Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

«Допущено до захисту»
В.о. завідувача кафедри БІСТ
Ольга МЕЛКОЗЬОРОВА

_____

« » червня 2024 р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Чат-застосунок для захищеного обміну даними»

оцінка « »		Керівник	к.т.н. Громико І.О.
Голова ЕК		Рецензент	д.т.н. Краснобаєв В.А.
Лемешко О.В.	_____	Виконавець:	студент групи КБ-41 Ніколаєнко В.О.

Харків – 2024

РЕФЕРАТ

Пояснювальна записка до проекту бакалавра містить 48 сторінок, 19 рисунків, 2 формули, 8 посилань на джерела.

Мета роботи полягає в дослідженні безпеки сучасних месенджерів і розробці більш захищеної версії чат застосунку для безпечного обміну даними в мережі Інтернет.

Об'єкт дослідження – безпека інформації при обміну повідомленнями і при зберіганні даних при використанні месенджерів.

Предмет дослідження – інструменти та методи, які використовуються для захищення даних користувачів в месенджерах.

Основними методами досліджень є визначення головних проблем, пов’язаних з захистом інформації в месенджерах, аналіз їх загального вирішення і реалізація більш захищеної версії чат-застосунку.

У роботі досліджено: безпеку даних при пересиланні повідомлень мережею, безпеку даних при їх збереженні в базі даних серверу, а також захист даних при отриманні доступу до пристрою, який брав участь в обміні повідомленнями.

Результати роботи можуть бути використані у різних наукових виданнях, а також для кращого розуміння безпеки інформації при використанні месенджерів.

Ключові слова: МЕСЕНДЖЕР, ШИФРУВАННЯ, СЕРВЕР, КЛІЄНТ, ЗАХИСТ ІНФОРМАЦІЇ В МЕСЕНДЖЕРАХ, НАСКРІЗНЕ ШИФРУВАННЯ, ОБМІН ДАНИМИ.

РЕФЕРАТ

The explanatory note to the bachelor's project contains 48 pages, 19 pictures, 2 formulas, 8 references.

The purpose of the work is to research the security of modern messengers and develop a more secure version of the chat application for secure data exchange on the Internet.

The object of research is the security of information when exchanging messages and storing data when using messengers.

The subject of the research is the tools and methods used to protect user data in messengers.

The main research methods are to identify the main problems related to the protection of information in messengers, analyze their general solution and implement a more secure version of the chat application.

The research investigates: data security when sending messages over the network, data security when storing them in the server database and data security when accessing a device that participated in the message exchange.

The results of the work can be used in various scientific publications, as well as for a better understanding of information security when using messengers.

Keywords: MESSENGER, ENCRYPTION, SERVER, CLIENT, INFORMATION SECURITY IN MESSENGERS, END-TO-END ENCRYPTION, DATA EXCHANGE.

ЗМІСТ

ВСТУП 5
1 БЕЗПЕКА ДАНИХ КОРИСТУВАЧІВ В СУЧАСНИХ МЕСЕНДЖЕРАХ 7
1.1 Загрози даним користувачів в сучасних месенджерах і їх загальне вирішення 7
1.2 Недоліки вирішення проблем безпеки даних в сучасних месенджерах та ідеї для їх усунення 13
1.3 Інструменти, використані при створення захищеного чат-застосунку . 15
2 ГОЛОВНІ БЕЗПЕКОВІ ФУНКЦІЇ ЧАТ-ЗАСТОСУНКУ 17
2.1 Шифрування повідомлень в чат-застосунку 17
2.1.1 Теоретичні відомості про використаний метод шифрування AES. 17
2.1.2 Реалізація методу шифрування AES при розробці чат-застосунку 21
2.2 Вирішення проблеми з отриманням доступу до даних користувачів третіми сторонами 27
2.3 Вирішення проблеми несанкціонованого отримання доступу до акаунту користувача 29
3 РЕАЛІЗАЦІЯ ЧАТ-ЗАСТОСУНКУ. АРХІТЕКТУРА І ІНТЕРФЕЙС 36
3.1 Файлова структура чат-застосунку 36
3.1.1 Архітектура клієнтської частини чат-застосунку 36
3.1.2 Архітектура серверної частини чат-застосунку 39
3.2 Огляд інтерфейсу чат-застосунку 40
4 КОНКУРЕНТНІСТЬ І ПОКРАЩЕННЯ ЧАТ-ЗАСТОСУНКУ 45
4.1 Конкурентність розробленого чат-застосунку 45
4.2 Можливі покращення розробленого чат-застосунку 46
ВИСНОВКИ 48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50

ВСТУП

У епоху цифрової комунікації, де швидкість і доступність отримання даних і інформації важливіші, ніж будь-коли, виникає безпрецедентна потреба в захисті приватності та безпеці при онлайн-спілкуванні. Впровадження все вищого рівня захисту даних користувачів месенджерів — це відповідь на зростаючу потребу у збереженні конфіденційності користувачів під час їхніх онлайн-спілкувань. В цьому контексті виникає необхідність в розробці технологій, які забезпечують шифрування, захист від перехоплення даних, та інші заходи безпеки, при цьому важливо, щоб у месенджера була мінімальна кількість інформації про користувачів, яку він може оброблювати або передавати.

Сектор безпеки сучасних месенджерів перебуває в постійній динаміці. Основні виклики, з якими стикаються месенджери, включають захист приватності користувачів, боротьбу зі зловживаннями даними, забезпечення безпеки від кібератак, а також відповідність законодавству щодо обробки особистих даних.

Наразі багато месенджерів використовують шифрування для захисту переписки від несанкціонованого доступу. Такі заходи дозволяють забезпечити високий рівень конфіденційності, але в той же час породжують питання щодо можливості дотримання законів про боротьбу зі злочинністю та тероризмом, які можуть вимагати доступу до даних для розслідування.

Крім того, месенджери стикаються з викликами щодо фільтрації контенту, боротьби з дезінформацією та кібербулінгом, що потребує вдосконалення алгоритмів модерації та співпраці з експертами з різних галузей.

Загалом, сектор безпеки месенджерів постійно еволюціонує і спрямований на забезпечення максимально можливого захисту користувачів і їх даних та збереження їхньої приватності в умовах швидкозмінного

цифрового середовища і при потребі в відповідності регіональним законодавствам.

Відповідно до зазначеного, метою роботи є створення захищеного чат-застосунку, при використанні якого дані, що пересилаються, будуть в максимальній безпеці.

Для досягнення цієї мети доведеться впустити таки зручні для користувача функції, як збереження історії чатів, або створення облікових записів. Дані повинні бути захищені за допомогою найефективнішого на час написання роботи методу шифрування даних як при їх пересиланні мережею, так і при їх тимчасовому зберіганні на сервері до їх видалення, при цьому вміст повідомлень, що зберігаються, не повинен бути доступним для серверу, так само сервер має отримувати мінімальний набір даних про пристрій користувача і мінімальні метадані, необхідні для функціонування застосунку. Також застосунок повинен бути максимально захищеним від отримання зловмисниками доступу до вже відправлених повідомлень.

1 БЕЗПЕКА ДАНИХ КОРИСТУВАЧІВ В СУЧАСНИХ МЕСЕНДЖЕРАХ

1.1 Загрози даним користувачів в сучасних месенджерах і їх загальне вирішення

В сучасному світі існує безліч месенджерів, частина з них вже отримала свою постійну аудиторію і зайняла певні ніші ринку. Всі ці месенджери мають як перелік переваг над конкурентами, так і перелік недоліків. Але загально серед них можна виділити одну спільну проблему: всі вони є комерційними проектами, які, в першу чергу, розроблюються для того, що приносити прибуток компанії-розробнику впродовж підтримки цих застосунків шляхом залучення нової аудиторії і втримання вже набутої для подальшої її монетизації. І саме по собі це не є проблемою, однак звідси випливає наступне: при розробці компанія має знаходити певний баланс між безпекою даних користувачів, зручністю використання месенджеру для кінцевого користувача і залишенням собі доступу до якихось даних для подальшого адміністрування та цензурування.

Перша проблема, яку породжує знаходження цього балансу, це, одночасно, і найочевидніша, і найважливіша проблема – недостатній захист повідомлень, які відправляються через мережу, або зберігаються на сервері. Ця проблема полягає в тому, що повідомлення можуть бути перехоплені і прочитані під час їх транспортування з пристрою одного користувача до серверу, під час транспортування з серверу до адресата повідомлення, або безпосередньо з серверу, коли повідомлення зберігається на ньому, що дає змогу зловмисникам потенційно отримати доступ до приватної інформації. В сучасних месенджерах проблема нешифрування даних відсутня повністю, дані в тому чи іншому вигляді шифруються при пересиланні мережею. Але існують два основні способи шифрування даних: перший – це «end-to-end» шифрування, а другий – шифрування від клієнта до серверу.

Метод наскрізного шифрування («від кінця до кінця») є найбезпечнішим для користувача, бо реалізує шифрування даних при їх відправці і

розшифрування безпосередньо при прибутті до іншого користувача. Це реалізується за рахунок того, що при створенні облікового запису користувача у нього на пристрої створюється його приватний ключ, а на сервері – його публічний. При відправці іншим користувачем повідомлення, воно шифрується за допомогою публічного ключа, який береться з серверу, після чого встановлюється безпечне з'єднання між двома користувачами через сервер, що означає, що поки повідомлення не дійде до адресату, воно буде в зашифрованому вигляді без можливості розшифрування. Коли це повідомлення дійде до пункту призначення, воно буде розшифровано приватним ключем користувача, який має лише кінцевий пристрій цього користувача. Відповідно, подібний спосіб вирішення проблеми недостатнього захисту в мережі є найефективнішим, бо не тільки захищає інформацію при транспортуванні від користувача до серверу, а впродовж всього часу, поки вона досягає адресата. Більшість сучасних месенджерів мають наскрізний метод шифрування, застосований до приватних чатів (двоє співрозмовників).

Інший популярний метод, шифрування від клієнту до серверу, тобто шифрування на транспортному рівні, полягає в тому, що мережею передається зашифроване повідомлення, але воно як шифрується ключами з серверу, так і містить на сервері ключі для розшифрування, що потенційно дає власнику серверу можливість отримати доступ до вмісту повідомлень, або ж вміст цих повідомлень може бути скомпрометований шляхом витоку даних месенджера. За таким принципом працює більшість групових чатів або каналів сучасних месенджерів.

Попри те, що, на перший погляд, наскрізне шифрування є однозначно переважним, лише нещодавно месенджери почали переходити на нього, а більшість і досі не користується ним в групових чатах. Причина невикористання наскрізного шифрування в групових чатах має технічне пояснення: по-перше, якщо чат є груповим і його можна знайти у відкритому доступі, то зникає необхідність подібного захисту, бо відповідальність за непоширення конфіденційної інформації покладається на учасників чату, які

мають розуміти, що до чату у відкритому доступі може приєднатись будь хто. Але навіть при цьому, повідомлення відправляються мережею у зашифрованому вигляді, використовуючи шифрування від клієнту до серверу. По-друге, через сам спосіб наскрізного шифрування в групових чатах він потребує більше ресурсів пристрою відправника для шифрування, бо без оптимізації алгоритму цьому пристрою необхідно буде шифрувати повідомлення для кожного учаснику чату його публічним ключем окремо, що може займати більше часу. Але навіть попри більшу складність в обчисленнях, багато сучасних месенджерів дають змогу створювати групові чати з наскрізним шифруванням, вирішуючи цю проблему шляхом обмеження максимальної кількості учасників такою кількістю, яка є оптимальною для швидкої роботи алгоритму шифрування.

Підбиваючи підсумок, проблема захисту повідомлень при їх транспортуванні або зберіганні від отримання несанкціонованого доступу до них майже в усіх сучасних месенджерах вирішується шляхом використання способу наскрізного шифрування, що робить можливим отримання доступу до нешифрованого вмісту повідомлень лише для співрозмовників.

Друга проблема безпеки конфіденційних даних в сучасних месенджерах – це використання даних користувачів месенджерів або їх чатів в інтересах розробників цих месенджерів, або ж узгоджена передача даних третім особам. Ця проблема є однією з причин того, чому багато месенджерів не поспішали з переходом на наскрізне шифрування навіть в приватних чатах, допоки це не стало необхідним для утримання аудиторії і конкурування з іншими месенджерами, які подібне шифрування ввели.

