Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
Розробка методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування

Виконав: студент II курсу
групи _____ спеціальності
121 – Інженерія програмного забезпечення
Пілецький Володимир Данилович

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.

«____»_____2021 р.

Опонент: к.т.н., доц. каф. КН Арсенюк І.Р.

«____»_____2021 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)

«____»_____2021 р.

ВНТУ – 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 13 » вересня 2021 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Пілецькому Володимиру Даниловичу

1. Тема роботи – розробка методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доц. каф. ПЗ, затверджені наказом вищого навчального закладу від « 24 » вересня 2021 р. № 277.

2. Строк подання студентом роботи
____1 грудня 2021 р. _____

3. Вихідні дані до роботи: сімейство операційних систем Windows та Ubuntu; середовище розробки – Intellij IDEA WebStorm, формат зберігання конфігурації – JSON; мова програмування – JavaScript; фреймворк для серверної частини – Express; фреймворки для клієнтської частини – Electron, React; розподілена система управління версіями – Git.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання та постановка задачі дослідження; розробка методів шифрування та дешифрування інформації; розробка клієнтської та серверної частини месенджеру; тестування програмного продукту; економічна частина; висновки; список використаних джерел; додатки

5. Перелік графічного матеріалу: аналоги додатку; структура інтерфейсу додатку; діаграми класів додатку; алгоритм виконання агрегації; загальний алгоритм роботи додатку; результати тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д.І., к.т.н., доц. каф. ПЗ		
5	Ратушняк О.Г., к.е.н., доц. каф. ЕПВН		

7. Дата видачі завдання ___14 вересня 2021 р._____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задачі дослідження	15.09. 2021 - 21.10.2021	Вик.
2	Розробка структур та алгоритмів інформаційної системи месенджеру	22.10.2021- 31.10,2021	Вик.
3	Розробка серверної та клієнтської частини програмного додатку	01.11.2021- 19.11.2021	Вик.
4	Тестування програмного продукту	20.11.2021- 21.11.2021	Вик.
5.	Економічна частина	22.11.2021- 30.11.2021	Вик.

Студент _____ **Пілецький В.Д.**

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Катєльніков Д.І.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК. 004.4.004.92

Пілецький В.Д. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2021. с.

У магістерській кваліфікаційній роботі розроблено програмний додаток месенджер, який складається з клієнтської та серверної частини призначений для миттєвого обміну повідомленнями між користувачами. Під час виконання магістерської кваліфікаційної роботи було проаналізовано предметну область, проаналізовано основні аналоги, виявлено їх головні недоліки та переваги, на основі яких було обрано основні функції для месенджеру. Також було створено гнучку архітектуру для подальшої інтеграції нового функціоналу.

Дістав подальшого розвитку метод захисту інформаційної системи-месенджеру, який значно підвищує конфіденційність інформації, так щоб доступ до неї міг отримати лише сам користувач.

За допомогою розроблених програмних засобів можна значно покращити конфіденційний обмін повідомленнями між користувачами. Розроблено клієнтську та серверну частину.

Обрано мову програмування JavaScript з використанням програмної платформи Node.js а також фреймворків: Express.js для серверної частини, Electron та React для клієнтської частини.

Ключові слова – месенджер, шифрування, асиметричне шифрування, програмний додаток, RSA.

ABSTRACT

Piletskiy V.D. Master's degree in specialty 121 – software engineering, educational program – software engineering. Vinnytsia: VNTU, 2021

In the master's qualification work developed a software application messenger, which consists of a client and server part designed for instant messaging between users. During the master's qualification work, the subject area was analyzed, the main analogues were analyzed, their main disadvantages and advantages were identified, on the basis of which the main functions for the messenger were selected. A flexible architecture has also been created to further integrate the new functionality.

The method of protection of the messenger information system was further developed, which significantly increases the confidentiality of information, so that only the user could access it.

With the help of developed software, you can significantly improve the confidential messaging between users. Client and server part developed.

