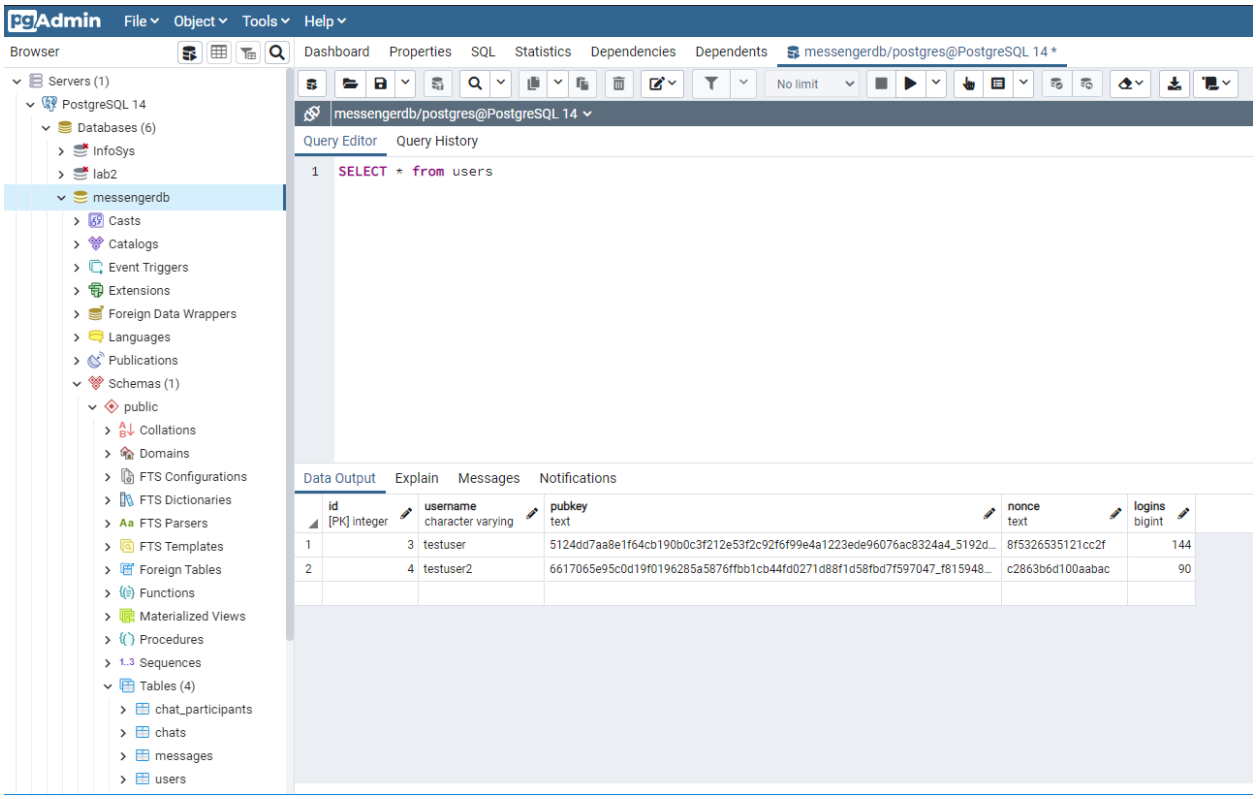

Рисунок 1 - Приклад інтерфейсу pqAdmin.

СКБД не має обмежень на розмір бази даних, але має обмеження на максимальний розмір таблиці – 32 Терабайти, а також, максимальний розмір поля – 1 Гігабайт. Обидва обмеження є несуттєвими для наших цілей. Перевагами PostgreSQL є:

— швидкі та надійні механізми транзакцій і репликацій;

— підтримка багатьох мов, таких як PL/pgSQL, PL/Perl, PL/Python та інших;

— наслідування;

— вбудована підтримка слабоструктурованих даних у форматі JSON;

— зручний інтерфейс взаємодії з базою даних pqAdmin.

Свої скорочені назви СКБД отримала, тому що базувалась на open-source проекті Postgres, який розроблявся у Каліфорнійському університеті в Берклі.

Функції (та запити) у PostgreSQL можуть бути написані за допомогою чистого SQL або ж вбудованої процедурної мови PL/pgSQL, який є аналогом мови PL/SQL з Oracle. Також, СКБД має тригери, механізм правил та підтримує низку типів даних, таких як: int, float, double, binary, bool, enum, тощо. PostgreSQL має підтримку індексів таких типів, як B-Tree, GiST, GIN, BRIN, Bloom, тощо. Також підтримується створення користувацьких типів, їх перетворень, доменів, функцій (включаючи агрегатні), операторів та мов. Таблиці PostgreSQL можуть наслідувати поля та характеристики батьківских таблиць. Усі таблиці в межах однієї БД зберігаються в спеціальному розділі, який називається Schemas (схеми).

РОЗДІЛ 3 ОГЛЯД ВИКОРИСТАНИХ КРИПТОГРАФІЧНИХ ЗАСОБІВ

Криптографія – наука про методи забезпечення конфіденційності, цілісності, автентифікації та шифрування даних. Саме завдяки криптографічним алгоритмам шифруються, підписуються та перевіряється цілісність даних та повідомлень. Додаток потребує всього вищезазначеного, тому потрібно розглянути використані технології більш детально. Для забезпечення автентифікації сервера було використано цифрові підписи та криптографію на еліптичних кривих, а для шифрування – блочний шифр з симетричним ключем.

3.1 Генерація ключів

Так як користувач повинен шифрувати свої повідомлення, що насправді буде виконуватись «невидимо» для нього, веб-клієнт на його комп'ютері повинен згенерувати ключі для підпису та шифрування даних. В нашому додатку ключі будуть генеруватись асиметричні, тобто публічний та приватний, де публічний – точка на еліптичній кривій, а приватний – велике число.

Щоб кінцевий користувач не запам'ятовував ключі або додаткові сід-фрази, ми будемо використовувати спеціальну функцію PBKDF2 (Password-Based Key Derivation Function), яка буде створювати ключі з нашого пароля та «криптографічної солі». Теоретично, можлива ситуація, коли два користувачі будуть мати однаковий пароль та однакову криптографічну сіль. Потрібно зазначити, що сіль видається сервером під час реєстрації користувача. Насправді ж, вірогідність того, що у двох користувачів будуть два однакові паролі вже досить мала, а вірогідність отримати однакову сіль, яка є 64 бітним числом робить її настільки малою, що нею можна знехтувати. В клієнтській

частині будуть невеликі обмеження на мінімальну та максимальну довжину пароля.

Таким чином, два користувачі, які в теорії можуть мати однаковий пароль, будуть мати різні ключі, адже вірогідність отримати однакове випадкове 64-бітне число – 2^{-64}. Щоб ще зменшити цю вірогідність – можна додати поштову адресу, вік користувача, ім'я та інше, але наша система не вимагає цих параметрів при реєстрації. Найпростіший спосіб зменшити цю вірогідність – збільшити розмір солі, наприклад, до 96 або 128 бітів. При потребі – це може бути зроблено в майбутньому зі збереженням зворотної сумісності. Таким чином, сервер буде зберігати публічні ключі користувачів, а вони, в будь-який момент зможуть відновити як публічний, так і приватний ключ використовуючи свій пароль та сіль.