Обробку і аналіз таких конфіденційних даних, як вміст повідомлень користувачів, розробники месенджерів могли використовувати, наприклад, для персоналізації реклами. Проте основною причиною була відповідність регіональним законодавствам. Уряди багатьох країн час від часу подають запити для доступу до приватної інформації, яку мають месенджери, при цьому погрожуючи санкціями, штрафами, або блокуванням на території

країни. З відомих широкому загалу фактів подібної передачі інформації, вона стосувалась дій, міжнародно визнаних злочинними, але сам факт передачі даних про приватні повідомлення між користувачами робить месенджер небезпечним для передачі конфіденційної важливої інформації.

Проблема несанкціонованої обробки чи передачі даних користувачів третім особам була частково вирішена введенням наскрізного шифрування, завдяки чому інформація про вміст повідомлень перестала бути доступною всім сторонам, окрім співрозмовників чату. Проте, ця проблема існує і досі, бо навіть при наскрізному шифруванні месенджер має доступ до метаданих повідомлень, таких як дата і час надсилання повідомлення, його адресати, обсяг переданих даних, тощо; до технічних даних користувачів, таких як тип пристрою, IP-адреса пристрою, версія операційної системи та інші технічні характеристики пристрою; до інформації про акаунт, такої як ім’я користувача, його номер телефону, електронна адреса та інші відомості, які користувачі надають при реєстрації і які можуть бути збережені месенджером для управління обліковим записом та забезпечення послуг. В сучасності вже були зафіксовані випадки передачі інформації месенджерами, які використовують наскрізне шифрування, третім особам.

Підбивши підсумок за проблемою використання даних користувачів третіми сторонами, яке санкціоноване розробниками месенджерів, то вирішенням проблеми для більшості месенджерів є введення наскрізного шифрування, але воно слугує більше як гарантія для користувача, що навіть при необхідності месенджер не зможе отримати доступ до вмісту повідомлень. Також, в певному сенсі, для зменшення недовіри до месенджерів, яка може бути викликана можливою передачею ними даних, деякі месенджери публікують у відкритому доступі випадки передачі інформації ними, а саме з якою метою і кому вона була передана. Але варто зазначити, що для зручності користування месенджерами вони мають такі функції, як можливість створення облікового запису, або історія чатів. У зв’язку з цим, у всіх сучасних месенджерів все одно є та чи інша інформація, незалежно від типу

шифрування, яку, за необхідності, вони можуть обробляти або передавати третім сторонам.

Третя проблема – це проблема втрати доступу до облікового запису користувача або отримання зловмисником несанкціонованого доступу до акаунту. Серед способів, що використовуються зловмисниками для цього, виділяють такі як фішинг, або ж використання соціальної інженерії для спроби омани користувача з метою отримати його персональні дані, що використовуються для входу в акаунт. Це може включати надсилання підроблених електронних листів або повідомлень, що виглядають як офіційні сповіщення від месенджера. Ці повідомлення можуть просити користувачів ввести свої облікові дані або надати доступ до свого облікового запису, наприклад, під виглядом оновлення безпеки або перевірки ідентифікації. Також, окрім цього, зловмисники можуть використовувати функцію ботів, яка присутня в деяких сучасних месенджерах, для маскування під корисні додатки для користувачів, при цьому запитуючи в них доступ до їх конфіденційних даних для нібито успішної роботи, в результаті чого отримуючи несанкціонований доступ до цих даних, про що користувач може навіть не здогадуватись. Проблему фішингу важко вирішити для розробників месенджерів, бо кількість підроблених акаунтів або шахрайських ботів може бути необмеженою, що робить неможливим їх завчасне виявлення і блокування, проте кроки для мінімізації ефективності цього способу все одно приймаються. Серед дій, що можуть застосовуватись, є такі, як автоматичне фільтрування вмісту повідомлень, якщо не використовується наскрізне шифрування, з подальшим повідомленням користувача про потенційну небезпеку, використання алгоритмів для відстеження потенційно небезпечних акаунтів, так само з повідомленням користувача, а також автоматичне оповіщення месенджером користувачів щодо небезпеки і методів, що використовуються при фішингу, з метою надання користувачам порад, як не стати його жертвою. Але одним з найефективніших способів боротьби з фішингом є впровадження користувачем двоетапної автентифікації, яка є

гарантією того, що, навіть при компрометації особистих даних користувача, таких як логін, пароль, електронна пошта, номер телефону, тощо, зловмисник все одно не матиме можливості отримати доступу до акаунту, бо той буде додатково захищений за допомогою, найчастіше, іншого облікового запису користувача.

Другим небезпечним способом отримання несанкціонованого доступу до акаунтів користувачів є перехоплення кодів підтвердження. Деякі месенджери для підтвердження входу з нового пристрою використовують коди, які генеруються випадково і надсилаються на електронну пошту або мобільний номер користувача, які були прив’язані до акаунту при реєстрації. Можливим перехоплення коду з електронної скриньки користувача робить те, що більшість популярних електронних пошт мають поганий захист як інформації, так і даних користувача. За рахунок цього і того, що електронні скриньки підв’язуються до багатьох сторонніх сайтів, які можуть мати погані безпекові алгоритми, багато електронних адрес користувачів є скомпрометованими і якщо не злитими в вільний доступ, то виставленими зловмисниками на продаж. З мобільних пристроїв коди підтверджень можуть бути отримані за рахунок перехоплення SMS-повідомлень, бо вони не шифруються при відправленні, а також за рахунок попереднього зараження пристрою вірусним програмним забезпеченням, яке може надавати зловмисникам доступ до SMS-повідомлень, які отримує цей пристрій. Захистом, який більшість месенджерів використовує для мінімізації ризиків перехоплення кодів підтвердження, є двоетапна автентифікація, а саме така, яка реалізована за допомогою сторонніх спеціалізованих на безпечній генерації кодів програм, або розроблена спеціально для захисту облікових записів розробниками самого месенджеру.

Останній спосіб отримання доступу до акаунтів користувачів є таким, на які месенджер може лише реагувати у взаємодії з користувачем, бо попередити його дуже важко: це отримання зловмисниками пристрою користувача, на якому вхід в акаунт вже виконаний. В такому випадку розробники месенджеру

не мають можливості попередити несанкціонований доступ до інформації з чатів, і вся відповідальність за це лягає на самого користувача (він може встановити пароль на пристрій, пароль на застосунок, тощо). Проте навіть для таких ситуацій у месенджерах передбачено механізми завчасного попередження загрози, такі як можливість встановити таймер автоматичного видалення для повідомлень або файлів, які мають залишатись конфіденційними, та механізми реагування на ситуацію, яка вже відбулась – через технічну підтримку, або, в деяких месенджерах, навіть просто з облікового запису користувача на іншому девайсі можна закрити сесію на втраченому пристрої, що при прийнятих завчасно мірах безпеки з боку користувача і за умови швидкого реагування на проблему може вберегти конфіденційність важливої інформації.

Підбивши підсумок з розглянутої проблеми отримання доступу зловмисником до облікового запису користувача, вирішенням проблеми з отриманням даних, які використовуються для входу в акаунт є введення якісної двоетапної автентифікації, яка використовує спеціалізовані застосунки чи алгоритми, які роблять отримання коду неможливим для третіх осіб. Для вирішення проблеми втрати пристрою, на якому виконаний вхід в акаунт, користувачу надається можливість встановлювати таймер на важливі повідомлення і закрити сесію на втраченому пристрої, використовуючи інший пристрій, на якому виконано вхід в той самий акаунт.

1.2 Недоліки вирішення проблем безпеки даних в сучасних месенджерах та ідеї для їх усунення

Проблема недостатнього захисту повідомлень при пересиланні мережею або при їх зберіганні сучасними месенджерами вирішується шляхом впровадження наскрізного шифрування. Це є найкращим методом вирішення цієї проблеми на сьогоднішній день, але недоліки полягають в реалізації функції наскрізного шифрування, а саме в тому, що в багатьох месенджерах ця функція за замовчуванням працює лише в приватних чатах, а в деяких

месенджерах її треба вмикати вручну. Враховуючи специфіку роботи, а саме створення захищеного чат-застосунку, захист даних повинен мати найвищий пріоритет, тому найкращим варіантом вирішення проблеми, що розглядається, буде впровадження наскрізного шифрування за замовчуванням для усіх чатів, як приватних, так і групових. Це дасть користувачу можливість не витрачати час на те, щоб налаштовувати безпекові функції перед початком обміну повідомленнями, а також зробить неможливими потенційні помилки при цьому налаштуванні, надаючи користувачам впевненість в тому, що ніхто, окрім співрозмовників, не зможе отримати доступ до їх повідомлень.

Проблему обробки і передачі даних третім особам зі сторони месенджеру вирішують так само за допомогою впровадження наскрізного шифрування, але воно направлене лише на захист вмісту повідомлень, відповідно, більшість месенджерів мають доступ до різної інформації про користувачів та чати, такої як метадані повідомлень, та частина даних облікових записів користувачів, яка зберігається в незашифрованому вигляді (електронні пошти, номери телефонів, імена користувачів, тощо). Враховуючи мету, з якою розроблюється чат-застосунок для роботи, він має впроваджувати максимальний захист даних користувача, незважаючи ні на що, відповідно, найкращим і найбезпечнішим варіантом буде оминати збереження будь яких даних про користувачів, окрім необхідних для відправки повідомлень, тобто створити чат-застосунок з використанням не акаунтів користувачів, а токенів для створення і підключення до чатів. В якості додаткових заходів безпеки, користувачу на кожну сесію, тобто кожен раз, коли він запускатиме застосунок, буде надаватись унікальний ідентифікатор, а творець чату зможе приймати або відхиляти спробу приєднання до цього чату, базуючись на цьому унікальному ідентифікаторі, який він буде бачити при спробі приєднання до чату.

Проблему отримання несанкціонованого доступу зловмисниками до акаунту користувача більшість месенджерів вирішує шляхом впровадження якісної двоетапної автентифікації, а мірами захисту від втрати пристрою, на

якому виконаний вхід в акаунт, є можливість встановлювати таймер на видалення повідомлення і можливість виходу з акаунту за допомогою інших пристроїв, на який виконаний в цей акаунт вхід. Недоліками такого вирішення проблеми є те, що фішингові атаки можуть обходити навіть складні двоетапні автентифікації шляхом, наприклад, запросивши введення номеру телефону чи адреси електронної скрині на підроблених сайтах, після чого ініціювавши вхід, за умови наявності всіх необхідних даних, і попросивши ввести код підтвердження, що надійшов від месенджеру, таким чином отримавши повний доступ до акаунту користувача і подальшу можливість або вкрасти акаунт, змінивши всі ідентифікаційні дані користувача, або продовжити спостерігати за акаунтом, маючи доступ до всіх його повідомлень. Також недоліком є те, що користувачі часто не сприймають можливість втрати пристрою як реальну небезпеку, тому таймер на видалення важливих повідомлень може ставитись не завжди, а для дистанційного закриття сесії на втраченому пристрої в деяких ситуаціях може бути надто пізно, зловмисник може отримати необхідні дані до того моменту. Вирішення цих проблем пропонується наступне: проблему фішингу вирішує створення токенного чату, замість чату з обліковими записами, а проблему отримання пристрою зловмисником - впровадження автоматичного видалення чатів при відсутності активності обох користувачів певний проміжок часу.

Такі рішення зроблять з чат-застосунку застосунок для безпечного обміну конфіденційними даними, який за замовчуванням буде впроваджувати такі заходи безпеки, які дозволять користувачам не хвилюватись про потенційну небезпеку приватності обміну повідомленнями.

1.3 Інструменти, використані при створення захищеного чат-застосунку

Для досягнення мети роботи будуть використовуватись таке шифрування, як AES (Advanced Encryption Standard), яке реалізовується за допомогою мови програмування JavaScript, а саме бібліотеки CryptoJS, яка дає можливість генерувати ключі для AES шифрування і проводити саме

шифрування і дешифрування повідомлень. Сам алгоритм шифрування є симетричним і реалізує наскрізний тип шифрування, що означає, що ключі зберігаються лише на клієнтський частинах застосунків і не передаються мережею.

Також для реалізації захищеного чат-застосунку будуть використані наступні інструменти web-розробки:

- HTML (HyperText Markup Language);
- CSS (Cascading Style Sheets);
- JavaScript;
- Webpack;
- Node.js;
- Express;
- Electron;
- TypeScript;
- WebSocket;
- Socket.IO;
- Mongoose.

В наступних розділах на використанні цих інструментів не буде наголошено, проте їх використання є повсюдним під час реалізації чат-застосунку і дозволяє розробці бути працездатною.