The JavaScript programming language using the Node.js software platform was chosen, as well as the frameworks: Express.js for the server part, Electron and React for the client part.

Keywords - messenger, encryption, asymmetric encryption, software application, RSA.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний етап розвитку відкритого суспільства з необхідністю включає в себе розвиток соціальної комунікації, нарощування горизонтальних зв'язків між членами суспільства, появу можливості спілкування, яке не обмежено ані географічними рамками, ані расовими, релігійними тощо. І одну з перших ролей у цій комунікації відіграють інформаційні засоби комунікації, найбільш простими і демократичними з яких є миттєві комунікатори - месенджери (англійською instant messenger). Ці програми дозволяють в реальному часі здійснювати обмін текстовими повідомленнями між комп'ютерами або іншими пристроями користувачів через комп'ютерні мережі (як правило через інтернет). Повідомлення зазвичай передаються між двома (канал) або більше сторонами (чат), коли кожен користувач вводить текст і запускає передачу одержувачам, які всі підключені до спільної мережі. Простота використання подібних систем зробила їх дуже популярними: спільноти сучасних найвідоміших месенджерів налічують від кількох десятків мільйонів до мільярда та більше користувачів! Нажаль широке охоплення аудиторії дає не тільки широкі можливості для пересічних користувачів, але й становить для них певну небезпеку: зловмисники можуть читати повідомлення особистої переписки, можуть здійснювати атаки man-in-the-middle та іншими способами порушувати права учасників обміну інформації. Тому актуальність задач розробки систем забезпечення конфіденціальності повідомлень в інформаційній системі-месенджері важко переоцінити.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення рівня безпеки даних користувачів інформаційної системи месенджеру з використанням асиметричного шифрування.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів шифрування інформації для визначення напрямку підвищення рівня безпеки інформаційної системи месенджеру;
- запропонувати новий:
  - метод захисту інформаційної системи месенджеру;
  - метод створення та читання зашифрованих повідомлень, таким чином, щоб для комунікації між двома користувачами не потрібно було зберігати ключі шифрування централізовано (на сервері);
- розробити програмне забезпечення на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів захисту інформаційної системи месенджеру.

Об’єктом дослідження є процес соціальної комунікації з використанням інформаційної системи месенджера.

Предмет дослідження – методи та алгоритми асиметричного шифрування даних користувача інформаційної системи месенджера.

Методи дослідження. У процесі дослідження використовувались: методи теорії кодування та захисту інформації для розробки методів; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Подальшого розвитку отримав метод захисту інформаційної системи месенджерів, який на відміну від існуючих методів використовує асиметричне шифрування з приватним ключем, який генерується при створенні аккаунта і ніколи не передається через жодні канали зв’язку, таким чином підвищуючи рівень безпеки інформації користувача.

2. Подальшого розвитку отримав метод асиметричного шифрування інформації користувача, який на відміну від існуючих методів здійснює паралельне шифрування повідомлень для надсилання до приймача та

збереження у базі даних, що дозволяє підвищити продуктивність шифрування інформації.

Практична цінність отриманих результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби захисту інформаційної системи месенджеру з використанням асиметричного шифрування.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованій праці [1], опублікованих у співавторстві, автору належать такі результати:

– аналіз методів і засобів реалізації програмного продукту;

– реалізація програмного продукту.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися на міжнародній науково-практичній Internet-конференції «Електронні інформаційні ресурси: створення, використання, доступ», ВНТУ, Вінниця, 9-10 листопада 2021 року.

Публікації. Основні результати досліджень опубліковано в матеріалах наукової конференції.