3.2 Протокол Diffie-Hellman

Шифрування за допомогою асиметричної криптографії є зручним, але дорогим та обмеженим. Для того, щоб використати симетричне шифрування нам потрібен спільний приватний ключ між двома користувачами. Для цього буде використано протокол генерації симетричного ключа, створений Діффі та Геллманом [8].

Завдяки цьому криптографічному протоколу, дві сторони можуть згенерувати спільний секретний ключ, використовуючи незахищений канал, який прослуховується, але повідомлення при цьому не підмінюються. В нашій системі сервер можна вважати довіреною стороною, яка буде надавати публічні ключі одних користувачів іншим.

Протокол працює так:

1. Сервер повинен надати (насправді можуть бути «зашиті» в код клієнта) параметри системи – велике просте число p та генератор скінченної циклічної групи g.

2. Під час реєстрації, користувач Аліса генерує свій закритий ключ “a”, та публічний ключ A = g^a mod p, де mod – операція взяття остачі від ділення, або простіше – модуль. Окремо від неї це також зробив користувач Б. В нашому випадку користувачі також відправляють публічні ключі на сервер, який потім надсилає їх у відповідь на відповідні запити.

3. Коли Аліса хоче зв’язатись з Бобом, вона запитує у сервера публічний ключ Боба і отримує B, тобто g^b mod p. Після чого підносить його до степені свого приватного ключа, тобто $(g^b)^a$ mod p => g^{ba} mod p. Це і є спільний ключ Аліси з Бобом. Так як в нашій системі ключі – це точки на еліптичній кривій, ми будемо подавати спільну точку на вхід іншого алгоритму.

Коли Боб отримує повідомлення від Аліси, він запитує у сервера її публічний ключ та робить ті ж самі операції, тобто $(g^a)^b$ mod p => g^{ab} mod p = g^{ba} mod p і розшифровує ним повідомлення від A. Візуалізація протоколу зображена на рис. 2.

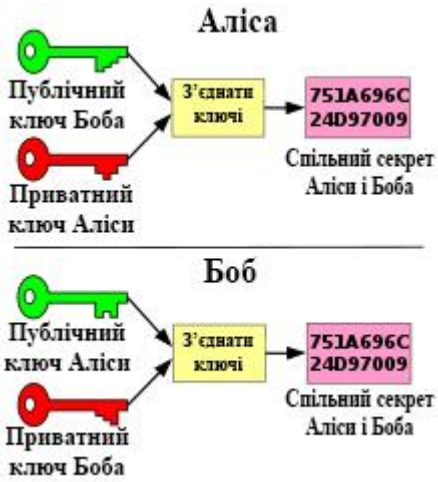

Рисунок 2 – Протокол Diffie-Hellman

Протокол Діффі-Геллмана опирається на складність обчислення дискретного логарифму (в групі точок еліптичної кривої). Через те, що публічні ключі в нашому додатку – еліптичні точки, спільний ключ також буде точкою. Тому, ми будемо використовувати її, а точніше лише координату

ікс, як вхідне значення до алгоритму хешування, а так як у обох користувачів вийде однакова точка – обидва згенерують однаковий симетричний ключ для шифрування.

3.3 Алгоритм симетричного шифрування AES-256

Advanced Encryption Standart (також відомий як Rijndael) – симетрий алгоритм блочного шифрування з розміром блоку 128 біт та ключами з вибірковою довжиною 128, 192 або ж 256 біт, створений Вінсентом Рейменом та Йоаном Дайменом [9]. Був прийнятий як стандарт шифрування урядом Сполучених Штатів Америки на заміну минулого стандарту – DES у 2002 році. Алгоритм використовує блоки підстановки (S-блоки) та блоки перестановки (P-блоки). Блоки підстановки обирались групою експертів, та вони є закодованими в алгоритм.

Алгоритм має такі функції як SubBytes, ShiftRows, MixColumns та інші. Початковий головний ключ використовується для генерації раундових ключів. Кількість раундів в алгоритмі відрізняється, в залежності від розміру основного ключа. Для 256-бітного ключа кількість раундів в алгоритмі становить 14. Раунд – це комбінація з однієї підстановки та двох перестановок, які в свою чергу використовуються вищезазначені функції.

Основна ідея полягає в тому, щоб за допомогою перестановок та підстановок «заплутати» текст, а точніше біти тексту, так, що розшифрувати їх зможе лише той, хто повинен отримати це повідомлення. Тобто, не маючи ключа, зловмисник не зможе росшифрувати повідомлення за адекватний час. Під адекватним часом мається на увазі десятки, якщо не сотні років, чого цілком достатньо для більшості повідомлень.

3.4 Алгоритм цифрового підпису ECDSA

Алгоритм цифрового підпису дозволяє автентифікувати користувача, а точніше – власника приватного ключа, який відповідає певному публічному ключу. Один з підходів для створення цифрового підпису – асиметричне шифрування, використовуючи приватний ключ (зазвичай використовують для шифрування публічний ключ і лише власник відповідного приватного може розшифрувати повідомлення). Аліса шифрує своїм приватним ключем повідомлення, і будь-який інший користувач може перевірити підпис за допомогою публічного ключа Аліси, та впевнитись, що цей підпис дійсно створила вона. Проблема може виникнути, якщо ми не знаємо чий це публічний ключ, або ж, ми отримаємо підпис та публічний ключ іншої людини, але не будемо цього знати.

Частіше всього підпис прикріпляється до певного повідомлення, але підписується не саме повідомлення, тому що воно може бути досить великим, а хеш-значення цього повідомлення. Тоді отримувач хешує отримане повідомлення та порівнює результат з розшифрованим підписом. Якщо вони співпадають – користувач впевнений що підпис створив саме власник цього публічного ключа. Сервер також може мати пару ключів та створювати підпис, що ми і будемо використовувати.

Є також інший підхід до створення цифрового підпису, який не шифрує повідомлення. Прикладом може слугувати використаний в роботі алгоритм цифрового підпису з використанням еліптичних кривих ECDSA (Elliptic Curve Digital Signature Algorithm) [10].

Алгоритми підпису та перевірки виглядають так:

1. Користувач Аліса обирає випадкове, або псевдовипадкове число k, в діапазоні [1, n-1], де n – публічний параметр (порядок групи), який відомий всім.

2. Вона обчислює точку $k*G = (x, y)$, де G – це також публічний параметр, що означає генератор циклічної групи порядку n.

3. Аліса обчислює r = x mod n і якщо r = 0, обирає інше k.

4. Аліса обчислює $s = k^{-1} * (e + dr) \mod n$, де e – хеш нашого повідомлення (e = hash(message)). Якщо s = 0, також обрати інше k і почати спочатку.