В якості бази даних для збереження необхідної інформації буде використовуватись нереляційна онлайн-база даних MongoDB, яка надає додатковий захист для даних, що зберігаються, шляхом впровадження шифрування на рівні транспортного протоколу при завантаженні даних в базу даних і при їх отриманні, так само як і надання можливість контролювати IP-адреси пристроїв, які матимуть доступ до бази даних підключатись.

В другому розділі роботи буде конкретизовано практичне вирішення наведених раніше проблем за допомогою вказаних інструментів.

2 ГОЛОВНІ БЕЗПЕКОВІ ФУНКЦІЇ ЧАТ-ЗАСТОСУНКУ

2.1 Шифрування повідомлень в чат-застосунку

2.1.1 Теоретичні відомості про використаний метод шифрування AES

Методом шифрування було обрано симетричне шифрування Advanced Encryption Standard (AES). Цей шифр розроблявся з 1997 року і був готовий до використання в 2001 році. Розробниками були бельгійські криптографи Вінсент Реймен та Йоан Дамен. Алгоритм мав початкову назву Rijndael і був розроблений під час конкурсу американського Національного інституту стандартів і технологій (NIST) і мав на меті замінити основний на той час стандарт в Сполучених Штатах Америки Data Encryption Standard (DES), який був застарілим і, з повсюдним збільшенням потужності обчислювальних машин, міг бути зламаний звичайним перебором можливих значень ключів, бо максимальна довжина ключа для нього складала всього 56 біт.

AES базується на принципі заміни-перестановки, і є ефективним як у програмному, так і в апаратному забезпеченні. AES — це метод із фіксованим розміром блоку 128 біт і розміром ключа 128, 192 або 256 біт. Більшість обчислень AES виконуються в певному кінцевому полі.

Розмір блоку - це фіксована кількість бітів або байтів, яка використовується для розділення даних на частини перед шифруванням або розшифруванням. У контексті шифрування AES розмір блоку визначає, на скільки біт або байтів будуть розбиті дані перед шифруванням.

В шифруванні AES, розмір блоку становить 128 біт (або 16 байтів). Це означає, що дані розбиваються на блоки по 128 біт перед тим, як вони піддаються операціям шифрування.

Розмір блоку важливий для безпеки і ефективності шифрування. Він визначає, які дані можуть бути оброблені за один раз, а також впливає на відповідність до стандартів безпеки і сумісність з іншими системами. У випадку AES, розмір блоку 128 біт був обраний як оптимальний компроміс між безпекою, швидкістю та ефективністю.

Розмір ключа в шифруванні означає кількість бітів або байтів, які використовуються для генерації ключа шифрування. Це важлива характеристика, оскільки від розміру ключа залежить безпека шифрування: чим більше бітів у ключі, тим складніше його зламати методами перебору.

У більшості випадків, коли говорять про розмір ключа, мають на увазі кількість бітів у ключі. Наприклад:

- Ключ довжиною 128 біт може бути представлений 16 байтами.
- Ключ довжиною 192 біти може бути представлений 24 байтами.
- Ключ довжиною 256 біт може бути представлений 32 байтами.

Ця довжина ключа визначається при генерації ключа або виборі ключа для шифрування. Обираючи ключ, обирається саме його довжина, яка визначається безпековими вимогами системи та рівнем захисту, який є необхідним.

Важливо розуміти, що зазвичай більший розмір ключа забезпечує більшу безпеку, оскільки зловмиснику буде важче зламати ключ методом перебору. Однак також потрібно враховувати, що більший розмір ключа може призвести до деякого збільшення обчислювальних витрат на обробку даних, а в контексті наскрізного шифрування, де шифрування виконується на системах користувачів, це є особливо важливим.

AES працює з матрицею 4 × 4 стовпця, що представлена у формулі 2.1, з основним порядком із 16 байтів b0, b1, ..., b15, що називається станом:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix} \qquad (2.1)$$

Кожен раунд складається з кількох етапів обробки, включаючи той, який залежить від самого ключа шифрування. Набір зворотних раундів застосовується для перетворення зашифрованого тексту назад у вихідний відкритий текст за допомогою того самого ключа шифрування.

Загальний опис алгоритму складається з таких кроків, як розширення ключа (KeyExpansion), під час якого ключі раунду отримуються з ключа шифру за допомогою розкладу ключів AES. Для AES потрібен окремий 128-бітний блок ключів для кожного раунду плюс ще один.

Після цього відбувається додавання ключа початкового раунду (AddRoundKey), де кожен байт стану поєднується з байтом раунд-ключа за допомогою побітового виключного АБО (XOR).

Залежно від довжини ключа, яка складає 128, 192 чи 256 біт, наступна послідовність дій виконується 9, 11 або 13 раундів відповідно:

- Крок нелінійної підстановки (SubBytes), де кожен байт замінюється на інший відповідно до таблиці пошуку.
- Крок транспозиції (ShiftRows), де останні три рядки стану циклічно зсуваються на певну кількість кроків.
- Операція лінійного змішування (MixColumns), яка працює над стовпцями стану, поєднуючи чотири байти в кожному стовпці.
- Додавання ключа початкового раунду (AddRoundKey).

Після цього виконується вся наведена послідовність ще раз, але вже без останнього пункту AddRoundKey.

На кроці SubBytes кожен байт $a_{i,j}$ у масиві стану замінюється на SubByte $S(a_{i,j})$ за допомогою 8-бітового поля підстановки. До раунду 0 масив стану є просто відкритим текстом/введенням. Ця операція забезпечує нелінійність шифру. Використаний S-блок отримано з мультиплікативного обернення $GF(2^8)$, який, як відомо, має хороші властивості нелінійності. Щоб уникнути атак, заснованих на простих алгебраїчних властивостях, S-блок будується шляхом поєднання оберненої функції з оборотним афінним перетворенням. S-блок також вибрано для уникнення будь-яких фіксованих точок (а також порушення), тобто $S(a_{i,j}) \neq a_{i,j}$, а також будь-яких протилежних фіксованих точок, тобто $S(a_{i,j}) \oplus a_{i,j} \neq FF_{16}$. Під час виконання дешифрування використовується крок InvSubBytes (зворотний до SubBytes), який вимагає

спочатку брати обернене до афінного перетворення, а потім знаходити мультиплікативне обернене.

Крок ShiftRows працює з рядками стану, він циклічно зсуває байти в кожному рядку на певне зміщення. Для AES перший рядок залишається без змін. Кожен байт другого рядка зсувається на один ліворуч. Подібним чином третій і четвертий рядки зсуваються на зміщення в два і три відповідно. Таким чином, кожен стовпець стану виведення кроку ShiftRows складається з байтів з кожного стовпця стану введення. Важливість цього кроку полягає в тому, щоб уникнути незалежного шифрування стовпців, у цьому випадку AES перетвориться на чотири незалежні блокові шифри.

На кроці MixColumns чотири байти кожного стовпця стану комбінуються за допомогою оборотного лінійного перетворення. Функція MixColumns приймає чотири байти як вхідні дані та виводить чотири байти, де кожен вхідний байт впливає на всі чотири вихідні байти. Разом із ShiftRows MixColumns забезпечує дифузію в шифрі.

Під час операції MixColumns кожен стовпець перетворюється за допомогою фіксованої матриці (матриця, помножена зліва на стовпець, дає нове значення стовпця в стані), як представлено у формулі 2.2:

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \quad 0 \le j \le 3 \qquad (2.2)$$

Матричне множення складається з множення та додавання елементів. Записи — це байти, які розглядаються як коефіцієнти полінома порядку x^7. Додавання — це просто виключне АБО. Множення є незвідним поліномом за модулем $x^8 + x^4 + x^3 + x + 1$. Якщо обробляється побітово, то після зсуву слід виконати умовне виключне АБО із $1B_{16}$, якщо зміщене значення більше FF_{16} (переповнення має бути виправлене відніманням генеруючого полінома). Це окремі випадки звичайного множення в $GF(2^8)$.

У більш загальному сенсі кожен стовпець розглядається як поліном над $GF(2^8)$, а потім множиться за модулем $01_{16} * z^4 + 01_{16}$ на фіксований поліном $c(z) = 03_{16} * z + 01_{16} * z + 01_{16} * z + 02_{16}$. Коефіцієнти відображаються в шістнадцятковому еквіваленті двійкового представлення бітових поліномів із GF(2)[x]. Крок MixColumns також можна розглядати як множення на показану конкретну матрицю MDS (Maximum Distance Separable) у кінцевому полі $GF(2^8)$.

На кроці AddRoundKey підрозділ поєднується зі станом. Для кожного раунду підключ виводиться з основного ключа за допомогою розкладу ключів Рейндаеля; кожен підключ має такий самий розмір, як і стан. Підключ додається шляхом поєднання стану з відповідним байтом підключа за допомогою побітового виключного АБО.

На сьогоднішній день, набір інструкцій AES інтегрований у ЦП, завдяки чому реалізує пропускну здатність у кілька ГБ/с, щоб покращити швидкість і безпеку програм, які використовують AES для шифрування та дешифрування. Незважаючи на те, що минуло 20 років з моменту його впровадження, зламати алгоритм AES перебором досі є неможливим, оскільки навіть із сучасними технологіями це займає дуже велику кількість чату. На даний момент єдиною вразливістю залишається реалізація алгоритму.

2.1.2 Реалізація методу шифрування AES при розробці чат-застосунку

Архітектура чат-застосунку буде детально описана у розділі 3, проте для пояснення як роботи, так і рівня захисту шифрування необхідно розуміти наступні речі: у проекті реалізована клієнтська архітектура, яка відповідає за вигляд, логіку і можливості застосунку, коли він встановлюється на комп’ютер, і серверна архітектура, яка відповідає за встановлення зв’язку між клієнтськими застосунками і обмін даними між ними. Сервер, в свою чергу, підключений до нереляційної онлайн-бази даних Mongodb, яка відповідає за збереження даних.

Відповідно до постанови задачі, шифрування повинно бути наскрізного (end-to-end) типу, тому виконуватись повинно на клієнтській стороні за допомогою ключів, які зберігаються лише на клієнтах, так само і з дешифруванням – інший клієнт повинен скористатись своїм ключем для розшифрування повідомлення. Сервер не повинен мати інформацію про ключі, бо вона не є необхідною для його функціонування, проте ставить під загрозу дані користувачів. Так само сервер не зберігає історію переписок навіть в зашифрованому вигляді, вони зберігаються локально на клієнтах, що тягне за собою такий момент, що при приєднанні користувача до кімнати під час переписки, він не бачитиме минулі повідомлення, що може слугувати додатковим заходом безпеки.

Шифрування повідомлень перед відправкою здійснюється в файлі “hash.ts”, який знаходиться за таким шляхом “../electron/src/render/utils/hash.ts”. Папка electron відповідає за клієнтську арктектуру проекту.

Після базового огляду архітектури проекту буде розглядатись безпосередньо шифрування. На рисунку 2.1 можна побачити базовий ключ, яким шифруються повідомлення, він об’явлений в файлі hash.ts з областю видимості на весь файл. Враховуючи симетричний тип шифрування AES, цей ключ є однаковим на всіх клієнтах і для всіх, хто встановлює чат-застосунок. Проте користувач доступу до нього не матиме, бо він матиме доступ лише до білду проекту, отримати доступ до вихідного коду в такому випадку буде неможливо. Проте можливість компрометації ключа буде врахована і прийняті міри для захисту від неї будуть наведені далі.

```
const baseKey =
  "wP/8s7+7O12Yib`mk3|@p1Z|0S'`FQda^&Wt98" +
  "_,;tgBC+a)fXkL-@<LKbO-Zu_Ce~b:&';mX*=65&.a$b=hiL" +
  '(.@TXQsVuG:(cD1FeNj2)k`=GL&,y4V@0>OJ;YEt>';
```

Рисунок 2.1 – Базовий ключ для шифрування і дешифрування повідомлень

Для виконання шифрування і розшифрування буде використовуватись бібліотека CryptoJS, яка має готові методи шифрування для низки алгоритмів, серед яких і AES. За шифрування і розшифрування відповідає написаний клас HashCore, в якому прописані два методи, один для шифрування повідомлень на виході, а інший – для розшифрування перед демонстрацією вмісту повідомлення користувачам-адресатам. Метод шифрування повідомлень продемонстровано на рисунку 2.2.

```
24  export default class HashCore {
25    public static Encrypt(message: string, chatId: string | undefined): string {
26      if (chatId === undefined) return 'error';
27      const roomKey = baseKey + chatId;
28      const newMsg = CryptoJS.AES.encrypt(
29        JSON.stringify(message),
30        roomKey,
31      ).toString();
32      console.log(newMsg);
33      return newMsg;
34    }
```

Рисунок 2.2 – Метод шифрування даних

Цей метод викликається перед відправкою повідомлення на сервер. Як аргументи він приймає саме повідомлення message типу string, ID кімнати чату для подальшої модифікації базового ключа chatId типу string, також виконується обробка ситуації, коли chatId може бути undefined. Метод повертає дані типу string. У рядку 27 відбувається модифікація базового ключа шляхом визначення нової змінної roomKey, яка буде специфічна для кожної кімнати. До базового ключа додається значення токену кімнати, відповідно, роблячи ключ унікальним для кожної з кімнат.