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’ятьох розділів, висновків, списку літератури, що містить 30 найменувань, 5 додатків. Робота містить 61 ілюстрацію, 5 таблиць.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану месенджерів

Месенджер - це тип онлайн-чату, що дозволяє передавати текст в режимі реального часу через Інтернет або іншу комп’ютерну мережу. Повідомлення зазвичай передаються між двома або більше сторонами, коли кожен користувач вводить текст і запускає передачу одержувачам, які всі підключені до спільної мережі. Він відрізняється від електронної пошти тим, що розмови за допомогою миттєвих повідомлень відбуваються в режимі реального часу (отже, "миттєвий"). Більшість сучасних месенджерів використовують технологію push і також додають інші функції, такі як смайлики (або графічні смайлики), передача файлів, чат -боти, голос по IP або можливості відеоконференції[2].

Системи миттєвого обміну повідомленнями, як правило, полегшують зв'язки між зазначеними відомими користувачами (часто використовують список контактів, також відомий як "список друзів" або "список друзів"), і можуть бути окремими програмами або інтегровані, наприклад. ширшу платформу соціальних медіа або веб-сайт, де її можна, наприклад, використовувати для спілкування у розмовах. Месенджер також може складатися з розмов у «чатах». Залежно від протоколу месенджеру, технічна архітектура може бути одноранговою (пряма передача «точка-точка») або клієнт-сервер (центр обслуговування чату повторно передає повідомлення від відправника до пристрою зв'язку). Зазвичай його відрізняють від текстових повідомлень, які, як правило, простіші і зазвичай використовують мережі стільникового зв'язку[2].

Для подібного роду комунікації необхідна клієнтська програма, так званий месенджер (messenger — кур'єр). Відмінність від електронної пошти тут у тому, що обмін повідомленнями йде в реальному часі (англ. instant – миттєво). Більшість IM-клієнтів дозволяє бачити, чи підключені абоненти, занесені до списку контактів. У ранніх версіях програм все, що друкував користувач, відразу

передавалося. Якщо він робив помилку та виправляв її, це теж було видно. У такому режимі спілкування нагадувало телефонну розмову. У сучасних програмах повідомлення з'являються на моніторі співрозмовника після закінчення редагування і відправлення повідомлення[2].

Як правило, месенджери не працюють самостійно, а підключаються до центрального комп'ютера мережі обміну повідомленнями, що називається сервером. Тому месенджери називають клієнтами (клієнтськими програмами). Термін є поняттям із клієнт-серверних технологій[2].

Широкому колу користувачів відомо кілька популярних мереж (і клієнтів) обміну повідомленнями, таких як IRC, Skype, MyChat, ooVoo, AIM, ICQ, MSN, Yahoo!, Jitsi, XMPP, Gem4me. Кожна з цих мереж розроблена окремою групою розробників, має окремі сервери та протоколи, відрізняється своїми правилами та особливостями. Між різними мережами зазвичай немає прямого зв'язку (тільки в XMPP існує поняття міжмережевого транспорту), таким чином користувач мережі Skype не може зв'язатися з користувачем мережі ICQ, однак ніщо не заважає бути одночасно користувачем декількох мереж[2].

Майже для кожної мережі є свій месенджер, розроблений тією ж командою розробників. Так, для користування трьома останніми з вищезгаданих мереж розробниками пропонують програми з однойменними назвами: ICQ, Windows Live Messenger, Yahoo! Messenger, а також Skype. Таким чином, якщо один з адресатів користується лише мережею ICQ, а інший - тільки мережею MSN, то можна спілкуватися з ними одночасно, встановивши на своєму комп'ютері і ICQ, і MSN Messenger та зареєструвавшись в обох мережах (або через відповідні транспорти XMPP) [2].

Як альтернативний месенджер можна вибрати програму стороннього виробника, як комерційну, так і безкоштовну. Популярними альтернативними програмами для спілкування в мережі ICQ є QIP 2005/QIP Infium, Psi/Psi+ (через XMPP-транспорт), Trillian, Miranda IM, Pidgin, MyChat. Також деякі з них дозволяють підключатися одночасно до кількох мереж, тобто є мультипротокольними, що позбавляє необхідності встановлювати окремий

месенджер для кожної мережі та дозволяє спілкуватися з усіма адресатами єдиним чином незалежно від мережі; всі перелічені у попередньому реченні клієнти ICQ, крім версії QIP 2005, підтримують і протокол XMPP[3].

