МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

Дипломна робота

РОЗРОБКА КРИПТОГРАФІЧНОЇ СИСТЕМИ МЕСЕНДЖЕР

Виконав: студент групи ПМК-41с
спеціальності
125 «Кібербезпеки»
(шифр і назва спеціальності)

(підпис) Васенда А.С.
(прізвище та ініціали)

Керівник (підпис) доц. Трушевський В. М.
(прізвище та ініціали)

Рецензент (підпис) Коковська Я. В.
(прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Кібербезпеки
Спеціальність 125 «Кібербезпека»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
"31 "серпня 2022 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Васенді Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1 .Тема роботи Розробка криптографічної системи месенджер

керівник роботи Трушевський Валерій Миколайович, доцент кафедри кібербезпеки
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від " 13" вересня 2022 року № 15

2. Строк подання студентом роботи 13.06.2023р.

3. Вихідні дані до роботи: аналіз та дослідження криптографічної системи месенджер, вибір технологій для реалізації, реалізація взаємодії між клієнтом та сервером

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

1. Теоретичні основи розробки криптографічної системи месенджер.
2. Дослідження роботи застосунку.
3. Створення бота.

5. Перелік графічного матеріалу:

1. Презентація доповіді, виконана в Microsoft PowerPoint.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 31 серпня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів дипломної роботи	Термін виконання	Примітки
1	Уточнення постановки завдання	10.09.2022	
2	Аналіз літератури	28.09.2022	
3	Обґрунтування вибору рішень	08.10.2022	
4	Збір даних	27.11.2022	
5	Теоретичні відомості	06.01.2023	
6	Дослідження розробки криптографічної системи месенджер	08.02.2023	
7	Практичне застосування	21.03.2023	
9	Оформлення презентацій	17.04.2023	
10	Отримання рецензій	10.05.2023	
11	Подання роботи на кафедру	12.06.2023	
12	Захист в ЕК	15.06.2023	

Студент _____ (підпис) Васенда А. С. (ініціали, прізвище)

Керівник роботи _____ (підпис) доц. Трушевський В. М. (ініціали, прізвище)

РЕФЕРАТ

Пояснювальна записка дипломного проекту складається із вступу, трьох розділів, що містять 14 рисунків, висновку та списку використаних джерел з 5 найменувань. Загальний обсяг роботи становить 29 сторінок.

Об'єкт дослідження. Дослідження охоплює криптографічні протоколи, приватність та безпеку, системну архітектуру, взаємодію з користувачем, ефективність та оптимізацію, а також валідацію та тестування системи месенджера.

Метою роботи є розробка криптографічної системи месенджера, спрямованої на забезпечення конфіденційності, цілісності та доступності даних користувачів. Робота включає створення ефективних криптографічних протоколів, захист від атак, оптимізацію системи та валідацію її безпеки.

Галузь застосування охоплює сферу комунікацій та обміну інформацією. Це може бути використано в особистих розмовах, бізнес-комунікації, медіа-передачі, урядових комунікаціях та будь-якому контексті, де приватність, безпека та конфіденційність даних є важливими.

Ключові слова: Socket.io, Node.js, Expressjs, Vue, Node-RSA.

ABSTRACT

The explanatory note of the diploma project consists of an introduction, three sections containing 14 figures, a conclusion and a list of used sources from 5 names. The total volume of work is 29 pages.

Object of study. Research covers cryptographic protocols, privacy and security, system architecture, user interaction, efficiency and optimization, and messenger system validation and testing.

The method of work is the development of a cryptographic messenger system aimed at ensuring the confidentiality, integrity and availability of user data. The work includes creating effective cryptographic protocols, protecting against attacks, optimizing the system and testing its security.

The field of application covers the field of communications and information exchange. It can be used in personal conversations, business communications, media transmissions, government communications, and any context where privacy, security, and data confidentiality are important.

Keywords: Socket.io, Node.js, Expressjs, Vue, Node-RSA.

ЗМІСТ

ВСТУП

Актуальність теми.

У наші дні сучасне суспільство неможливо уявити без засобів зв'язку. Більшість часу люди проводять, використовуючи свої гаджети. Те, що вони роблять, це читають новини, дивляться якісь цікаві відеоролики або ж смішні картинки. Також люди спілкуються між собою за допомогою соц-мереж та месенджерів. Месенджер - програмне забезпечення, за допомогою якого два користувача можуть обмінюватися текстовими повідомленнями або будь-якою іншою інформацією в реальному часі. Безперечно, величезною перевагою месенджерів є зберігання повідомлень, і будь-якої миті можна знайти необхідну інформацію. Це також призводить до неактуальності телефонних дзвінків, і є ймовірність, що, у найближчому майбутньому їх замінять месенджери. Актуальність роботи полягає у шифруванні, при роботі якого використовуються криптографічні ключі. У зв'язку з цим ніхто з поза не може отримати інформацію, що передається на сервер або до клієнта. На сьогодення зростає тенденція на збереження особистої інформації, тому при розробці будь-якої інформаційної системи прагнуть зробити її більш-захищеною, та використовують все більш надійні алгоритми шифрування.

Мета роботи.

Мета даної роботи полягає у створенні шифрованого онлайн-чату, для обміну повідомлень без страху, що ці повідомлення будуть викриті третіми особами, або будуть збережені і використовуватись проти наших прав. Розробка власної інформаційної системи, яка буде виконувати основні функції необхідні для обміну особистими зашифрованими повідомленнями.