Безпосередньо шифрування починається з рядку 28, де визначається нова змінна newMsg, значення якої встановлюється в результаті виконання функції CryptoJS.AES.encrypt(), яка приймає першим значенням повідомлення, яке треба зашифрувати, а другим – секретну фразу для генерації 256-бітного ключа, за яким це повідомлення зашифровувати. Після виконання

методу, значення, що він повертає, приводиться до типу string, виводиться в консолі клієнта і повертається як результат виконання методу Encrypt().

Для перевірки виконання шифрування буде наведено вивід консолі на клієнті одразу після шифрування і вивід консолі на сервері перед пересиланням іншим учасниками чату, це зображено рисунку 2.3. Вхідними даними для чату є ID кімнати чату «595522» і повідомлення «test message 123».

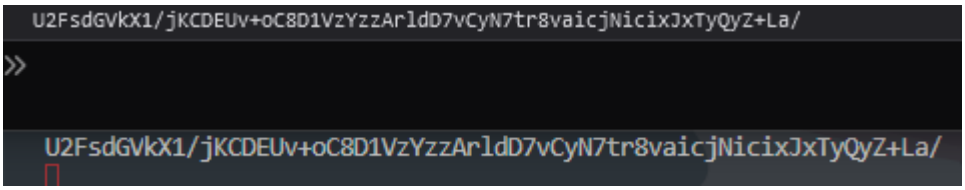

Рисунок 2.3 – Результат виведення повідомлень після шифрування на клієнті (згори) і на сервері (знизу)

На рисунку зображено хеш-код, що був утворений після шифрування вмісту повідомлення за ключем, що був згенерований за секретною фразою baseKey + «595522». Як показує цей рисунок, сервер отримує зашифровані дані, які неможливо розшифрувати без знання ключа. Щоб підтвердити, що ключа сервер не має, можна скористатись сніфером, який може перехоплювати пакети, таким як Wireshark. За правильного налаштування фільтрації він дасть змогу перехопити і побачити необхідний пакет з даними, що відправляються з клієнта на сервер. Використання сніфера представлено на рисунку 2.4, пакет, що було перехоплено, пересилав повідомлення, зашифрований вміст якого було виведено раніше на рисунку 2.3.

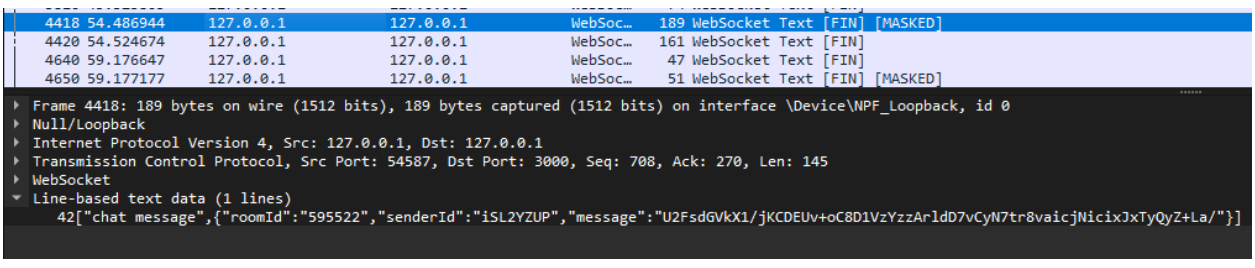

Рисунок 2.4 – Вміст перехопленого пакету після відправки повідомлення з клієнту на сервер

З зображеного на рисунку 2.4 інтерес представляє останній пункт «Line-based text data» і його тіло. Як можна побачити, з клієнту сервер отримує лише ID кімнати, ID відправника повідомлення і саме повідомлення в зашифрованому вигляді, ключі по транспортному протоколу не ходять. Відповідно, за умови коректного розшифрування на інших клієнтах, яким це повідомлення адресоване, реалізація шифрування може вважатись наскрізного типу.

Розшифрування здійснюється так само на клієнті, але вже на клієнті-адресаті. Сервер відправляє зашифроване повідомлення від користувача, ID якого було передано на нього в одному пакеті з ID кімнати, і зашифрованим повідомленням, всім учасникам кімнати з вказаним ID. На рисунку 2.5 за допомогою сніфера Wireshark показано перехоплення пакету, відправленого з сервера на клієнт.

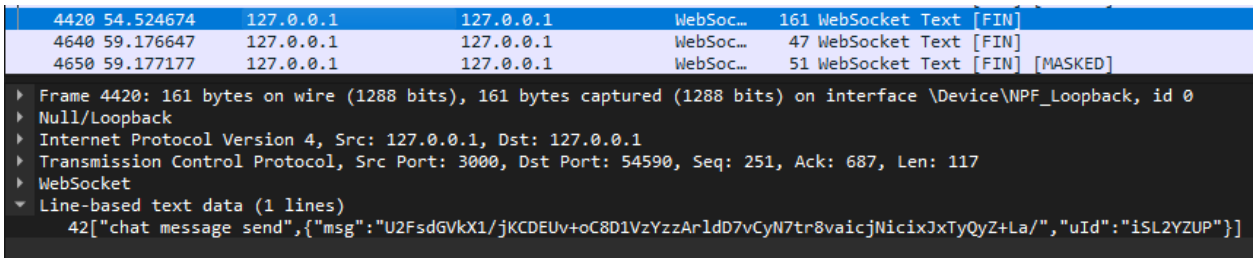

Рисунок 2.5 – Вміст перехопленого пакету після відправки повідомлення з серверу на клієнт

Пакети, ідентичні до того, який можна побачити на рисунку, будуть відправлені до кожного користувача, який знаходиться в кімнаті з переданим раніше ID. Вони містять зашифроване повідомлення і ID користувача-відправника. В результаті, під час всього циклу передачі повідомлення, воно знаходиться в зашифрованому вигляді, а ключі для його розшифрування присутні лише на клієнтах, тому шифрування є наскрізним.

Алгоритм розшифрування прописаний в тому ж класі HashCore, що і метод шифрування, він має назву Decrypt() і продемонстрований на рисунку 2.6.

```
36  public static Decrypt(message: string, chatId: string | undefined): string {
37    if (chatId === undefined) return 'error';
38    const roomKey = baseKey + chatId;
39    const newMsg = CryptoJS.AES.decrypt(message, roomKey);
40    const decryptedData = JSON.parse(newMsg.toString(CryptoJS.enc.Utf8));
41    return decryptedData;
42  }
```

Рисунок 2.6 – Метод шифрування даних

Метод під назвою Decrypt() має такі атрибути, як повідомлення message типу string і номер кімнати чату chatId, також типу string. Здійснюється перевірка на існування кімнати з вказаним ID. Метод повертає дані типу string. На рядку 38, аналогічно методу шифрування, здійснюється розрахунок ключа для розшифрування шляхом додання до значення baseKey значення ID кімнати і записуванням отриманого результату в змінну roomKey. На 39 рядку відбувається розшифрування повідомлення методом CryptoJS.AES.decrypt(), куди як аргументи відправляються спочатку зашифроване повідомлення, отримане з серверу, а потім – розрахований ключ кімнати. На виході будуть дешифровані дані, які на рядку 40 перетворюються з JSON типу у тип string, після чого відправляються як результат виконання функції Decrypt().

На рисунку 2.7 зображено коректність роботи програми після виконання повного циклу обміну повідомленнями: шифрування після відправки на одному клієнті, перехід пакету транспортним рівнем до серверу, перехід з серверу на клієнт адресата і розшифрування з виведенням у область повідомлень.

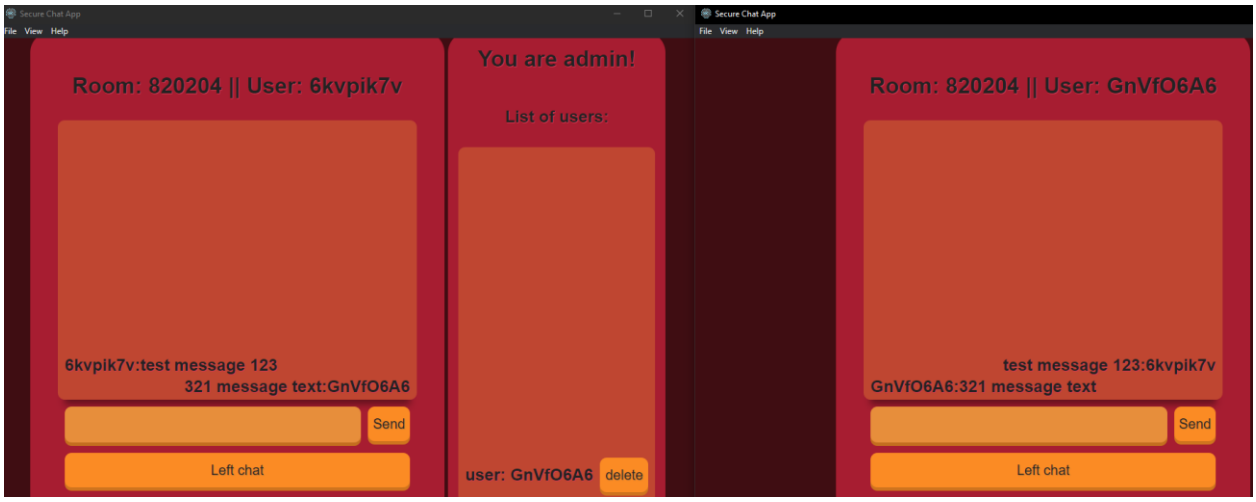

Рисунок 2.7 – Відправлення повідомлень в межах однієї кімнати користувачами один одному

Як можна бачити на рисунку, процес дешифрування повідомлень виконується коректно, а разом із попередньо наведеними даними, можна стверджувати, що наскрізне симетричне AES-шифрування було реалізовано під час розроблення проекту в повній мірі.

2.2 Вирішення проблеми з отриманням доступу до даних користувачів третіми сторонами

В межах цього пункту буде розглянуто не несанкціонований доступ до інформації користувачів, а доступ до такої інформації з боку серверу. Метою впровадження такого захисту є забезпечення конфіденційності повідомлень при можливій компрометації бази даних серверу та забезпечення неможливості обробки даних користувачів самим сервером в інтересах розробників месенджеру та неможливості передачі чи продажу цих даних третім особам для їх подальшої обробки.

Реалізовано в розроблюваному проекті такий захист було наступним чином: враховуючи невеликий функціонал проекту, а саме лише безпечну відправку текстових повідомлень, кількість метаданих, необхідна для функціонування чат-застосунку, є невеликою і складається лише з IP-адреси користувача, що відправляє повідомлення, і IP-адреси користувача, який їх

приймає. Однак, навіть при цьому, IP-адреси користувачів можна перехопити лише на транспортному рівні, тобто, при проходженні пакетів між користувачами і сервером і це можливе, тому що IP-адреса користувача є тим необхідним унікальним ідентифікатором, який використовується для маршрутизації в мережі Інтернет.

Дані IP-адреси – це єдині реальні дані про користувача, які взагалі фігурують в процесі передачі повідомлень. Але напряму вони ні в коді клієнту, ні в коді серверу не фігурують, бо сервер з клієнтом використовують власний ідентифікатор користувачів, такий як userId. За допомогою технології WebSocket до серверу спочатку здійснює підключення користувач, що створює кімнату, після чого – інші користувачі, які до кімнати приєднуються, і, завдяки спільному підключенню до одного серверу WebSocket і збереженні даних про кожну кімнату і її користувачів в базі даних, встановлюється постійне з'єднання між цими користувачами, що дає змогу обмінюватись повідомленнями в межах кімнати.

На рисунку 2.8 зображено дані, які містяться на базі даних за умови існування однієї кімнати чату, до якої підключені 3 користувачі.