Більшість IM-мереж використовує закриті протоколи, тому альтернативні клієнти теоретично можуть мати меншу кількість базових функцій, ніж офіційні, хоча на практиці частіше буває навпаки. Однак при змінах протоколу на стороні сервера мережі альтернативні клієнти можуть раптово перестати працювати[3].

В якості альтернативи пропрієтарним протоколам для IM був розроблений відкритий і добре розширюваний протокол XMPP (також відомий як Jabber), що використовується в таких сервісах, як Google Talk, Я. Онлайн та ін. Цей протокол часто використовується для організації спілкування в корпоративних та інших локальних мережах має ряд істотних переваг, як, наприклад, шифрування повідомлень і стабільність на нестійких каналах зв'язку. Протокол децентралізований, його архітектура нагадує електронну пошту, де можливе спілкування між користувачами, які мають облікові записи на різних серверах. Якщо порушиться робота одного сервера, це не вплине на роботу всієї мережі[3].

Протокол IRC був першим, який досяг широкого поширення для месенджерів. Пізніше, у 1990 -х роках, ICQ був одним з перших закритих та комерціалізованих месенджерів, а згодом з’явилося кілька конкуруючих служб, коли він став популярним у мережі Інтернет . Починаючи з першого представлення у 2005 році, BlackBerry Messenger, який спочатку був доступний лише на смартфонах BlackBerry, незабаром став одним із найпопулярніших мобільних додатків для миттєвих повідомлень у всьому світі. Наприклад, BBM був найбільш використовуваним додатком для мобільних повідомлень у Великобританії та Індонезії. Месенджери залишається дуже популярними сьогодні; Месенджери є найбільш широко використовуваними програмами для смартфонів: у 2018 році у WhatsApp та Facebook Messenger було 1,3 мільярди користувачів щомісячно, та понад 980 мільйонів користувачів у китайського месенджеру WeChat[3].

1.2 Порівняльний аналіз аналогів

На даний момент існує велика кількість месенджерів. Нижче наведено деякі з них.

Telegram (рисунок 1.1) – кросплатформенна система миттєвого обміну повідомленнями (месенджер) з функціями VoIP, що дозволяє обмінюватися текстовими, голосовими та відеоповідомленнями, стікерами та фотографіями, файлами багатьох форматів. Також можна здійснювати відео- і аудіозвонкі, організовувати конференції, розраховані на багато користувачів групи і канали. Клієнтські програми Telegram доступні для Android, iOS, Windows Phone, Windows, macOS і GNU / Linux. Кількість щомісячних активних користувачів сервісу станом на січень 2021 року становить близько 500 млн осіб[4].

Недоліком даного месенджеру є те, що з максимальним рівнем захищеності повідомлення шифруються лише у спеціальних захищених чатах.