РОЗДІЛ 1. РОЗБІР АНАЛОГІВ

1.1 Telegram

Telegram – це кросплатформна програма для миттєвої передачі повідомлень. Перша версія вийшла у серпні 2013 року. Програма написана на C++. Для нього створили протокол MTProto, який використовує декілька протоколів шифрування. При авторизації користувача
застосовуються алгоритми RSA-2048, DH-2048 для шифрування. Також використовуються криптографічні хеш-алгоритми SHA-1 та MD5.

У Telegram присутня не тільки передача повідомлень, а й дзвінки, у тому числі й шифровані, боти, канали.

- Бот – допоміжна програма, яка автоматично або по заданим розкладом виконує якісь дії через користувальницький інтерфейс.
- Канал – розрахований на багато користувачів анонімний інструмент комунікації користувачів, в якому один або кілька людей можуть ділитися будь-якою інформацією.

Особливість застосування в тому, що при авторизації на новому пристрої не потрібно завантажувати бекап-файли, все відбувається автоматично, вся історія завантажується сама по собі, а файли зберігаються на сторонньому сервері та доступні будь-якої миті.
Також Telegram підходить для корпоративних цілей. Робочий колектив може користуватися чатом, в якому може бути одночасно до 50 000 користувачів.
Зв'язуватися з іншим користувачем можна не тільки знаючи його номер телефону, але й коротке ім'я, яке він вказав у налаштуваннях.
Ще одним плюсом цієї програми є безкоштовні стікери, які можуть бути створені користувачами самостійно.

1.2 Viber

Viber є кросплатформовою програмою, що дозволяє здійснювати дзвінки та обмін повідомленнями через Wi-Fi та мобільну мережу. Також існує платна функція ViberOut, яка допомагає зв'язатися з пристроєм без доступу в інтернет або ж, на якому не встановлено Viber. Програма має шифрування, секретні листування, дзвінки, відеодзвінки. Є безліч стікерів платних та безкоштовних.

Viber можна вважати корпоративним месенджером у зв'язку з тим, що великі мережі та магазини, які мають ваші дані в базі, можуть надсилати вам рекламу про

які-небудь свої акції. З одного боку, це дуже зручно та практично, але з іншого – набридливо і багатьом не потрібно. На щастя, це легко вимикається.
У додатку також доступні розраховані на багато користувачів чати, боти і громадські канали.

Однак є і недолік: реклама в додатку дуже набридлива і нав'язлива. Додаток збирає ваші дані активності та підбирає рекламні афіші спеціально для вас.
Зв'язатися з кимось із користувачів можна за номером мобільного телефону, які програма синхронізує з телефонної книги. Також тут є можливість малювати та відправляти створені малюнки як картинки, однак цей функціонал непомітний користувачеві та багато хто про нього не знає.

1.3 WhatsApp

WhatsApp - популярний месенджер миттєвого обміну текстовими повідомленнями для мобільних та інших платформ з підтримкою голосового та відеозв'язку. Програма була розроблена мовою Erlang. Перша версія була випущена у 2009 році. Є наймасовішим месенджером. Спочатку додаток був платним, але згодом став безкоштовним. WhatsApp використовує модифікований протокол Presence Protocol (XMPP). При встановленні створюється обліковий запис на сервері s.whatsapp.net, який використовує номер телефону як ім'я користувача.

Версія під Android автоматично використовує як пароль MD-5-хеш від зміненого ідентифікатора IMEI а версія під IOS використовує MO5-хеш від MAC-адреси. Програма автоматично синхронізує список контактів з телефонної книги пристрою. Це можливо завдяки тому, що всі користувачі реєструються за допомогою номера телефону. У месенджері є можливість додавання так званого «статусу», в якому може бути текст, фотографія або невелике відео, яке автоматично видаляється через 24 години. Вперше такий функціонал з'явився у Instagram.

У версіях для персонального комп'ютера є особливість, яка не дозволяє використовувати систему, якщо на мобільному пристрої немає зв'язку з Інтернетом. Також не можна бути активним одночасно у веб версії та версії для персонального комп'ютера.

РОЗДІЛ 2. ПІДГОТОВКА РЕАЛІЗАЦІЇ

2.1 Постановка задачі

- Провести аналіз переваг та недоліків захисту інформації існуючих месенджерів;
- Розробити систему обміну повідомлень яка передбачає застосування гібрідного підходу: асиметричної системи для встановлення спільного секрету двох учасників мережі та симетричної криптосистеми для шифрування повідомлень.

2.1 Вибір технологій

Для створення чату використані технології:

- Node.js – програмна платформа на двигуні V8 яка транслює JavaScript-код в байт-код, через що JavaScript з вузько направленої мови програмування перетворюється в мову загального призначення.
- Express – фреймворк для Node.js, за допомогою якого створюється серверна частина веб-застосунку, яка обробляє запити клієнтів.
- Vue.js – фреймворк для JavaScript, який відповідає за створення клієнтського графічного інтерфейсу, взаємодії з ним, та логіку застосунку.
- Socket.io – бібліотека для JavaScript, яка є обгорткою над WebSocket та реалізує обмін інформацією між сервером та клієнтами за допомогою технологій WebSocket, long-pooling (якщо з'єднання хороше, то будуть використовуватись сокети, якщо з’єднання з перебоями – long-pooling).
- Node-RSA – бібліотека для JavaScript, яка реалізує функції генерування публічного та приватного ключів та шифрування та дешифрування інформації.