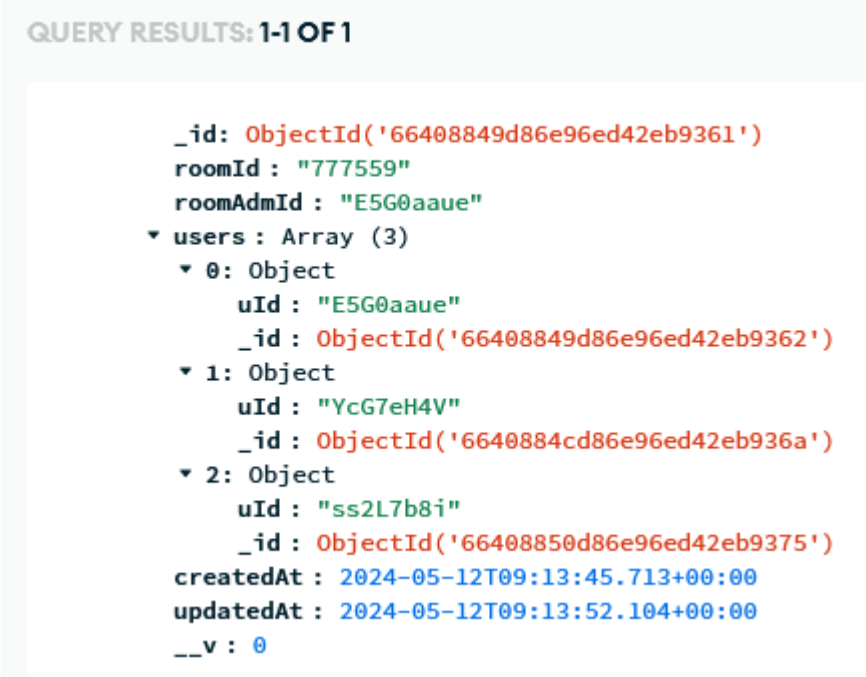

Рисунок 2.8 – Вміст бази даних при підключенні 3 користувачів до кімнати

Запис в базі даних містить лише два типи параметрів: ID кімнати та ID користувачів. При підключенні або відключенні користувачів від кімнати запис в базі даних оновлюється шляхом зміни вмісту масиву об’єктів-користувачів на актуальний для поточного стану кімнати, тобто або додаючи запис з ID користувача, що підключився, до масиву, або видаляючи запис користувача, що відключився. Таким чином, база даних зберігає лише ті ідентифікатори користувачів, яка сама і створює і які не несуть ніякої корисної інформації поза середовищем чат-застосунку, і також оброблює лише дані тих користувачів, які в поточний момент підключені до якоїсь кімнати.

Згідно наведеним даним щодо впровадження безпеки даних користувачів від їх обробки на стороні серверу і санкціонованої передачі третім особам для проведення їх обробки, у створеному чат-застосунку було впроваджено високий рівень захисту цього типу, оскільки сервер не має ніякої інформації про користувачів, які знаходяться в процесі обміну повідомленнями, так само як і до самих повідомлень, натомість коректне функціонування чат-застосунку забезпечується за допомогою обробки внутрішніх даних і токенів, що присвоюються кожному користувачеві у кожній чат-кімнаті.

2.3 Вирішення проблеми несанкціонованого отримання доступу до акаунту користувача

Останньою визначеною проблемою, пов’язаною з конфіденційністю обміну повідомленнями між користувачами лишається проблема «обману системи» шляхом отримання зловмисником доступу до переписок від імені користувача, який в цих переписках приймав участь. Враховуючи специфіку розроблюваного чат-застосунку, це можуть бути проблеми фізичної втрати пристрою, з якого здійснювалась передача або прийняття повідомлень, і подальшою компрометацією вмісту переписки, так само як і приєднання до кімнати чату сторонньої людини під час переписки.

Ця проблема є однією з найважчих для розв’язання в сучасних чат застосунках, бо підвищення рівню безпеки на цьому напрямі тягне за собою зменшення зручності для користувача, і навпаки. Проте, враховуючи специфіку роботи зі створення саме захищеного чат-застосунку, захист буде впроваджуватись на найвищому можливому рівні.

Серед заходів, що покликані мінімізувати небезпеку отримати доступ від чужого імені до переписки під час того, як вона відбувається, приєднавшись до чат-кімнати, є використання випадково генерованого кожного разу токену чату і токену користувача.

При запуску програми, користувачу будуть доступні дві дії: створити кімнату та приєднатись до кімнати за її номером. Ці номери генеруються випадково на стороні клієнта одразу після натискання кнопки створення кімнати. Вони складаються з 6 цифр від 0 до 9 і одразу відправляються на сервер для зберігання, разом із ID користувача-адміністратора, тобто творця кімнати, і масивом користувачів, початково з одним елементом – об’єктом з ID користувача-адміністратора. В подальшому, в процесі приєднання нових користувачів до чату, запис бази даних буде оновлюватись, заповнюючи масив об’єктами з ID нових користувачів.

Замість створення і зберігання даних акаунтів, кожного разу, як користувач запускає застосунок, для нього буде генеруватись випадковий токен-ідентифікатор з восьми символів в межах a-zA-Z0-9. Цей токен буде використовуватись на сервері для ідентифікації користувача, але так само його можна використовувати як додатковий спосіб ідентифікації користувача. Це можливо завдяки тому, що при приєднанні до кімнати користувач, який її створив і, відповідно, має права адміністратора, повинен підтвердити або відхилити запит користувача, що хоче приєднатись, базуючись на цьому токенті, як це зображено на рисунку 2.9.

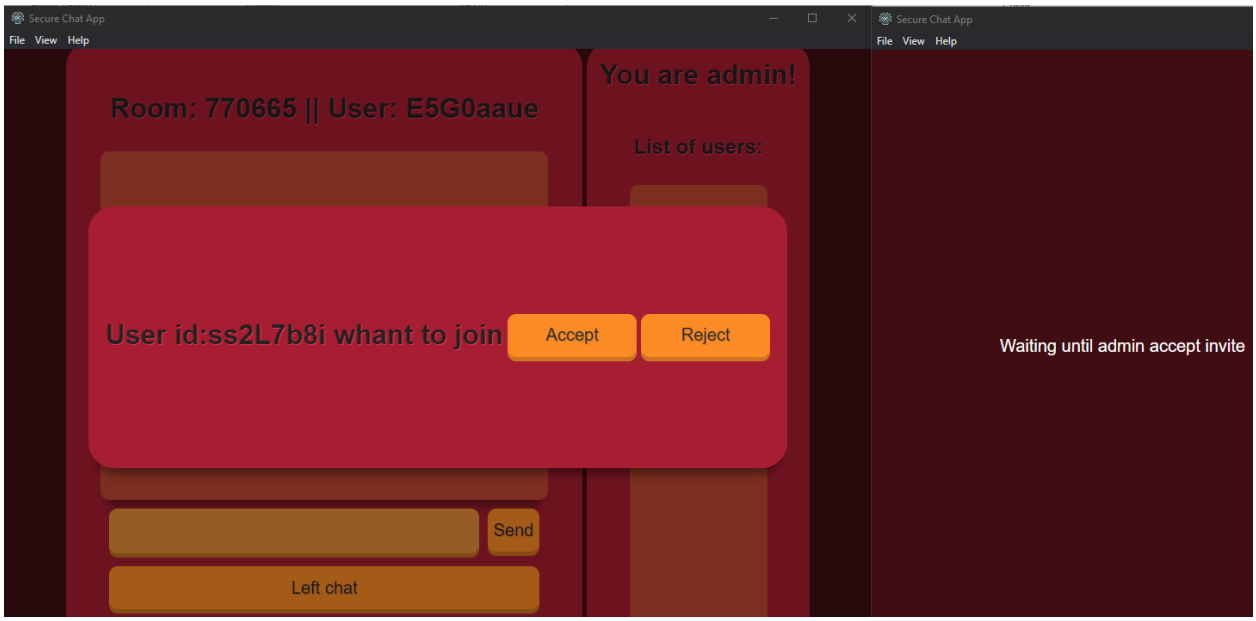

Рисунок 2.9 – Інтерфейс адміністратора кімнати при спробі приєднання користувача (ліворуч) і користувача, що хоче приєднатись (праворуч)

Як зображено на рисунку, допоки творець кімнати не підтвердить запит користувача, що хоче приєднатись, цей користувач не зможе прийняти участь в переписці.

Логіка створення токену кімнати і токену користувача розташовується на стороні клієнту у файлі «../electron/src/render/utils/hash.ts». Генерація токенів зображена на рисунку 2.10: токен кімнати генерується методом класу HashCore.createRoomId(), який не приймає даних і повертає значення типу string, а токен користувача – створенням об’єкту типу інтерфейсу IUser, який визначає поля об’єкту і їх тип з подальшим експортуванням змінної в файли проекту, в яких вона необхідна.

```
44    public static createRoomId(): string {
45      return `${makeid(6, true)}`;
46    }
47  }
48
49  export const user: IUser = {
50    uId: `${makeid(8, false)}`,
51  };
```

Рисунок 2.10 – Генерація токенів чату і користувача

Виклик функції makeid() запускає логіку генерації токенів, і, як зображено на рисунку 2.11, приймає два значення: довжина токену length типу string і прапорець isOnlyNumbers типу boolean, який відповідає за генерацію токену лише з цифр, або включно з нижнім і верхнім регістром англійської розкладки клавіатури. На початку роботи функції створюються дві змінних: змінна result, яка на 10 рядку ініціалізується порожнім рядком і змінна characters, яка створюється на 11 рядку і не ініціалізується одразу. Після цього з 12 по 14 рядок включно відбувається перевірка прапорця isOnlyNumbers, відповідно до якої змінна characters визначається або рядком з цифр з 0 до 9, або рядком з усіма символами a-zA-Z0-9. На 15 і 16 рядках визначаються параметри для роботи алгоритму, а сам алгоритм працює з 17 до 20 рядку включно, роблячи кількість ітерацій, яка дорівнює довжині, переданій як атрибут до функції, на кожній збільшуючи раніше ініціалізовану змінну result на один випадковий символ з визначеної послідовності characters. В кінці функції рядок result повертається як значення результату роботи функції.

```
9   function makeid(length: number, isOnlyNumbers: boolean) {
10    let result = '';
11    let characters;
12    if (isOnlyNumbers) characters = '0123456789';
13    else
14      characters = 'ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz0123456789';
15    const charactersLength = characters.length;
16    let counter = 0;
17    while (counter < length) {
18      result += characters.charAt(Math.floor(Math.random() * charactersLength));
19      counter += 1;
20    }
21    return result;
22  }
```

Рисунок 2.11 – Логічна функція, що відповідає за генерацію токенів

Подібна реалізація дозволяє мінімізувати можливість приєднання до активного чату злочинця, бо токени генеруються випадково кожного разу і адміністратор сам приймає рішення, чи надавати користувачеві доступ до чату, що при обміні інформацією про ID кімнати і ID користувачів у

відкритому доступі робить обмін повідомленнями в чат-застосунку набагато більш захищеним від подібної компрометації.

При приєднанні до кімнати чату, користувач не матиме змоги бачити повідомлення, що були відправлені до його приєднання. Це є як особливістю реалізації чат-застосунку, а саме те, що повідомлення на сервері не зберігаються навіть в зашифрованому вигляді, так і додатковим захистом, якщо адміністратор випадково прийме до чату людину, яка отримала токен чату, але не повинна мати доступу до переписки, як це зображено на рисунку 2.12.