5. Пара (r,s) є підписом повідомлення m.

Щоб перевірити цей підпис, Боб виконує наступний алгоритм:

6. Обчислює хеш повідомлення та число $w = s^{-1} \mod n$.

7. Обчислює $u_1 = e*w \mod n$ та $u_2 = r*w \mod n$.

8. Обчислює координати точки $X = (x, y) = u_1 * G + u_2 * Q$ де Q – публічний ключ Аліси. Варто зазначити, що (x, y) це не та сама точка, яку обчислювала Аліса.

9. Якщо X – «нульова» точка, або, іноді її називають точкою на бескінечності, то підпис не приймається. Інакше перевіряється що x mod n дорівнює r, якщо це так – підпис приймається.

Підпис не використовується для кожного повідомлення, натомість, він використовується для автентифікації сервера та авторизації користувача в системі. Підписуючи певний nonce, користувач доводить, що знає свій приватний ключ, що в нашому випадку прирівнюється до знання паролю від цього акаунту. Публічний ключ сервера «зашитий» в код веб-сторінки і завдяки цьому буде перевірятись, що нам відповідає саме сервер.

РОЗДІЛ 4 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

Наступним етапом після вибору усіх технологій, протоколів та алгоритмів, які потрібні нам для розробки, було розроблено серверу та клієнтську частини, веб-сервер та веб-клієнт відповідно. Сервер надає декілька кінцевих точок (ендпоінтів), які дозволяють обмінятись початковою інформацією між клієнтом та сервером. Вони слугують для реєстрації та авторизації користувача з автентифікацією сервера, відкриття веб-сокет з'єднання та пошуку користувача. Обмін повідомленнями між двома користувачами здійснюється через WebSocket-з'єднання, що надає можливість отримувати їх швидко та зручно. Важливо, щоб лише користувач, який належить до відповідного чату міг отримати його історію. Саме тому подібні дії будуть виконуватись також через WebSocket-з'єднання. Розробка серверної частини виконувалась в IDE GoLand від JetBrains з використанням фреймворку Echo.

4.1 Огляд схеми бази даних

Перед створенням веб-серверу було спроектовано схему бази даних, в якій будуть зберігатись усі потрібні дані про користувача та його повідомлення у зашифрованому вигляді. Було також розглянуто інші схеми для подібних застосунків. Деякі таблиці та поля можуть бути додані за потреби розширення функціоналу, наприклад видалення повідомлень.

На рис. 3 можна побачити наступні сутності:

— users – кінцеві користувачі;

— chats – чати між користувачами (слугує «мостом» між іншими таблицями);

— chat_participants – учасники окремого чату;

— messages – повідомлення користувачів.

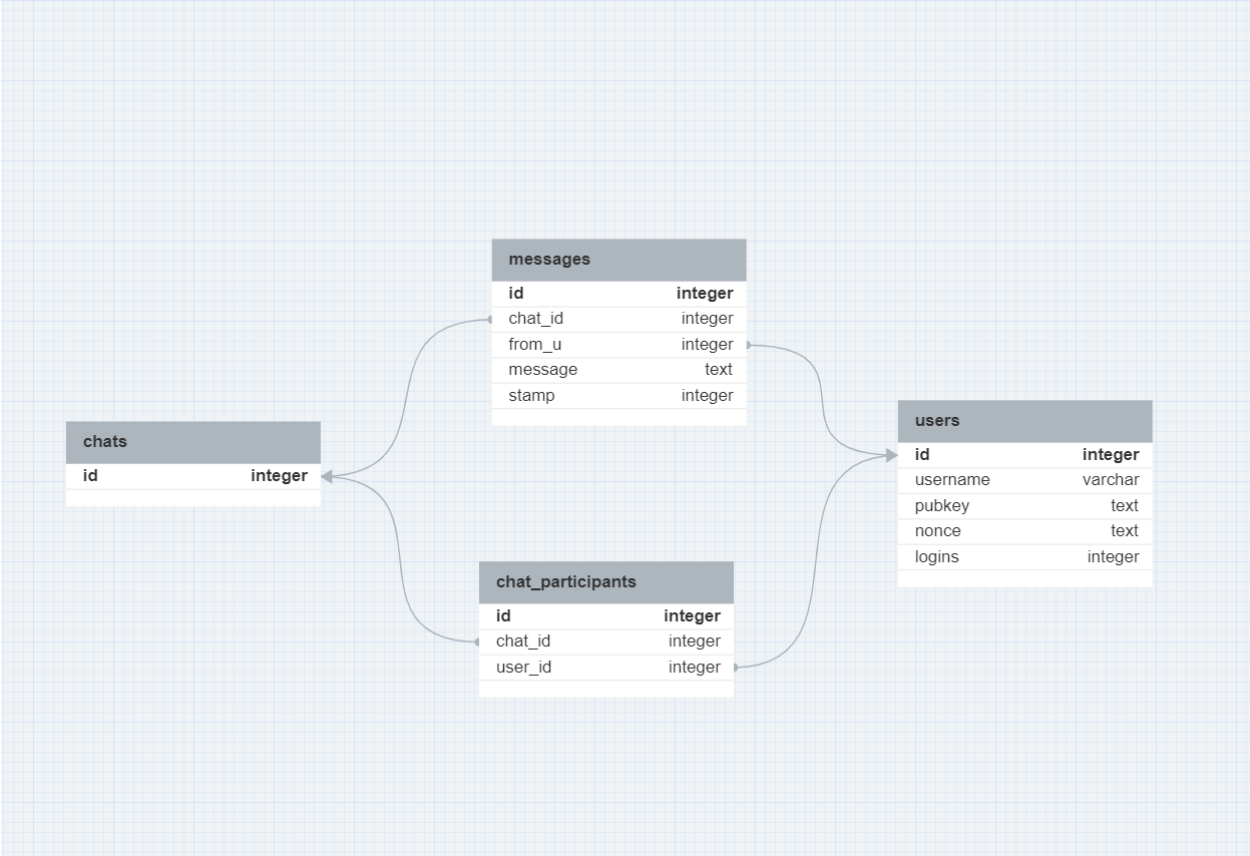

Рисунок 3 – Схема бази даних

Для більш глибоко розуміння нижче приведений опис даних у кожній з таблиць. Кожна таблиця містить ідентифікатор запису – id. Він не вказується в таблицях нижче, але потрібно пам’ятати що він є.

У таблиці 1 містяться поля для збереження даних про повідомлення.

Таблиця 1 – сутність повідомлень у БД

Ім’я атрибуту	**Тип атрибуту**	**Опис**
chat_id	integer	Ідентифікатор чату
from_u	integer	Ідентифікатор відправника
message	text	Зашифроване повідомлення
stamp	integer	Час UNIX

Таблиця сутності чатів не розглядається, адже єдине поле там – порядковий номер чату. Ця сутність використовується як «міст» між повідомленнями та користувачами, та для зручного сортування повідомлень. Вона може бути покращена додаванням певних полів, наприклад, останнього повідомлення в чаті.