2.2 Архітектура системи

В наш час у додатках використовують клієнт-серверну архітектуру. Ця архітектура передбачає два рівня – клієнт і сервер. Клієнт та сервер можна вважати різними застосунками, оскільки вони можуть бути написані на різних мовах програмування та використовувати різні технології. Як-правило кожен із них розташовується на різних комп'ютерах і взаємодіють між собою за допомогою мережевих протоколів, в нашому випадку - WebSockets. Оскільки серверна частина виконує та обробляє запити від різних клієнтських застосунків,

її розміщують на спеціальному виділеному комп'ютері (сервері) з білою статичною IP-адресою.

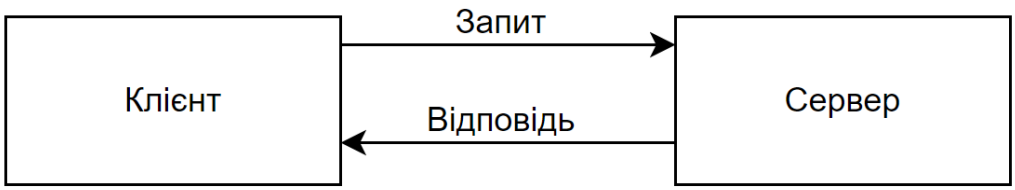

Рис. 2.1.1 Дворівнева схема архітектури клієнт-сервер

Оскільки ми додатково використовуємо бібліотеку Socket.IO, яка представляє собою додатковий сервер WebSocket, то нашу схему можна поділити на багаторівневу.

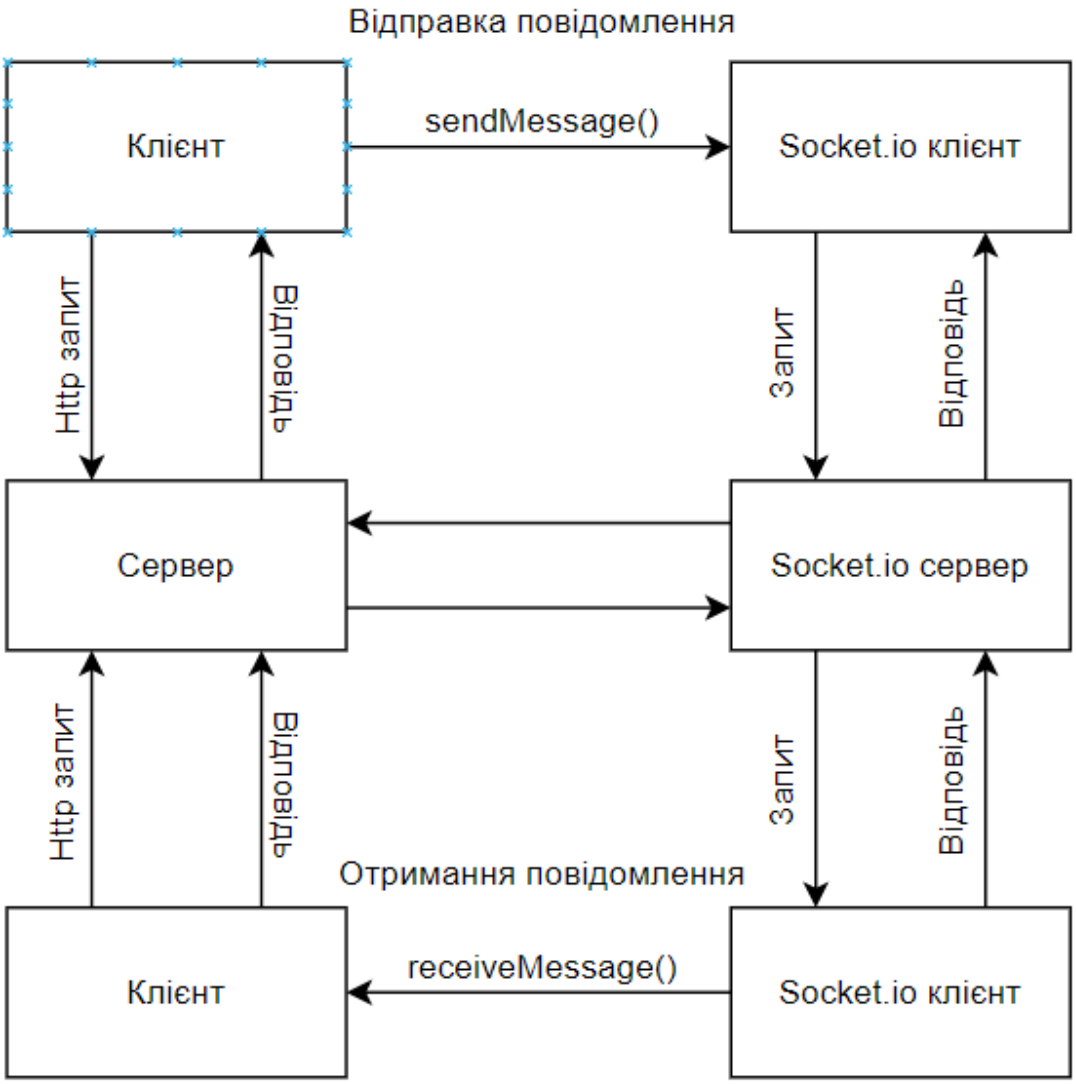

Рис. 2.1.2 Багаторівнева схема архітектури клієнт-сервер

Для початку клієнт має відправити серверу запит на отримання клієнтського застосунку. Клієнт реагує на запит і відправляє йому клієнтський застосунок після чого клієнт може відправляти повідомлення через Socket.io на сервер, який в свою чергу буде перешле потрібному клієнту це повідомлення.