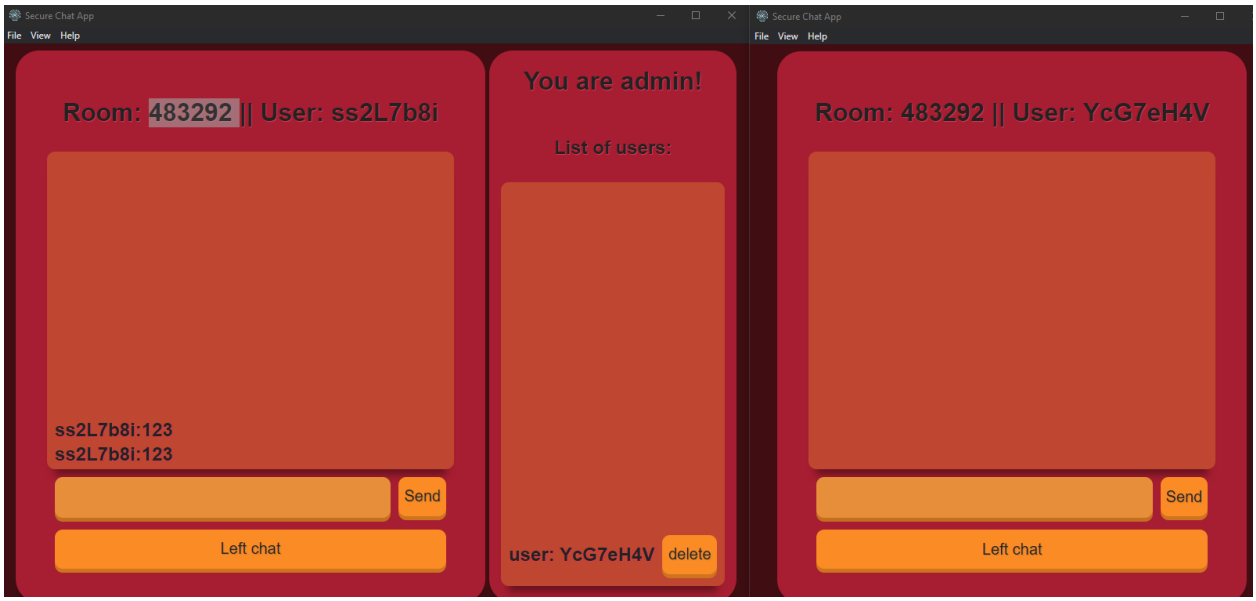

Рисунок 2.12 – Неможливість користувачем, що приєднався, бачити історію переписки

Так само на цьому рисунку можна побачити панель адміністратора (праворуч від чату), це в режимі реального часу відображаються всі користувачі чату, окрім самого адміністратора, і також присутня кнопка видалення користувача. Надання такої можливості покликане наділити адміністратора кімнати засобами реагування, якщо було випадково прийнято зловмисника до чат-кімнати.

Захист від отримання доступу до пристрою, з якого здійснювалась переписка, після її завершення впроваджений за допомогою того, що при

покиданні користувачем кімнати він більше ніколи не зможе отримати доступ до історії переписки, навіть при повторному приєднанні історія чату не відображатиметься, бо вона знаходиться лише на пристроях користувачів, що беруть в ній участь, і видаляється при виході. При цьому користувач буде покидати кімнату не лише за умови натискання кнопки виходу, а і за таких умов, як вихід з кімнати чату адміністратора цієї кімнати, або після спливання 300-секундного періоду, яким визначено час до видалення кімнати за умови відсутності нових повідомлень в ній. Метод, що відповідає за видалення кімнати, розташований на стороні серверу у файлі «../sever/src/socket.ts» і є методом класу SocketConnect.inactiveChatTimeout(), він зображений на рисунку 2.13.

```
142  async inactiveChatTimeout(socket:Socket, roomId:string, uId:string) {
143      const room = await Chat.findOne({ roomId: roomId }).lean();
144      let currentRoom = this.activeUsers.filter(chat => chat.roomId === roomId)[0];
145      const adminId = room?.roomAdmId
146
147      console.log(socket.rooms)
148      if (adminId && adminId === uId) {
149          socket.emit('chat message admin', {uId: adminId});
150      } else {
151          currentRoom.users.map((user) =>{
152              socket.in(user.uId).emit('chat message admin', {uId: adminId});
153          })
154      }
155      console.log("Inactive, delete chat", room?.roomId + '||' + roomId);
156  }
```

Рисунок 2.13 – Код методу inactiveChatTimeout()

Метод приймає атрибутами об’єкт, необхідний для роботи з сокетом, який відповідає за встановлення зв’язку між клієнтами і має назву socket і тип Socket, значення токену кімнати roomId типу string та значення токену користувача, для якого запускається функція, uId типу string. На рядку 143 знаходиться запис в базі даних, який відповідає відправленому токену кімнати, на рядку 144 в змінну currentRoom записується об’єкт з двома полями: номером кімнати, що був переданий, і масивом користувачів, які в ній присутні, далі визначається ID адміністратора знайденої кімнати на рядку 145.

З рядку 148 починається логіка видалення чату, де викликається функція видалення чату, відповідно до того, чи є надісланий ID користувача ID адміністратора кімнати. Цей метод викликається у методі setTimeout(), який і встановлює час його спрацювання. При кожному надходженні нового повідомлення в межах кімнати метод перевиконується, тим самим встановлюючи час до видалення чату на початкові 300 секунд.

Таким чином, сукупність доданих заходів безпеки робить чат-застосунок добре захищеним одночасно і від перехоплення повідомлень, шляхом введення наскрізного шифрування, і від потенційної обробки конфіденційних даних користувачів на сервері, шляхом зберігання на ньому лише токенів, які надаються і використовуються лише в межах чат-застосунку, і від спроб приєднатись до активного чату або отримати доступ до історії переписок, шляхом надання творцям кімнати функції адміністрування і додаванням таймеру видалення чату при неактивності користувачів.

3 РЕАЛІЗАЦІЯ ЧАТ-ЗАСТОСУНКУ. АРХІТЕКТУРА І ІНТЕРФЕЙС

3.1 Файлова структура чат-застосунку

3.1.1 Архітектура клієнтської частини чат-застосунку

За клієнтську сторону чат-застосунку відповідає папка за назвою «electron», яка описує логіку і зовнішній вигляд чат-застосунку на клієнтському боці, тобто за все, до чого матиме доступ користувач.

На рисунку 3.1 зображено вміст папки electron, серед якого присутні як додаткові папки, так і окремі файли.

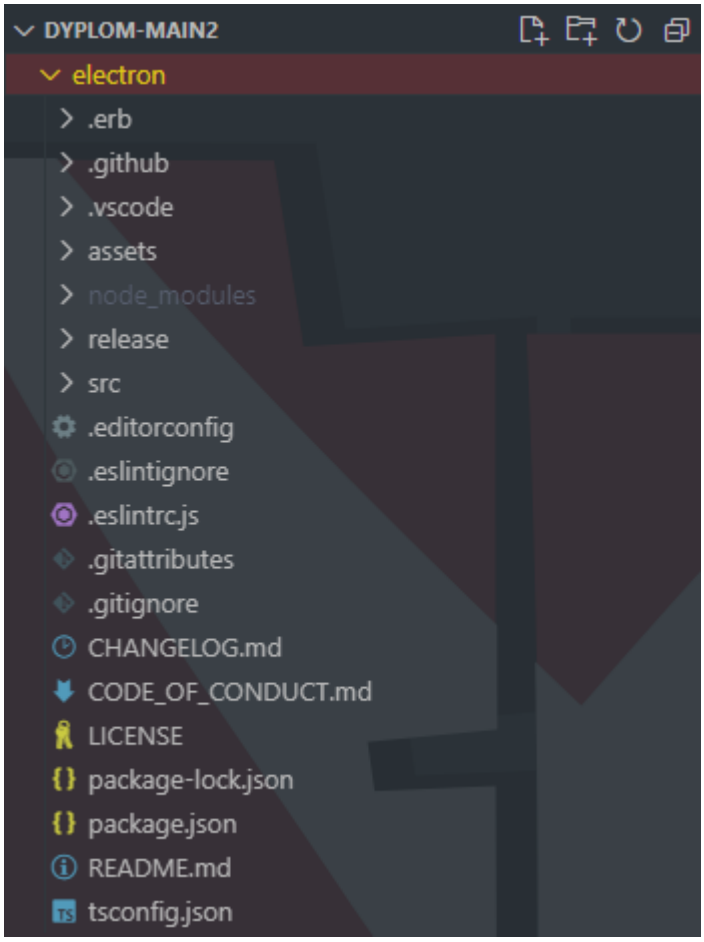

Рисунок 3.1 – Вміст папки «electron», де описується клієнтський бік застосунку

Папки «.erb», «.github», «.vscode» та «node_modules» відповідають за налаштування і конфігурацію інструментів, що були використані при створенні чат-застосунку, так само як і файли «.editorconfig», «.eslintignore»,

«.eslintrc.js», «.gitattributes», «.gitignore», «CHANGELOG.md», «CODE_OF_CONDUCT», «LICENSE», «package-lock.json», «package.json», «README.md» та «tsconfig.json». Ці файли і папки створюються і заповнюються автоматично при підключенні тих чи інших модулів, які використовуються при створенні чат-застосунку.

Папка «assets» зберігає іконки, які використовуються в чат-застосунку, так само як і їх конфігураційні файли. Папка «release» створюється автоматично при побудові проекту, тобто після завершення його написання і результатом її створення є файл, який встановлює клієнтську версію чат-застосунку на пристрій.

Папка «src» зберігає всю логіку клієнтської частини, так само як її зовнішній вигляд. Вона містить дві папки: «main» та «render». Файли папки «main» відповідають за запуск вікна чат-застосунку і його попередні налаштування, налаштування контекстного меню при правому кліку і за безпечну реалізацію роботи з файловою системою пристрою, на який встановлюється чат-застосунок.

Папка «renderer» описує більшу частину логіки, html-структуру і стилізацію. Вона має структуру, зображену на рисунку 3.2.

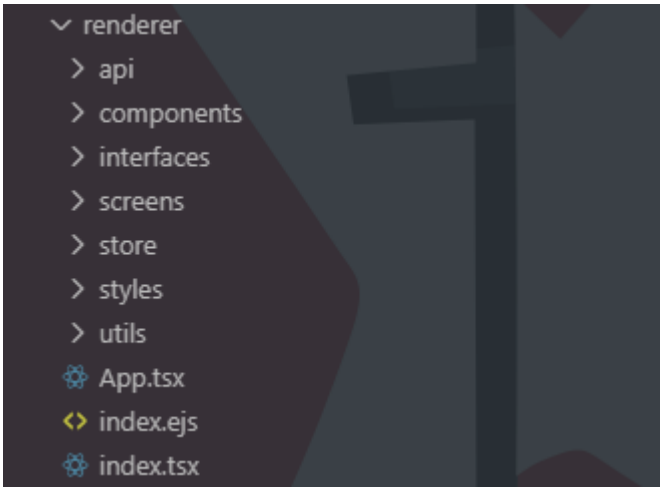

Рисунок 3.2 – Вміст папки «renderer»

Папка «api» містить всі запити з клієнту до серверу на отримання потрібних даних, або на їх відправку. Також файли папки містять логіку, яка виконується при підключенні користувачів до кімнати, або при їх відключенні.

Папка «components» містить файл, який відповідає за налаштування модального, тобто спливаючого, вікна, яке бачить адміністратор чат-кімнати, коли отримує запит на приєднання, і користувач, який цей запит робить, поки він не буде прийнятий або відхилений.

Папка «interfaces» утримує інтерфейси об’єктів, які часто створюються, таких як об’єкт чату, екземпляри якого зберігаються у базі даних, об’єкт користувача чат-застосунку, об’єкт, який використовується при відправці повідомлень на сервер та об’єкт кімнати чату.

У папці «screens» реалізується html-структура і логіка окремих елементів, таких як кнопки, для двох екранів: для стартового, коли застосунок тільки запускається, і для екрану кімнати, коли користувач створює чат або приєднується до вже існуючого.

Папка «store» зберігає створення сховища для важливих даних для більш швидкого і зручного доступу до них.

Папка «styles» відповідає за опис стилів для всіх компонентів інтерфейсу. Вона містить файли типу .css, які налаштовують низку параметрів, таких як розмір html-елементів, колір тексту, анімація натискання кнопок, тощо.

У файлах папки «utils» описується логіка створення токенів користувачів і чатів, а також здійснюється шифрування і розшифрування повідомлень, які відправляються на сервер і приходять з нього, відповідно.

Файл «App.tsx» описує шляхи, за якими будуть вивантажуватись сторінки застосунку, наприклад, при завантаження застосунку у початковому вікні буде вивантажуватись початкова сторінка, яка описана в папці «screens».

Файл «index.ejs» містить html-структуру з елементом з ID «root», який є точкою, в яку будуть вивантажуватись екрани з папки «screens».

Файл «index.tsx» є точкою входу в фреймворк React, де вказується елемент з ID «root», який був створений в файлі «index.ejs».

3.1.2 Архітектура серверної частини чат-застосунку

Серверна частина чат-застосунку представлена папкою «server», яка відповідає за логіку серверу, куди входить прийняття і обробка даних від застосунків на клієнтах, організація збереження необхідних даних і переправлення даних між користувачами. Вміст цієї папки наведено на рисунку 3.3.