У таблиці 2 наведено поля для збереження інформації про користувача.

Таблиця 2 – сутність користувачів у БД

Ім'я атрибуту	**Тип атрибуту**	**Опис**
username	varchar	Логін користувача
pubkey	text	Публічний ключ користувача
nonce	text	Криптографічна сіль
logins	integer	Nonce для авторизації

У таблиці 3 наведено поля для збереження інформації про учасників кожного чату. В майбутньому може бути використано для створення групових чатів.

Таблиця 3 – сутність учасників окремого чату у БД

Ім'я атрибуту	**Тип атрибуту**	**Опис**
chat_id	integer	Ідентифікатор чату
user_id	integer	Ідентифікатор користувача-учасника

4.2 Огляд доступних запитів

Для обміну даними під час деяких операцій, наприклад – реєстрації, було створено API для HTTP запитів. Користувач відправляє запит з даними, які

знаходяться в URL або у тілі запиту у форматі JSON на сервер, там він обробляється і, якщо немає ніяких помилок, повертає користувачу потрібні дані. Для отримання даних використовується метод GET, який не може мати тіла запиту (тобто не підтримує передачу даних у форматі JSON від користувача до сервера), POST – для додавання певних даних. Нижче вказується лише суфікс запиту, тобто не вказується адреса, порт (або ж доменне ім'я) сервера та протокол HTTP/HTTPS. Усі запити починаються з "/api/", далі йде дія та сутність до якої вона застосовується.

— "/preregister/user" – POST запит для першого етапу реєстрації користувача. Приймає два поля у форматі JSON – логін та випадкове велике число, яке використовується для автентифікації сервера. Перевіряє чи не є зайнятим такий логін користувача. У позитивному випадку повертає логін, ECDSA підпис великого числа, яке надав користувач, за допомогою ключів сервера, велике 64-бітне число - криптографічну сіль та початкове значення поля logins – 0. В БД додає до сутності users запис, який містить логін, сіль та наступне значення logins – 1;

— "/register/user" – POST запит для другого, фінального, етапу реєстрації користувача. Приймає дані у форматі JSON – логін, ECDSA підпис початкового значення logins (0) та публічний ключ для перевірки цього підпису. Перевіряє підпис і у позитивному випадку додає до бази даних публічний ключ користувача з таким логіном. Також перевіряє, що у користувача з таким логіном ще не має публічного ключа (захист від потенційної атаки підміни ключа);

— "/login/user" – POST запит для першого етапу авторизації користувача. Так само як і у випадку першого етапу реєстрації – приймає логін та випадкове число. Повертає підпис цього числа, сіль користувача та поточне значення поля logins;

— "/ws/:name/:sig" – GET запит для налаштування WebSocket-з'єднання між клієнтом та сервером. Є другим етапом авторизації користувача. У стрічці

URL, замість «:name» потрібно вказати логін користувача, у акаунт якого виконується авторизація, а замість «:sig» – ECDSA підпис поточного значення logins цього користувача у форматі “r||_||s”, де || – операції конкатенації. Сервер перевіряє чи існує такий користувач, та за допомогою його публічного ключа перевіряє наданий підпис, якщо все вірно – приймає WebSocket-з’єднання від клієнта;

— “/get/user/:name” – GET запит для отримання інформації про користувача. У стрічці URL заміть «:name» потрібно вказати логін користувача, інформацію про якого потрібно знайти. Повертає дані у форматі JSON – ім’я користувача та його публічний ключ.

4.3 Обмін даними з клієнтом через WebSocket-з’єднання

Після авторизації користувача, весь подальший обмін інформацією, окрім пошуку користувачів та їх публічних ключів, виконується через WebSocket-з’єднання. Їх реалізацію надає бібліотека “gorilla/mux”. Для кожного сокету було реалізовано 2 функції – окрема для зчитування даних з сокету та ще одна для всього іншого функціоналу. Кожна з них запускається в окремій горутині. Це зумовлено тим, що операція зчитування повідомлення з сокету блокує поточний контекст програми, а всю іншу логіку можна імплементувати використовуючи оператор select, який прослуховує та очікує події з каналів. Коли сокет отримує повідомлення, він відправляє його в спеціальний канал – структуру, за допомогою якої в мові Go горутини спілкуються між собою. В іншій горутині цього сокета прослуховуються три канали – один для закриття з’єднання з клієнтом, один для отримання повідомлення від клієнта та останній для отримання повідомлень від сервера, тобто, він іншого клієнта для поточного. Кожне повідомлення обробляється та виконується певна дія – збереження даних до бази, або ж надсилання відповіді. Логіка отримання повідомлень зображена на рис. 4.

```
func SocketReadLogic(username string, soc *models.Socket) {
    for {
        log.Println("Reading in ReadLogic     ", username)
        var packet models.Packet
        if err := soc.Conn.ReadJSON(&packet); err != nil {
            log.Println("error while reading message from the user ", err)
            return
        }
        soc.Out <- packet
    }
}
```

Рисунок 4 - Логіка зчитування повідомлення від клієнта

Для обміну між клієнтом та сервером було створено структуру даних Packet, яка містить такі поля:

— Type (integer) – тип повідомлення;

— From (string) – логін користувача від якого надійшло повідомлення. Якщо вказане таке ім’я, яке відрізняється від імені користувача, з яким встановлено з’єднання – пакет відкидається (ігнорується);

— To (string) – логін користувача, якому потрібно надіслати повідомлення. Може бути пустим полем для деяких запитів;

— Message (string) – корисне навантаження пакету. Вміст відрізняється, залежно від типу повідомлення. Може бути пустим для деяких типів, але не може бути пустим для типу 0.

Система має 7 типів повідомлень, які можна доповнити новими за потреби:

а) Тип 0 – зашифроване повідомлення від одного користувача іншому. Якщо користувач з логіном, яке зазначено в полі «To» знаходиться в додатку – йому одразу надсилається це повідомлення по відкритому WebSocket-з’єднанню.

б) Тип 100 – повідомлення-запит до сервера на отримання п’яти останніх чатів, в яких присутній клієнт, окрім тих, які вже отримані. Може мати порожню відповідь.

в) Тип 101 – відповідь на запит клієнта з типом 100, передає у відповідь користувачу повідомлення , в якому знаходиться п’ять останніх чатів.

г) Тип 102 – фактично те ж саме що і тип 101, але зазначає, що це був останній чат, який користувач міг отримати.

д) Тип 110 – повідомлення-запит до сервера на отримання 10 останніх неотриманих повідомлень з певного чату. Може мати порожню відповідь.

е) Тип 111 – відповідь на запит 110, містить 10 останніх зашифрованих повідомлень.

ж) Тип 112 – фактично те ж саме, що і попередній тип, але зазначає, що більше неотриманих повідомлень для цього чату немає.