Коли клієнт отримує клієнтський застосунок, потрібно ввести логін, після чого створюються публічний та приватний ключі і відправка та отримання повідомлень відбуваються в зашифрованому варіанті:

- Користувач вводить повідомлення.
- Повідомлення шифрується публічним ключем користувача якому ми відправляємо його та відправляється на сервер.
- Сервер пересилає це повідомлення клієнтській частині користувача, якому ми відправили повідомлення.

- Клієнтська частина користувача розшифровує повідомлення приватним ключем.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Авторизація користувача

На головній сторінці відображається модуль «Login.vue» який відповідає за отримання логіну від користувача, можна вважати, що це є авторизацією, але ми замінимо введення логіну і паролю лише логіном (Рис. 3.1). Він відображається поки поле «isLogin» === false.

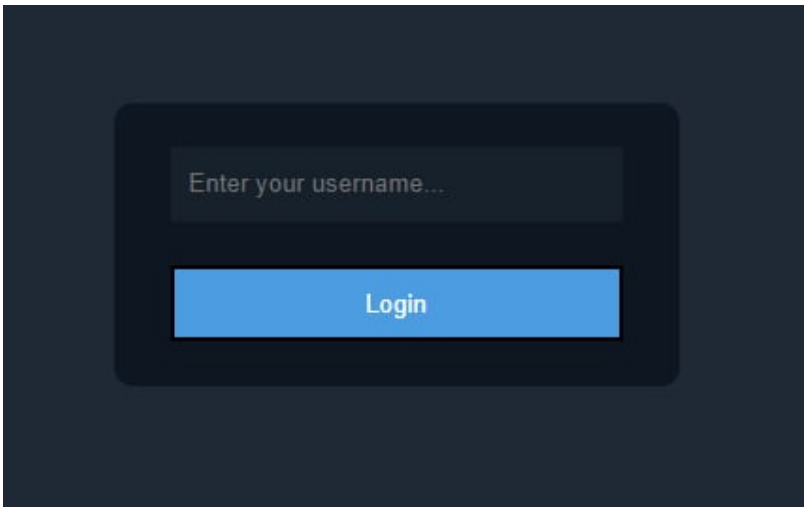

Рис. 3.1.1 Форма авторизації

Після вводу логіну і натисканні кнопки «Login» ми перевіряємо валідність логіну(чи він не є порожньою строкою) за допомогою методу checkUsername() і створюємо подію «setUsername» (Рис. 3.1.1).

```
methods: {
  checkUsername(){ /* Перевірка валідності логіну */
    if (this.username.length > 0){ /* Якщо логін не пуста строка */
      this.showError = false;
      return true;
    } else {
      this.showError = true;
      return false;
    }
  },
  login(){ /* Авторизація */
    const isUsernameValid = this.checkUsername();
    if (!isUsernameValid){
      return;
    }
    this.$emit('setUsername', this.username) /* Створення події setUsername */
  }
}
```

Рис. 3.1.2 Функції авторизації

Корінний компонент «App.vue» прослуховує подію «setUsername» і викликає функцію «login()», яка у свою чергу робить підключення.

```
login(username){ /* Функція авторизації */
  this.sessionUsername = username;
  let keys = NodeRSA({ b: 512 });  /* Генеруємо ключі */
  this.sessionPublicKey = keys.exportKey( format: "public"); /* Записуємо public ключ */
  this.sessionPrivateKey = keys.exportKey( format: "private"); /* Записуємо private ключ */
  socket.auth = { username: this.sessionUsername, publicKey: this.sessionPublicKey }
  socket.connect() /* Створюємо підключення */
  this.isLogin = true;
},
```

Рис. 3.1.3 Функція створення підключення

Після підключення сервер обробляє і надсилає йому масив з усіма онлайн клієнтами.

```
let users = []; /* Масив з підключеними клієнтами */


io.on('connection', (socket) => { /* Нове підключення */
    users.push({socketId: socket.id, username: socket.username, publicKey: socket.publicKey})
    io.sockets.emit( ev: 'receiveUsers', users) /* Створюємо подію receiveUsers */
    console.log(`New connection. ID: ${socket.id} Username: ${socket.username}`);
```

Рис. 3.1.4 Серверна функція обробки підключень

```
socket.on( ev: 'receiveUsers', listener: (users) => {
  this.users = users.filter((user) => user.socketId !== this.sessionWSID);
})
```

Рис. 3.1.5 Функція отримання всіх онлайн клієнтів

Оскільки «isLogin» === true, на головній сторінці відображається компонент «Chat.vue» (Рис. 3.1.6).