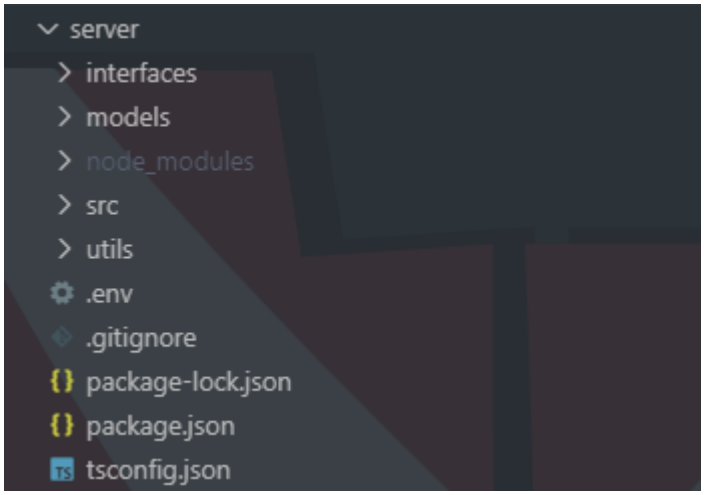

Рисунок 3.3 – Вміст папки «server», в якій описано серверну частину чат-застосунку

В папці «interfaces» містяться інтерфейси об’єктів для швидшого і простішого створення екземплярів таких об’єктів. В цій папці наведені інтерфейси таких об’єктів, як об’єкт чату, екземпляри якого зберігаються у базі даних, об’єкт користувача чат-застосунку та об’єкт, в форматі якого пересилаються повідомлення користувачів мережею.

Папка «model» містить файл, який описує схему, за якою зберігаються дані в базі даних і створює її модель для подальшого експорту для організації даних у визначеному для їх зберігання в БД форматі.

Папка «node_modules» і файли «.gitignore», «package-lock.json», «package.json» та «tsconfig.json» відповідають за налаштування і конфігурацію підключених і встановлених інструментів і бібліотек.

Папка «src» містить два файли: «main.ts» та «socket.ts». Ці файли відповідають за логіку серверу. Файл «main.ts» описує логіку запуску серверу, підключення до бази даних і за роботу кінцевих точок (endpoints), які, в свою чергу, відповідають за обмін даними між сервером і клієнтом. В файлі «socket.ts» описується логіка підключення клієнта до сервера, пересилання повідомлень між учасниками чату, збереження та видалення даних з бази даних, відключення клієнта і видалення чату після спливання таймеру неактивності.

В папці «utils» описується функція підключення до бази даних, яка експортується з файлу і імпортується в файлі «main.ts».

Файл «.env» зберігає змінні оточення, встановлюючи їх область видимості на всі файли серверу. Серед цих змінних присутні такі, як порт серверу, адреса і порт, на яких запускається клієнтський застосунок і посилання на базу даних.

3.2 Огляд інтерфейсу чат-застосунку

Після встановлення чат-застосунку на пристрій, він буде готовий до запуску без жодних додаткових налаштувань, за умови того, що сервер запущено або на локальному хості, тоді чат буде працювати лише в межах локальної IP-адреси 127.0.0.1, або за допомогою віддаленого хостингу, в такому випадку чат-застосунок буде працювати в межах мережі Інтернет.

Після запуску застосунку буде відкрито вікно з початковою сторінкою, яка наведена на рисунку 3.4.

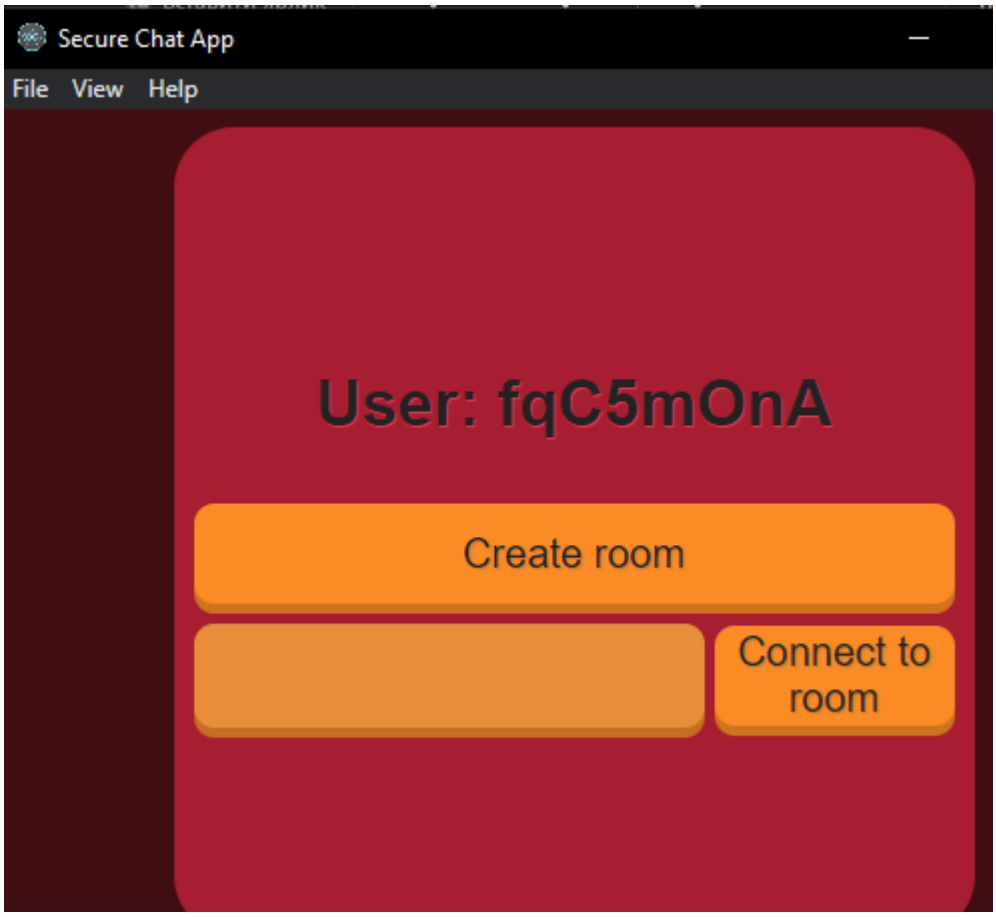

Рисунок 3.4 – початкова сторінка після запуску чат-застосунку

Нагорі відображається випадково згенерований токен користувача, який генерується на клієнтському боці і створюється заново після кожного перезапуску програми. Під токеном користувача знаходиться кнопка «Create room», яка створює нову кімнату чат-застосунку зі своїм унікальним токеном, за яким зможуть підключатись інші користувачі. Внизу знаходиться зв’язані поле для введення даних і кнопка «Connect to room», які відповідають за введення токену кімнати і подальше підключення до неї. На цьому етапі жодних записів в базі даних не створюється, бо процес обміну повідомленнями ще не було розпочато.

Після натискання кнопки «Create room» одразу створюється нова кімната чат-застосунку, запис, який містить її токен, токен адміністратора і масив учасників (на момент створення кімнати – лише один запис з токеном творця), відправляється на сервер, після чого зберігається у базі даних. Коли інший користувач вводить токен цього чату, який було передано по відкритих

каналах, і натискає кнопку «Connect to room», він бачить вікно з очікуванням підтвердження приєднання від творця кімнати, в цей же час творець кімнати бачить запит і має або його прийняти, або відхилити, як зображено на рисунку 3.5.

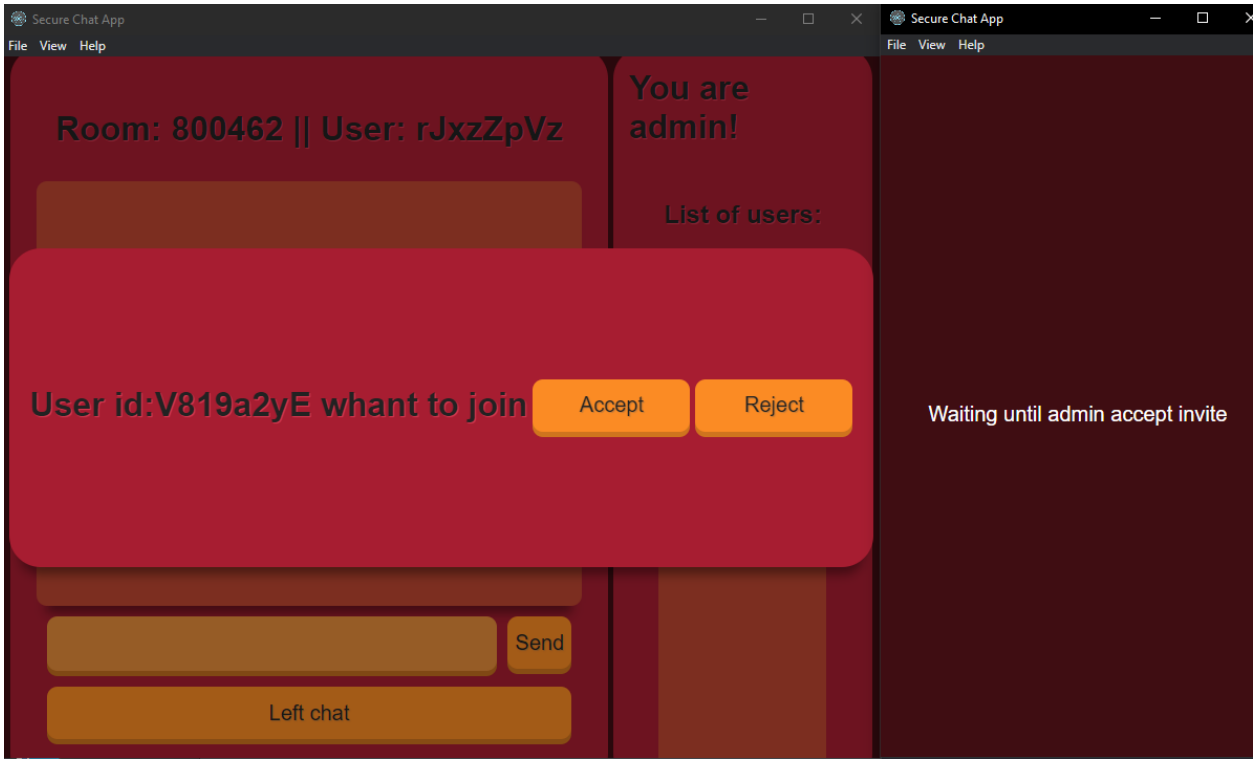

Рисунок 3.5 – Інтерфейс адміністратора кімнати при спробі приєднання користувача (ліворуч) і користувача, що хоче приєднатись (праворуч)

При натисканні адміністратором кнопки «Reject» відбувається відхилення запиту на приєднання, користувач, що намагався підключитись, буде повернений на початковий екран. Якщо була натиснута кнопка «Accept», запит приймається і на сервер відправляються дані нового користувача для оновлення запису в базі даних, а саме для розширення масиву користувачів шляхом додавання в нього нового об’єкту з полем ID-користувача, тобто його токену. Прийнятий користувач, в свою чергу, отримує доступ до кімнати чату, де може читати і відправляти повідомлення.

На рисунку 3.6 зображено вигляд вікна чату після його створення, підключення до нього другого користувача і після відправки повідомлення кожним з користувачів.