Більшість повідомлень будуть мати тип 0, адже це обмін інформацією між користувачами, що наша система і повинна надавати. Повідомлення з типом 100 надсилається автоматично, один раз, при авторизації користувача. Подальші запити можуть бути ініційовані користувачем за його бажанням. Типи 102 та 112 потрібні, для того щоб веб-клієнт розумів, в який момент кнопки на веб-сторінці потрібно прибрати. Саме тому відповіді на ці запити можуть бути «порожніми».

РОЗДІЛ 5 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

Під час розробки серверу водночас розроблялась клієнтська частина системи, так як використання веб-сокетів вимагало наявність веб-клієнту. Клієнтська частина будувалась з використанням фреймворку Vue, бібліотеками vue-router, Bootstrap та вбудованим сховищем Vuex. Для генерації ключів та підписів на еліптичних кривих використовувалась бібліотека elliptic-js, яка підтримує велику кількість різних еліптичних кривих. Для наших цілей було обрано криву “p256”, яка є одним з стандартів NIST. Для шифрування використовувались потужні бібліотеки crypto-js та jscrypto. Обидві надають велику кількість алгоритмів хешування та шифрування зі стандартними існуючими режимами. Для шифрування повідомлень користувачів використовувався AES з розміром ключа 256 бітів та режимом CBC.

5.1 Особливості розробки веб-сторінки з Vue

Основна розмітка елементу, що розроблений з використанням Vue знаходиться в блоці шаблону. Там міститься весь HTML код поточної компоненти, а також можуть міститись спеціальні теги, які характерні лише для фреймворку. Як приклад, тег v-for (див. рис 5) допомагає зображати елементи, в залежності від розміру масиву, який зберігає дані про ці елементи. Тобто, якщо ми маємо список останніх чатів або повідомлень, за допомогою цього оператору ми зможемо зобразити їх усі, використовуючи мінімальну кількість коду. Це дуже зручно як для розробника, так і для читабельності коду іншими людьми.

```
<chat-user
  v-for="chatter in chatters"
  :key="chatter.username"
  :username="chatter.username"
  :pubKey="chatter.pubKey"
  @startChat="startChat"
>
```

Рисунок 5 - Приклад оператору v-for

Також, кожна компонента має блок стилів, в якому знаходиться перелік CSS стилів та їх специфікація для поточної компоненти. Також стилі можна задавати під час розмітки сторінки в першому блоці, але це скоріше виключення, аніж гарна ідея.

Найважливішим для нас є блок скриптів, адже саме там будуть відбуватись запити до сервера, створення ключової пари, шифрування та розшифрування повідомлень. Vue має спеціальні функції, які відповідають різним етапам життєвого циклу компоненти (див. рис 6). Найкориснішою для нас буде функція created(), яка відбувається майже на самому початку життєвого циклу. Вона використовується не так часто, а саме – для ініціалізації початкових даних головної сторінки користувача. Це потрібно, тому що WebSocket-з'єднання створюється на сторінці авторизації, а використовується на зовсім іншій. Щоб вирішити цю проблему, ми зберігаємо об'єкт WebSocket-з'єднання в глобальному сховищі, а потім використовуємо на головній сторінці користувача. Також, це дозволяє визначити так звані callback-функції в тілі головної компоненти, що досить важливо, адже на інших сторінках ми не маємо доступу до масивів чатів та повідомлень.

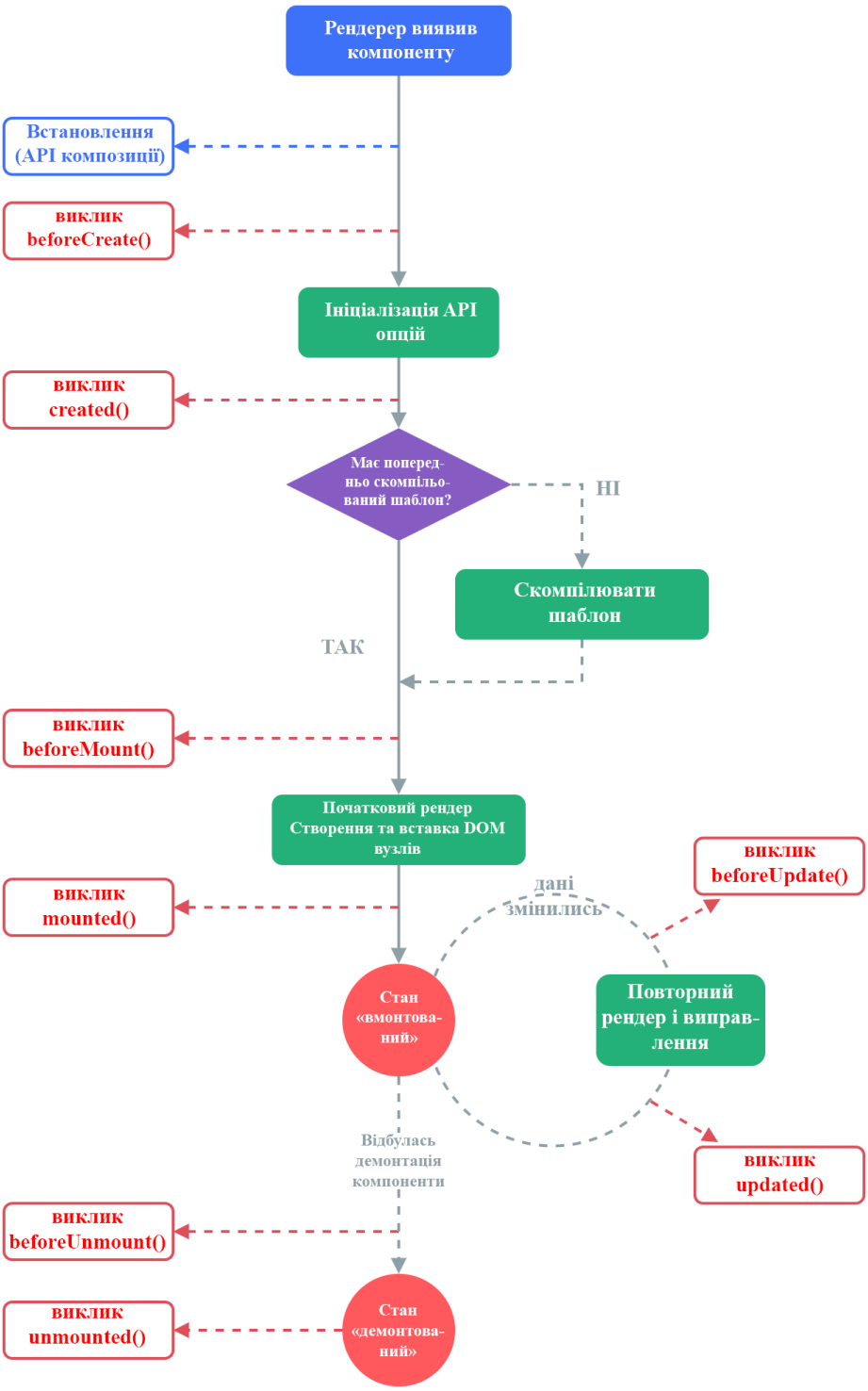

Рисунок 6 - Життєвий цикл компоненти

5.2 Збереження даних на стороні клієнта

Не дивлячись на те, що користувачу не потрібно постійно зберігати всю історію повідомлень та всі чати, йому все ще потрібно зберігати історію