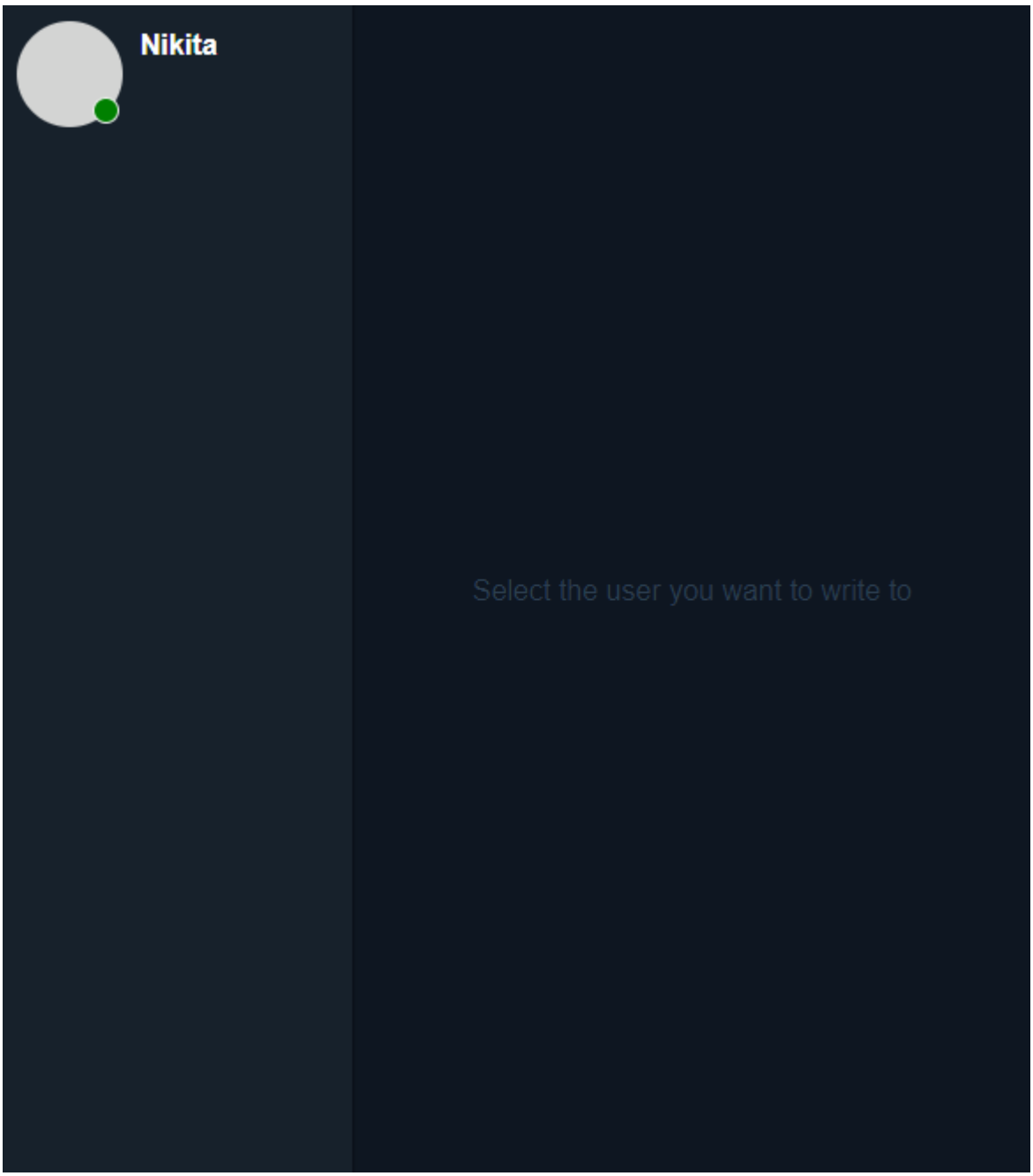

Рис. 3.1.6 Початковий екран компонента «Chat.vue»

3.2 Взаємодія з чатами

Симетричне шифрування є одним з методів шифрування для обміну повідомленнями і використовує один ключ як для шифрування, так і для розшифрування повідомлення. Основна ідея симетричного шифрування полягає в тому, що відправник і отримувач повідомлення використовують спільний ключ для забезпечення конфіденційності і цілісності даних. Повідомлення шифрується за допомогою ключа на відправнику і розшифровується за допомогою того ж ключа на отримувачі.

Асиметричне шифрування – це метод шифрування, в якому використовується пара ключів: приватний ключ і публічний ключ. Кожен користувач має свою пару ключів. Приватний ключ залишається секретним і відомим тільки власнику, тоді як публічний ключ розповсюджується серед інших користувачів. Основна ідея асиметричного шифрування полягає в тому, що повідомлення можна зашифрувати за допомогою публічного ключа, а розшифрувати його може тільки власник приватного ключа. Тобто, якщо ви хочете надіслати повідомлення користувачеві А, ви використовуєте його публічний ключ для шифрування повідомлення, і тільки користувач А, якому належить приватний ключ, може розшифрувати його.