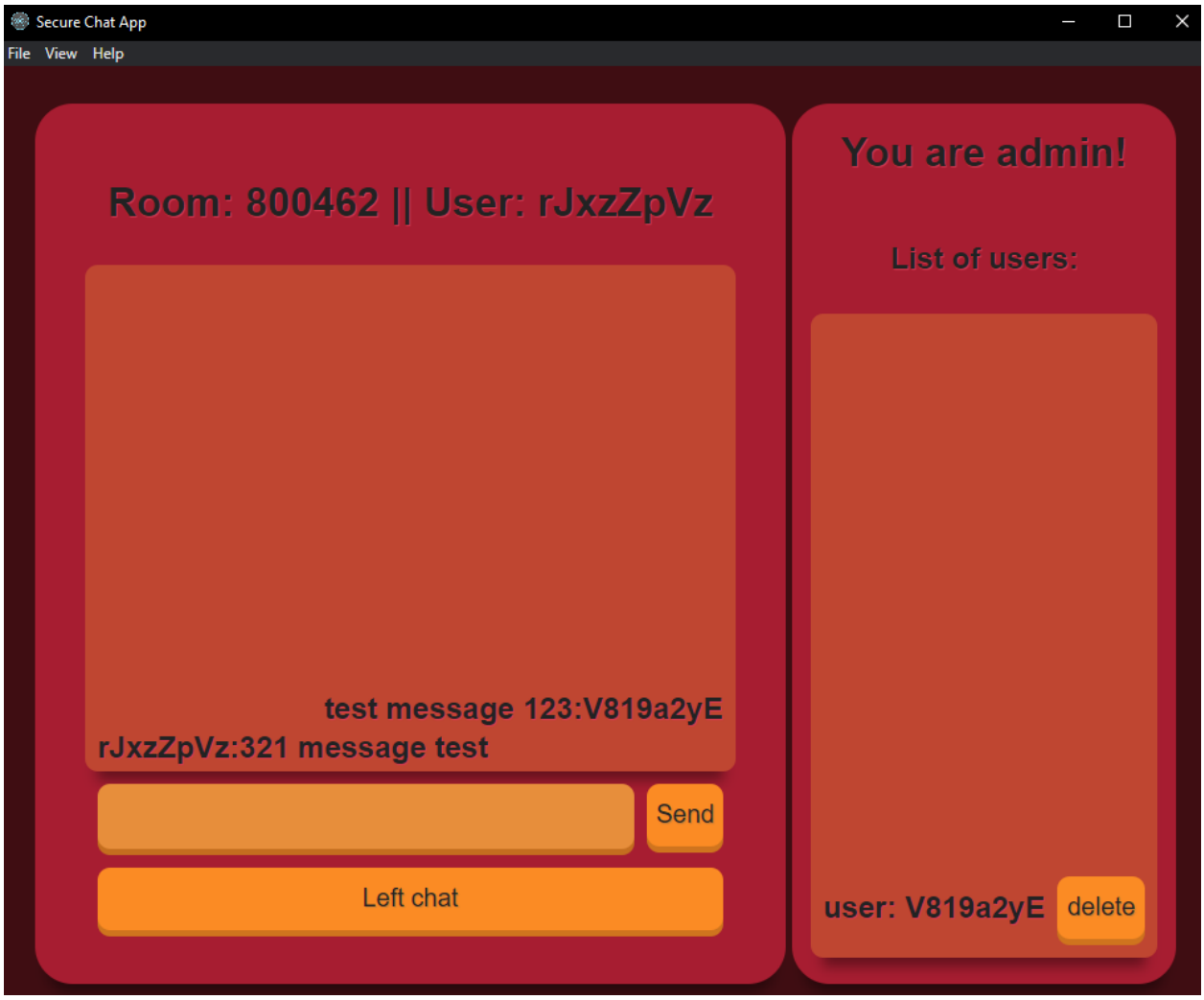

Рисунок 3.6 – Зовнішній вигляд вікна адміністратора чату під час здійснення переписки

В лівій верхній частині рисунку можна побачити токен кімнати і токен користувача. Нижче зображена область повідомлень, де наведено вигляд повідомлень, які приходять користувачу, і які від нього відправляються. Ліворуч від кнопки «Send» знаходиться поле для введення повідомлення, ця кнопка відповідає за його відправку на сервер і далі до кожного з учасників кімнати. За допомогою кнопки «Left chat» користувач може покинути чат, тоді з запису про кімнату чату у базі даних видаляється його ID, а користувач

повертається на початкову сторінку. Це є повним функціоналом для користувачів, які приєднались до раніше створеного чату.

На рисунку 3.6 зображено саме інтерфейс адміністратора, відмінність якого полягає в тому, що при натисканні адміністратором кнопки «Left chat» чат буде видалено, а всі його учасники будуть відправлені на початкову сторінку. Також у правій частині рисунку присутня панель адміністратора, яка дає змогу бачити всіх активних користувачів чату і, за потребою, вилучати їх, після чого вилучений користувач так само буде направлений на початкову сторінку.

4 КОНКУРЕНТНІСТЬ І ПОКРАЩЕННЯ ЧАТ-ЗАСТОСУНКУ

4.1 Конкурентність розробленого чат-застосунку

Чат-застосунок відповідає тим вимогам, за якими розроблювався, а саме: впровадження надійного і безпечного для користувача наскрізного шифрування, забезпечення конфіденційності даних користувачів і чатів при збереженні і мінімізація ризиків компрометації даних при отриманні зловмисником фізичного доступу до пристрою, з якого здійснювалась переписка.

Розроблений застосунок не може конкурувати з сучасними месенджерами за критеріями зручності для користувача і підтримки різних версій і різних операційних систем. Проте, за рахунок відкидання багатьох рішень, пов’язаних зі зручністю, застосунок надає доволі високий рівень безпеки, що, навіть, може конкурувати і з сучасними месенджерами.

Шифрування є найслабшим місцем в захисті застосунку, його покращення і альтернативи будуть розглянуті в підрозділі 4.2. AES має великі ключі, надійний алгоритм шифрування, проте воно є симетричним, звідки виходить, що ключі мають або передаватись мережею, від чого шифрування перестає бути наскрізним, або бути константними і створюватись на кожному клієнті однаковими, саме цей варіант і було обрано. Проте ключі модифікувались за допомогою токену чат кімнати, що зменшувало ризик компрометації безпеки всього чат-застосунку при отриманні зловмисниками базового ключа шифрування. Відповідно, в сучасних месенджерах існують більш безпечні і надійні методи обміну ключами, такі як алгоритм Діффі-Геллмана, через що розроблений чат-застосунок не надає максимальний можливий рівень захисту при шифруванні даних.

Однією з вагомих переваг розробленого застосунку є те, що він не зберігає жодної інформації про користувачів, окрім тієї, яку сам їм надає випадковою генерацією: база даних зберігає лише інформацію про токени кімнат і поточних користувачів цих кімнат, відповідно, застосунок не

проводить роботу з реальними даними користувачів взагалі, на відміну від сучасних месенджерів, які зберігають і оброблюють таку інформацію, як операційна система на пристрої, її версія, тощо. Іноді і такі конфіденційні дані, як номер телефону, геолокація або електронна пошта. Всі ці дані можуть оброблюватись серверами месенджерів і, при недостатньому їх захисті, можуть бути отримані зловмисниками.

Також скласти конкуренцію сучасним месенджерам в сфері безпеки складає захист від отримання зловмисником фізичного доступу до пристрою, з якого здійснювалась переписка. Більшість месенджерів пропонує реагувати на подібні події після того, як вони відбулись, бо впровадження заходів безпеки до цього сильно вдаряє по зручності цих месенджерів. Найкращі з методів захисту щодо цієї проблеми наразі – це встановлення паролю на застосунок месенджеру і встановлення таймеру на видалення потрібних повідомлень, проте ці функції не є увімкненими за замовчуванням, що змушує проводити додаткові налаштування перед обміном конфіденційною інформацією. Натомість, розроблений чат-застосунок за замовчуванням впроваджує видалення кімнати чату і для користувачів, і з серверу, якщо нових повідомлень не було протягом 5 хвилин, або якщо адміністратор кімнати покинув чат.

У підсумку, розроблений чат-застосунок може конкурувати з сучасними месенджерами, але лише в певних аспектах, пов’язаних з безпекою даних користувачів і їх повідомлень.

4.2 Можливі покращення розробленого чат-застосунку

Одне з основних покращень чат-застосунку, яке можна і необхідно додати в майбутньому – це перехід на асиметричний протокол обміну ключами, а саме – протокол Діффі-Геллмана. Реалізація його алгоритму дозволяє двом або більше користувачам отримати спільний секретний ключ шляхом обміну відкритими ключами і їх модифікації секретними ключами на своїх пристроях. Тобто, секретний ключ повинен вираховуватись для кожної

кімнати, різний для різної кількості поточних користувачів, шляхом використання пари ключів, секретного і публічного, кожного з учасників чату. Цей алгоритм і його модифікації є одними з найнадійніших методів обміну ключами шифрування на день написання роботи і використовуються багатьма сучасними месенджерами.

Друге з дуже важливих покращень – це перехід на використання https замість http. Хоча зараз і використовується шифрування повідомлень, при перехопленні пакетів, що йдуть мережею, зловмисник може отримати доступ до нехай і лише внутрішніх, але даних користувачів: токену кімнати, токену користувача і шифрованому повідомленню. Тому впровадження https-з'єднання необхідно для додаткового захисту інформації при пересиланні даних між клієнтом і сервером.

В доповнення, яке може зробити чат-застосунок набагато кориснішим, треба додати можливість відправляти файли різних типів, при цьому запровадити для них той самий рівень захисту, що і для текстових повідомлень.

Опціональні покращення зручності використання чат-застосунку включають додання можливості для користувачів вказувати псевдонім при спробі підключення до чату, або при запуску застосунку. Це може спростити і пришвидшити розпізнавання адміністратором кімнати користувачів, що можуть приєднатись, але при цьому токени користувачів повинні залишитись, оскільки вони створюються випадково і є унікальними, що унеможливлює їх компрометацію, на відміну від псевдонімів користувачів, які можуть бути заздалегідь визначені зловмисником, після чого можуть бути використані при спробі підключення до кімнати чату.

Також для покращення зручності користування чат-застосунком може бути додано передачу прав адміністратора іншому користувачеві, так само як і налаштування таймеру видалення чату і можливість видалення повідомлень користувачем чи адміністратором.

ВИСНОВКИ

На підставі проведених досліджень було визначено недоліки в реалізації безпекових рішень в сучасних месенджерах. Цими недоліками було визначено такі, як неповсюдне використання наскрізного типу шифрування, що тягне за собою можливість перехоплення ключів, які використовуються для шифрування і дешифрування повідомлень і можливість отримати доступ до цих ключів при компрометації вмісту бази даних серверу. Цього недоліку можна позбавитись шляхом впровадження такого алгоритму шифрування, де саме шифрування і розшифрування повідомлень проводиться на боці клієнтських застосунків, при цьому ключі цього алгоритму не повинні передаватись мережею, а створюватись і існувати виключно на пристроях користувачів. На практиці цю проблему було вирішено шляхом впровадження симетричного шифрування AES. Ключі шифрування генерувались на клієнтських застосунках зі значення базового ключа, який є набором випадкових 128 символів, який модифікувався значенням токену кімнати.

Другим недоліком було визначено можливість обробки і передачі персональних даних користувача сервером, наприклад, такої інформації, як номер телефону або електронна пошта, за умови, що при передачі повідомлень впроваджено наскрізне шифрування. Цю проблему можна вирішити шляхом збирання мінімальної необхідної кількості інформації про користувача для функціонування чат-застосунку. В реалізованому чат-застосунку цю проблему було вирішено шляхом створення двох внутрішніх ідентифікаторів: токену користувача і токену чат-кімнати. Сервер зберігав і оброблював лише ці дані, які навіть при їх перехопленні не дали б зловмиснику жодної корисної інформації.

Третій недолік сучасних месенджерів було визначено тим, що вони мають недостатній захист даних в випадку отримання зловмисником пристрою, з якого проводилась переписка. Для мінімізації ризиків подібної загрози, при реалізації чат-застосунку було додано підтвердження приєднання

до чату адміністратором кімнати, видалення всієї історії переписки при покиданні користувачем чату і автоматичне від’єднання користувачів і видалення всіх повідомлень чату при відключенні від кімнати адміністратора або при відсутності нових повідомлень протягом 5 хвилин.

Таким чином, визначені проблеми були вирішені за допомогою реалізації власної версії захищеного чат-застосунку, але при цьому застосунок має можливості для покращення, такі як перехід на більш надійних асиметричний метод обміну ключами, перехід на https-з’єднання, додання можливості відправляти файли різних розширень за допомогою чату і інші покращення, які не мають суттєвого впливу на безпеку застосунку, проте підвищують зручність користування ним.

Наведені можливі покращення можуть бути реалізовані при подальшому покращенні чат-застосунку для підвищення його актуальності і конкурентоздатності відносно сучасних месенджерів в сфері безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. У пошуках безпечного месенджера. *KR. Laboratories*: режим доступу: https://kr-labs.com.ua/blog/top-secure-and-privacy-messaging-apps/ (дата звернення: 22.04.2024).

2. What makes a messaging app secure? *Kaspersky official blog*: режим доступу: https://www.kaspersky.com/blog/what-makes-a-messenger-secure/48671/ (дата звернення: 22.04.2024).

3. What is end-to-end encryption (E2EE)? *Cloudflare*: режим доступу: https://www.cloudflare.com/learning/privacy/what-is-end-to-end-encryption/ (дата звернення: 23.04.2024).

4. Наскрізне шифрування. *Wikipedia*: режим доступу: https://uk.wikipedia.org/wiki/Наскрізне_шифрування (дата звернення 23.04.2024).

5. What Is AES Encryption and How Does It Work. *Simplilearn*: режим доступу: https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption (дата звернення: 11.05.2024).

6. Advanced Encryption Standard. *Wikipedia*: режим доступу: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (дата звернення: 11.05.2024).

7. Advanced Encryption Standard (AES). *GeeksforGeeks*: режим доступу: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ (дата звернення: 11.05.2024).

8. CryptoJS Documentation. *CryptoJS*: режим доступу: https://cryptojs.gitbook.io/docs (дата звернення: 13.05.2024).