повідомлень з поточним обраним користувачем та інші дані, такі як ключі, чати в яких він приймав участь, тощо. Дані можуть зберігатись як всередині компоненти Vue, так і в глобальному веб-сховищі. Для глобального сховища було використано бібліотеку Vuex. В ньому зберігаються ключі користувача, його логін та об'єкт WebSocket, який зберігає з'єднання з сервером. Сюди також можна додати історію повідомлень з усіх останніх чатів, але це б значно підвищило б використання пам'яті на пристрої користувача, тому від цього довелось відмовитись. Зручною властивістю цього сховища для розробника є його документованість – не можна виконати дію зі змінною, окрім її отримання, не записавши це в історію. Сховище змінюється, коли викликається певна дія (Action) – це функція, яка викликає commit, або – мутацію, яка документує дію та змінює стан сховища, в нашому випадку – поле, яке ми хотіли змінити. Приклад цього циклу зображений на рисунку 7.

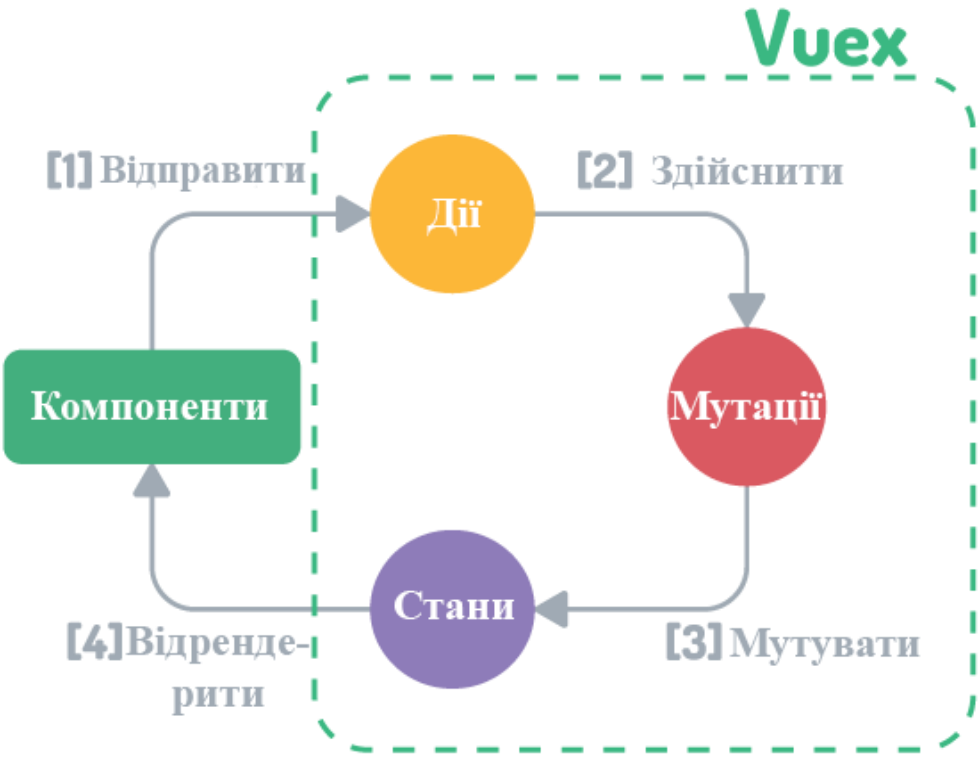

Рисунок 7 - Цикл взаємодії зі сховищем Vuex

Кожна компонента Vue зберігає певну інформацію, яка потрібна для відображення. Головна сторінка користувача зберігає інформацію про чати в яких брав участь користувач, ім'я поточного користувача, з яким відкритий

чат, історію повідомлень цього чату, спільний ключ шифрування та інше. Компонента повідомлення зберігає розшифроване повідомлення та його автора. Компоненти реєстрації та авторизації зберігають лише введені дані користувача, тобто логін та пароль, а усі одноразові числа зберігаються лише всередині відповідних функцій, які їх потребують.

5.3 Процес реєстрації та авторизації користувача

При створенні акаунту користувач повинен заповнити всього два поля – логін та пароль, зміна якого не передбачається в додатку. Після натискання відповідної кнопки реєстрації, клієнт відправляє введений логін та випадкове велике число, яке потрібне для автентифікації сервера. Сервер перевіряє, чи існує користувач з таким іменем і, в позитивному випадку, підписує nonce, генерує ще одне 64-бітне велико число та відправляє користувачу позитивну відповідь. У цьому випадку користувач генерує ключову пару, використовуючи надану сіль та свій пароль за допомогою функції PBKDF2, яку надає бібліотека jscrypto та підписує значення нуля (початкове значення logins) і надсилає це серверу. Сервер перевіряє підпис і у позитивному випадку – реєстрація закінчилась. Діаграма протоколу реєстрації зображена нижче на рисунку 8.