Ось приклад використання асиметричного та симетричного шифрування у програмі: коли користувач А відкриває чат з користувачем Б, з яким до цього не спілкувався, через веб-сокети відбувається обмін симетричним ключем за допомогою публічного асиметричного ключа. Тобто, першочергово, клієнтська частина користувача А запитує у клієнтської частини користувача Б його публічний ключ, він у свою чергу генерує пару асиметричних ключів та відправляє у відповідь публічний ключ. Клієнтська частина користувача А генерує симетричний ключ, зберігає його у пам'ять пристрою, а потім шифрує його за допомогою публічного ключа, який відправила клієнтська частина користувача Б та відправляє його клієнтській частині користувача Б. Клієнтська частина користувача Б розшифровує зашифрований симетричний ключ за допомогою приватного ключа та також зберігає його у пам'яті пристрою, після чого у обох користувачів відкривається чат. Повідомлення якими вони будуть обмінюватись, будуть шифруватись симетричним ключем, яким вони обмінялись.

Після авторизації необхідно вибрати одного користувача, який зараз онлайн, після чого відкривається чат з ним, за це відповідає функція «selectChat()» (Рис 3.2.1)

```
selectChat(user){
  this.target = user;
},
```

Рис. 3.2.1 Функція вибору чату

За відправку повідомлення відповідає функція «sendMessage» яка створює подію «sendMessage» (Рис 3.2.2)

```
sendMessage(){ /* Відправлення повідомлення*/
  if (this.input_message.length > 0) {
    this.$emit('sendMessage', this.target, this.input_message);
    this.input_message = '';
  }
}
```

Рис. 3.2.2 Функція відправки повідомлення

```
sendMessage(target, message){ /* Шифрування та відправка повідомлення*/
  let encrypted_message = this.encrypt(target.publicKey, message) /* Шифрування */
  let raw_message = {from: this.sessionWSID, to: target.socketId, text: message}
  let prepared_message = Object.assign( target: {}, raw_message);
  prepared_message.text = encrypted_message

  socket.emit( ev: 'sendMessage', prepared_message) /* Відправка */
  this.saveMessage(target.socketId, raw_message) /* Збереження до історії листування */
}
```

Рис. 3.2.3 Функція відправки та шифрування повідомлення

Після відправки повідомлення на сервер, він пересилає його потрібному клієнту генеруючи подію «receiveMessage»

```
socket.on('sendMessage', (message) => {
    console.log(message)
    io.sockets.to(message.to).emit( ev: 'receiveMessage', message)
});
```

Рис 3.2.4 Серверна функція пересилки повідомлення

```
socket.on( ev: 'receiveMessage', listener: (message) => { /* Отримання повідомлення з серверу */
  message.text = this.decrypt(message.text) /* Дешифрування */
  this.saveMessage(message.from, message) /* Збереження до історії листування */
})
```

Рис 3.2.5 Функція отримання повідомлення

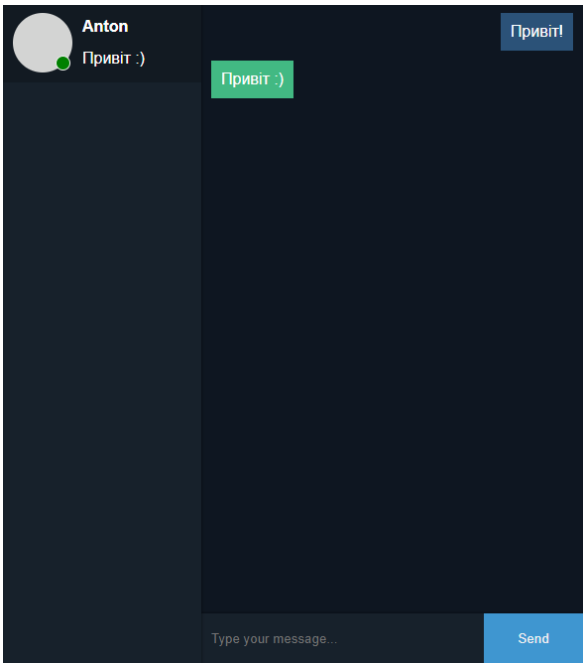

Рис 3.2.6 Листування

ВИСНОВКИ

В процесі розробки месенджера для дипломної роботи були обрані Node.js, Vue.js і Socket.io як основні технології. Вибір цих технологій був обґрунтований їхньою перспективністю, зручністю використання та широким застосуванням у сфері розробки веб-додатків.

Node.js був використаний в якості серверної платформи, оскільки він є легким для розуміння та швидким у виконанні. Він здатний обробляти багато одночасних з'єднань із клієнтами, що дуже важливо для реалізації функціональності месенджера. Node.js також має велику кількість пакетів та бібліотек, які полегшують розробку і розширення додатку.

Vue.js, як молодий фреймворк, був обраний для розробки користувацького інтерфейсу месенджера. Він пропонує просту та зрозумілу структуру компонентів, що дозволяє ефективно управляти складністю інтерфейсу та забезпечує швидку реакцію на зміни даних.

Socket.io був використаний для реалізації зв'язку в реальному часі між клієнтами і сервером. Ця бібліотека надає зручний API для створення багатокористувацьких чатів та передачі повідомлень у реальному часі. Вона підтримує двосторонній зв'язок і автоматичне перепідключення, що забезпечує надійність та стабільність з'єднання.

Результатом розробки є кросплатформений месенджер, який працює у браузері. Це означає, що користувачі можуть використовувати його на різних пристроях та пристроях без необхідності установки додаткового програмного забезпечення. Кросплатформеність дозволяє забезпечити широкий охоплення аудиторії та полегшує розповсюдження месенджера серед користувачів.

Окрім того, застосування симетричного та асиметричного шифрування в месенджері дозволяє забезпечити конфіденційність та цілісність переданих повідомлень. Обмін симетричним ключем за допомогою асиметричного шифрування забезпечує безпеку зв'язку між користувачами та захист від несанкціонованого доступу до повідомлень. Висновки відображають успішне використання сучасних технологій для розробки месенджера, який має потенціал стати популярним та корисним інструментом для комунікації між користувачами. Розробка кросплатформного месенджера на базі Node.js, Vue.js і Socket.io дозволяє забезпечити ефективну роботу між клієнтами та сервером, забезпечуючи швидку та безпечну передачу повідомлень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Express. Express.js – Вікіпедія. Отримано з [URL].
2. Socket.io. Introduction | Socket.IO. Отримано з [URL].
3. Node.js. Документація | Node.js. Отримано з [URL].
4. Vue.js. Creating a Vue Application | Vue.js. Отримано з [URL].
5. Node-RSA. node-rsa - nmp. Отримано з [URL].

ДОДАТКИ

App.vue

```
<template>
<Login v-if="!isLogin" @setUsername="login"/>
<Chat v-else :users="users" :messages="messages" :sessionWSID="sessionWSID" @sendMessage="sendMessage"/>
</template>

<script>
import Login from "@/views/Login";
import NodeRSA from "node-rsa";
import socket from "@/socket";
import Chat from "@/views/Chat";

export default {
  components: {Chat, Login},
  data(){
    return {
      isLogin: false,
      sessionWSID: '',
      sessionUsername: '',
      sessionPublicKey: '',
      sessionPrivateKey: '',
      users: [],
      messages: new Map([])
    }
  },
  methods: {
    login(username){
      this.sessionUsername = username;
      let keys = NodeRSA({ b: 512 });
      this.sessionPublicKey = keys.exportKey("public");
      this.sessionPrivateKey = keys.exportKey("private");
      socket.auth = { username: this.sessionUsername, publicKey: this.sessionPublicKey }
```

```
    socket.connect()
      this.isLogin = true;
    },

    encrypt(partnerPublicKey, message){
      const key = new NodeRSA(partnerPublicKey);
      return key.encrypt(message, "base64");
    },

    decrypt(message){
      const key = new NodeRSA(this.sessionPrivateKey);
      return key.decrypt(message, "utf8");
    },

    saveMessage(socketId, message){
      if(this.messages.has(socketId)){
        this.messages.set(socketId, [...this.messages.get(socketId), message])
      } else {
        this.messages.set(socketId, [message])
      }
    },

    sendMessage(target, message){
      let encrypted_message = this.encrypt(target.publicKey, message)
```

```
      let raw_message = {from: this.sessionWSID, to: target.socketId, text: message}
      let prepared_message = Object.assign({}, raw_message);
      prepared_message.text = encrypted_message

      socket.emit('sendMessage', prepared_message)
      this.saveMessage(target.socketId, raw_message)
    }
  },
  created() {
    socket.on('connect', () => {
      this.sessionWSID = socket.id
    })

    socket.on('receiveUsers', (users) => {
      console.log('receivedUsers')
      this.users = users.filter((user) => user.socketId !== this.sessionWSID);
    })

    socket.on('receiveMessage', (message) => {
      message.text = this.decrypt(message.text)
      this.saveMessage(message.from, message)
    })

    socket.on('disconnect', () => {
      this.isLogin = false;
    })
  },
  beforeUnmount() {
    socket.disconnect()
  }
}
</script>

<style>
*, *:before, *:after {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
```

```
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  /*text-align: center;*/
  color: #2c3e50;
  background: #1F2936;

  height: 100vh;
  width: 100%;

  display: flex;
  align-items: center;
  justify-content: center;
}

#nav {
  padding: 30px;
}

#nav a {
  font-weight: bold;
```

```
  color: #2c3e50;
}

#nav a.router-link-exact-active {
  color: #42b983;
}
</style>
```

Login.vue

```
<template>
<div class="login">
  <input type="text" class="form-control form_input" :class="{error: showError}" placeholder="Enter your username..." v-model="username" @input="checkUsername()">
  <button class="form-control form_button" @click="login()">Login</button>
</div>
</template>

<script>
export default {
  name: "Login",
  data(){
    return {
      showError: false,
      username: ''
    }
  },
  methods: {
    checkUsername(){
      if (this.username.length > 0){
        this.showError = false;
        return true;
      } else {
        this.showError = true;
        return false;
      }
    },
    login(){
      const isUsernameValid = this.checkUsername();
      if (!isUsernameValid){
```

```
return;
      }
      this.$emit('setUsername', this.username)
    }
  }
}
</script>

<style scoped>
  .login {
    height: 150px;
    width: 300px;
    background: #0E1621;
    display: flex;
    flex-direction: column;
    align-items: center;
```

```
    justify-content: space-evenly;
    border-radius: 10px;
  }

  .form-control {
    height: 40px;
    width: 80%;
  }

  .form_input {
    padding-left: 10px;
    background: #17212B;
    border: none;
    outline: none;
    color: #fff;
  }

  .form_button {
    background: #4C9CE2;
    color: #fff;
  }

  .error {
    border: 2px solid red;
    outline-color: red;
    border-radius: 3px;
  }
</style>
```