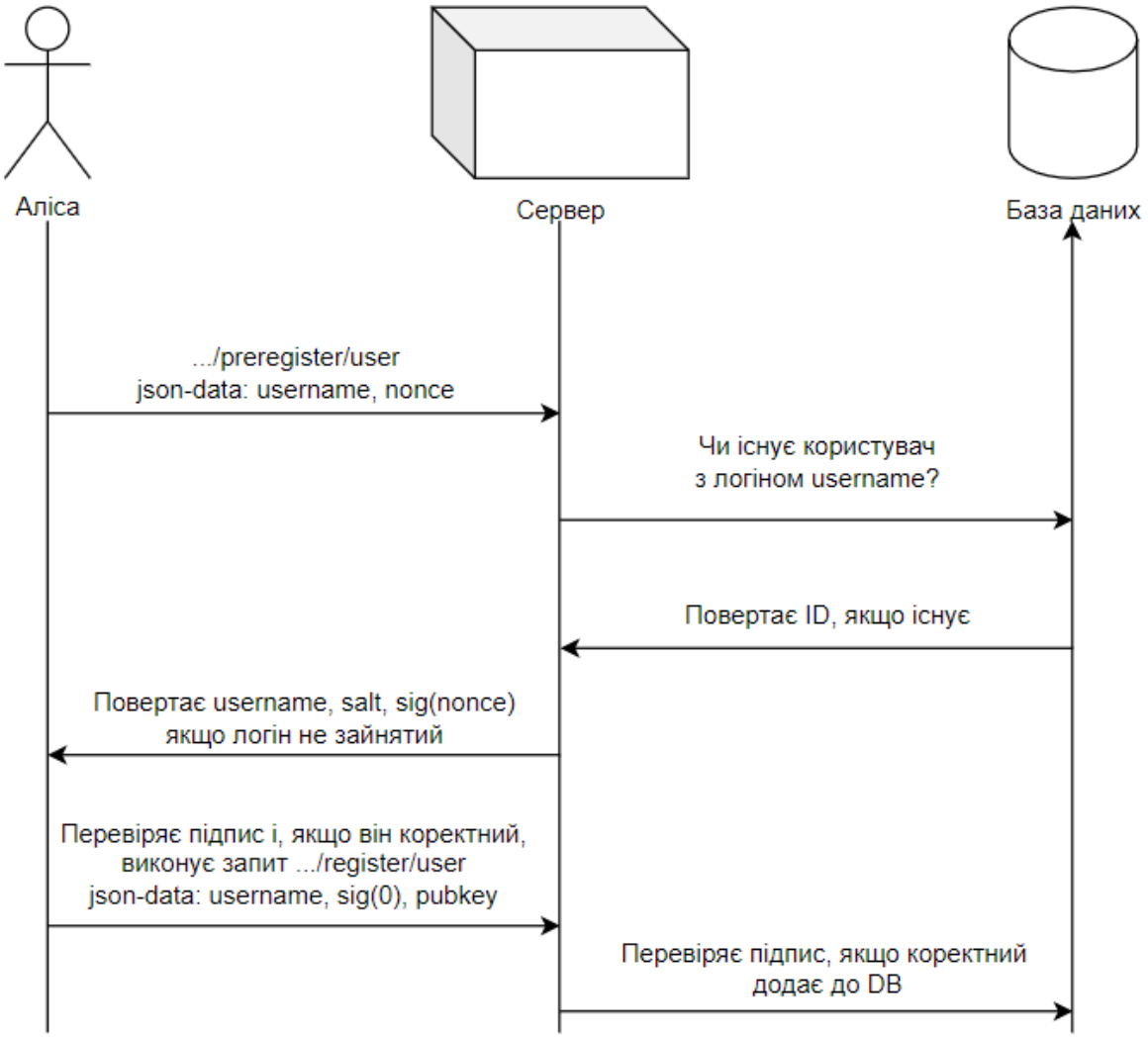

Рисунок 8 - Протокол реєстрації користувача

Для авторизації ж – користувач вводить логін, та відправляє запит на сервер, який повертає криптографічну сіль та значення, яке потрібно підписати в цей раз, у випадку існування користувача з таким логіном. Він використовує сіль та пароль, для генерації ключової пари (такої ж самої, як і при реєстрації). Після цього він підписує потрібне число та відправляє запит на відкриття WebSocket з'єднання, додаючи до URL своє ім'я та підпис. Сервер перевіряє ці дані та у позитивному випадку приймає з'єднання. Користувач перенаправляється на головну сторінку. Діаграма протоколу авторизації зображена на рисунку 9.

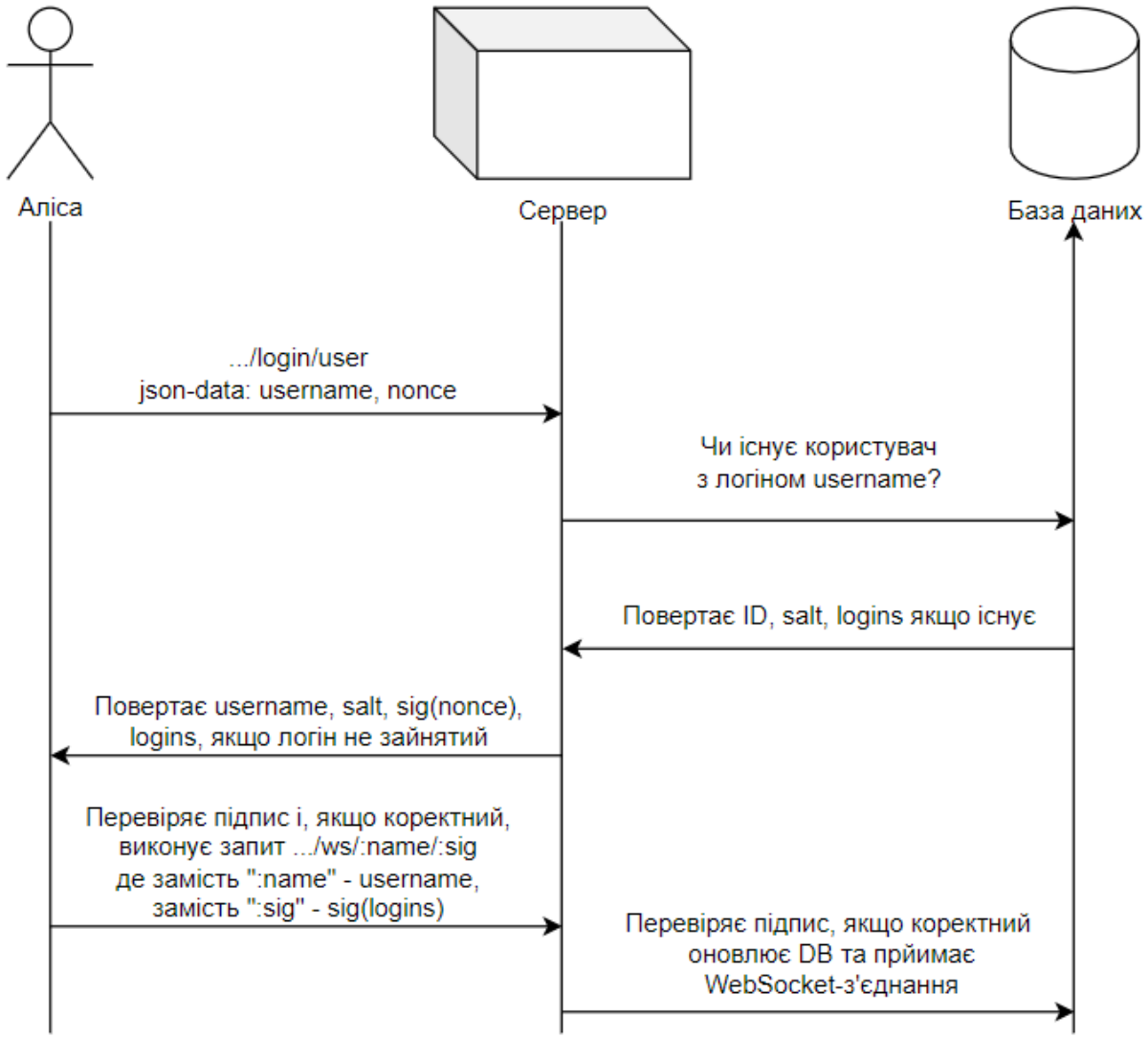

Рисунок 9 – Протокол авторизації користувача

5.4 Взаємодія користувача з системою

Після створення акаунту та авторизації клієнт отримує доступ до основних функцій додатку – обміну повідомленнями (див. рис. 10). Щоб знайти співбесідника, користувачу потрібно правильно ввести його логін, який може бути отриманий при зустрічі, поштою, або іншими доступними методами. Користувач надсилає серверу запит і у відповідь отримує публічний ключ цього користувача, якщо він був знайдений. Цей ключ, разом з приватним ключем користувача використовується для генерації спільного за допомогою протоколу Діффі-Геллмана на еліптичних кривих (ECDH).

Отриманим результатом буде точка на еліптичній кривій, тому її ікс-координата хешується функцією SHA-256 і це хеш-значення використовується як ключ для алгоритму шифрування AES. Це все виконується клієнтом, а не користувачем, він же в свою чергу може починати писати повідомлення співбесіднику.

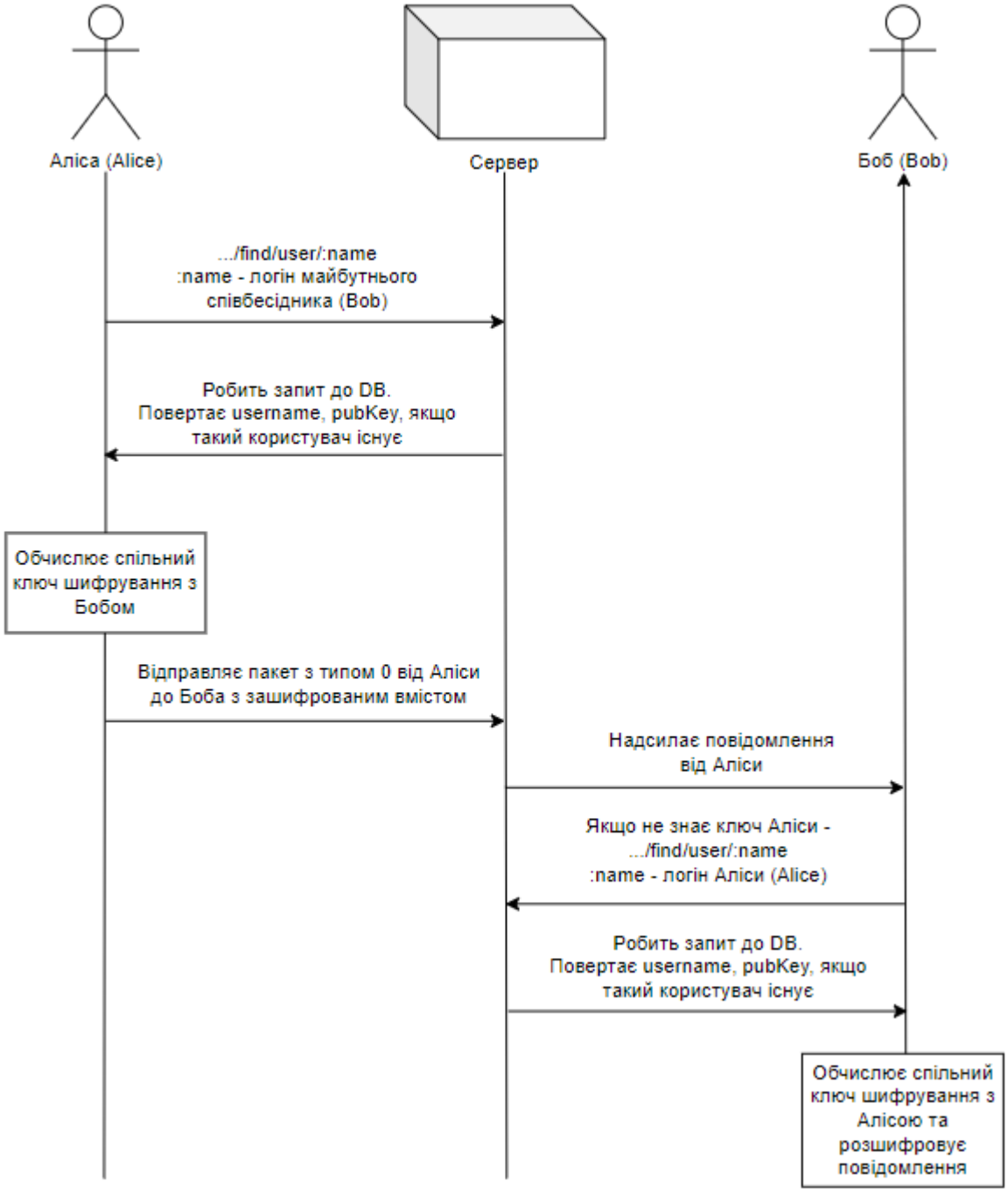

Рисунок 10 – Протокол обміну повідомленнями між користувачами.

Отримуючи повідомлення від невідомого користувача, клієнт додає його до списку чатів, і, у випадку відкриття цього чату користувачем, так само генерує з ним спільний ключ.

Щоб сторінка користувача не займала багато пам’яті, початкового завантажується невелика кількість чатів – 5, та невелика кількість повідомлень у відкритому чаті – 10. За потреби, користувач може завантажити більше даних за допомогою відповідних кнопок.

ВИСНОВКИ

Було проаналізовано існуючі рішення у сфері систем обміну миттєвими повідомленнями, і на основі цього дослідження була розроблена система з наскрізним шифруванням. Основний функціонал додатку – обмін повідомленнями між двома користувачами та отримання таких повідомлень. Усі вони зберігаються на боці сервера у зашифрованому вигляді, тобто сервер не може їх прочитати, якщо буде діяти відповідно до протоколу. Було спроектовано базу даних для збереження даних про повідомлення, користувачів та чати між ними. За допомогою криптографії було реалізовано протоколи створення ключів користувача з його паролю, створення спільних ключів між співбесідниками, автентифікація сервера та користувача, який бажає увійти в свій акаунт. За допомогою фреймворку Vue.js було спроектовано дизайн та створено веб-клієнт, який надає зручний функціонал користувачу для виконання вищезазначених дій. Система може бути використана як незалежний месенджер, або ж, як будівельний блок для створення більш великих систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Месенджер Telegram [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/
2. Месенджер WhatsApp [Електронний ресурс] – Режим доступу до ресурсу: https://www.whatsapp.com/
3. Месенджер Viber [Електронний ресурс] – Режим доступу до ресурсу: https://www.viber.com/
4. Месенджер Signal [Електронний ресурс] – Режим доступу до ресурсу: https://signal.org/
5. Мова програмування Golang [Електронний ресурс] – Режим доступу до ресурсу: https://go.dev/
6. Фреймворк Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/
7. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/
8. Schneier B. Applied Cryptography, Second Edition: Protocols, Algorthms, and Source. Code in C / Bruce Schneier – John Wiley & Sons, Inc., 1996
9. Mao W. Modern Cryptography: Theory and Practice / Wenbo Mao – Prentice Hall PTR, 2003
10. Elliptic Curve Digital Signature Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s10623-003-6154-z