Chat.vue

```
<template>
  <div class="chat">
    <div class="users">
      <div class="user online" v-for="user in users" :key="user.socketId" :class="{active: user.socketId ===
target.socketId}" @click="selectChat(user)">
        <div class="user__icon"></div>
        <div class="user__name">{{ user.username }}</div>
        <div class="user__last-message">{{messages.has(user.socketId) ? messages.get(user.socketId).slice(-1)[0].text :
''}}</div>
      </div>
    </div>
    <div class="messaging" @keydown.enter="sendMessage()">
```

```
      <div class="messages">
        <div class="messages_placeholder" v-if="!target.socketId">Select the user you want to write to</div>

        <div class="message" v-else v-for="message in messages.get(target.socketId)" :class="{my: message.from ===
sessionWSID}"><div class="message__text">{{ message.text }}</div></div>
      </div>
      <div class="controls" :style="{display: target.socketId ? 'flex' : 'none'}">
        <input type="text" placeholder="Type your message..." class="control control__input" v-model="input_message">
        <button class="control control__button" @click="sendMessage()">Send</button>
      </div>
    </div>
  </div>
</template>
```

```
<script>
export default {
  name: "Chat",
  props: {
    users: {
      type: Array,
      require: true
    },
    messages: {
      type: Map,
      require: true
    },
    sessionWSID: {
      type: String,
      require: true
    }
  },
  data(){
    return {
      input_message: '',
      target: {},
    }
  },
  methods: {
    selectChat(user){
      this.target = user;
    },

    sendMessage(){
      if (this.input_message.length > 0) {
        this.$emit('sendMessage', this.target, this.input_message);
        this.input_message = '';
      }
    }
  }
}
</script>

<style scoped>
.chat {
  height: 100%;
  width: 100%;
  background: #f2f2f2;
  display: flex;
}

.messaging {
  height: 100%;
  flex: 1;
```

```
  display: flex;
  flex-direction: column;
  background: #0E1621;
}

.messages {
  flex: 1;
  overflow-y: auto;
  padding: 10px;
  display: flex;
  flex-direction: column;
```

```
}
.message {
  align-self: start;
}
.message__text {
  padding: 10px;
  margin-bottom: 10px;
  display: inline-block;
  background: #42b983;
  color: #fff;
}

.message.my{
  align-self: end;
}
.message.my .message__text {
  background: #2B5278;
}
.messages_placeholder {
height: 100%;
width: 100%;
display: flex;
flex-direction: column;
justify-content: center;
text-align: center;
}
.controls {
height: 50px;
  box-shadow: 0 -1px 2px rgba(0,0,0,0.3);
}
.users {
  height: 100%;
  width: 200px;
  z-index: 999;
  background: #17212B;
  box-shadow: 1px 0 2px rgba(0,0,0,0.3);
}
.user {
height: 80px;
width: 100%;
position: relative;
}
.user:hover {
  background: #202B36;
}
.user.active {
  background: rgba(0,0,0,0.2);
}

.user .user__icon {
height: 60px;
width: 60px;
```

```
background: #d3d4d3;
border-radius: 50%;
position: absolute;
top: 10px;
left: 10px;
}
.user .user__name {
  color: #fff;
```

```
position: absolute;
top: 15px;
left: 80px;
font-weight: bold;
}
.user .user__last-message {
position: absolute;
bottom: 15px;
left: 80px;
  color: #fff;
}
.user.online .user__icon:before {
content: '';
position: absolute;
width: 15px;
height: 15px;
background: green;
border: 1px solid #f2f2f2;
border-radius: 50%;
bottom: 2px;
right: 2px;
}

.controls{
display: flex;
  background: #17212B;
}

.control {
height: 100%;
}

.control__input {
flex-grow: 1;
padding-left: 10px;
  border: none;
  outline: none;
  background: transparent;
  color: #fff;
}

.control__button {
width: 100px;
  border: none;
  background: #3F96D0;
  color: #f2f2f2;
}

</style>
```

Server/index.js

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const path = require('node:path')

const cors = require('cors')
app.use(cors())
```

```
const io = require("socket.io")(server, {
    cors: {
        origin: ["localhost:3000", "127.0.0.1:3000"],
        methods: ["GET", "POST"],
        transports: ['websocket', 'polling'],
        credentials: true
    },
    allowEIO3: true
});

app.use('/', express.static(path.join(__dirname, '../dist')));
app.get('/', (req, res) => {
    res.sendFile(__dirname + '/dist/index.html');
});



io.use(async (socket, next) => {
    const {username, publicKey} = socket.handshake.auth;
    socket.username = username;
    socket.publicKey = publicKey;
    return next();
});

let users = []; /* Масив з підключеними клієнтами */


io.on('connection', (socket) => { /* Нове підключення */
    users.push({socketId: socket.id, username: socket.username, publicKey:
socket.publicKey})
    io.sockets.emit('receiveUsers', users) /* Створюємо подію receiveUsers */
    console.log(`New connection. ID: ${socket.id} Username: ${socket.username}`);

    socket.on('sendMessage', (message) => {
        console.log(message)
        io.sockets.to(message.to).emit('receiveMessage', message)
    });

    socket.on('disconnect', () => {
        users.splice(users.indexOf(socket.id), 1);
        io.sockets.emit('receiveUsers', users)
        console.log(`Disconnect ID: ${socket.id} Username: ${socket.username}`);
    })
});

const PORT = 3000;
server.listen(3000, () => {
    console.log(`listening on http://localhost:${PORT}`);
});
```