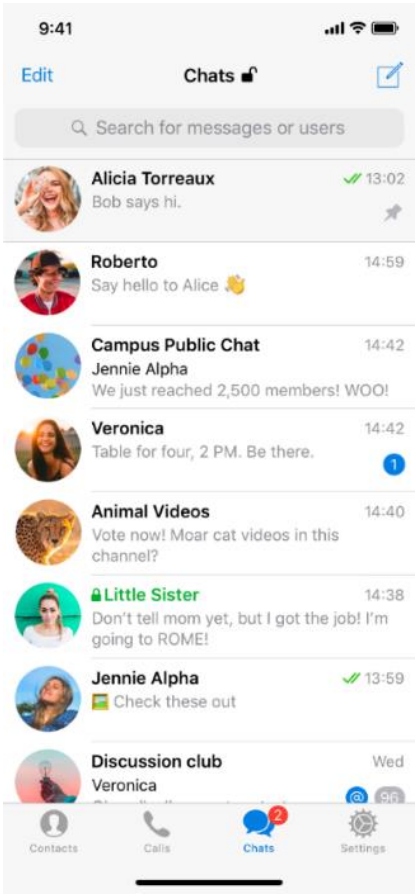


Рисунок 1.1 – Вікно додатку Telegram

Viber (рисунок 1.2) – месенджер, який дозволяє відправляти повідомлення, здійснювати відео та голосові VoIP-дзвінки через інтернет. Голосові дзвінки між користувачами Viber повністю безкоштовні (оплачується лише інтернет трафік по тарифу оператору зв'язку). Viber дає можливість відправляти текстові, голосові, відео повідомлення, документи, зображення, відеозаписи та файли[5].

Для авторизації користувачів та пошуку контактів додаток використовує номер телефону та передає вміст телефонної адресної книги на сервери компанії Viber Media S.à r.l., Люксембург. Також вони збирають інформацію про здійснені дзвінки та передані повідомлення, довжину дзвінків, учасників дзвінків та чатів[5].

Недоліком є незручний спосіб синхронізації повідомлень, при втраті телефону є ймовірність втратити всю попередню переписку, а також наявність платних функцій.

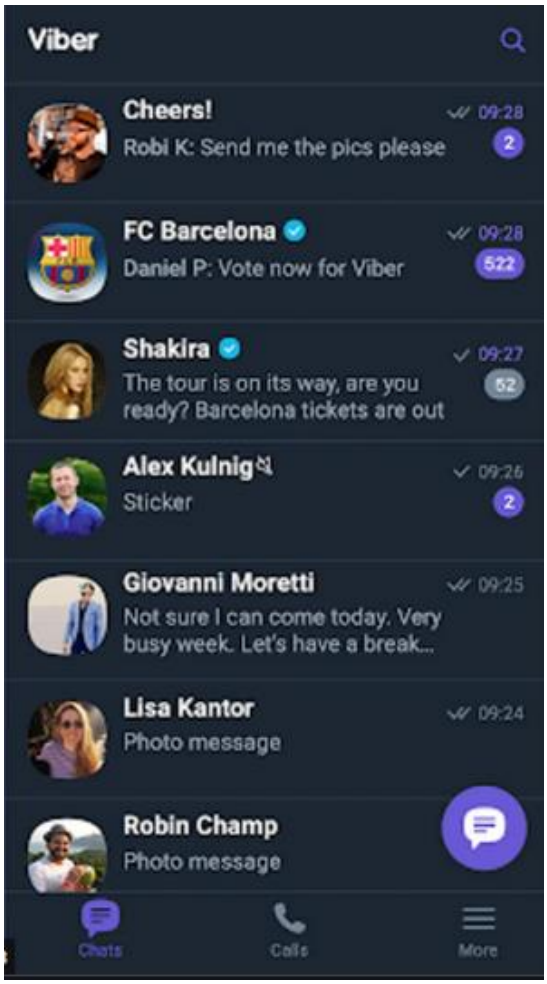


Рисунок 1.2 – Вікно додатку Viber

WhatsApp (рисунок 1.3) - популярний месенджер для мобільних і інших платформ з підтримкою голосового зв'язку і відеозв'язку. Дозволяє пересилати текстові повідомлення, зображення, відео, аудіо, електронні документи та навіть програмні установки через Інтернет[6].

Клієнт працює на платформах Android, iOS, S40, KaiOS, а також Windows, macOS і у вигляді веб-додатки[6].

Компанія WhatsApp Inc., яка створила мессенджер, була заснована Яном Кумом і Брайаном Ектон 24 лютого 2009 року і розташована в Маунтін-В'ю, США; з жовтня 2014 року належить Facebook Inc. З 2016 додаток офіційно стало безкоштовним і до цього дня є таким, користувач оплачує лише використаний додатком інтернет-трафік. Додатком користується понад мільярд людей[6].

Недоліком даного додатку є те, що він часто піддається хакерським атакам, після чого дані користувачів сервісу потрапляють до рук зловмисників.

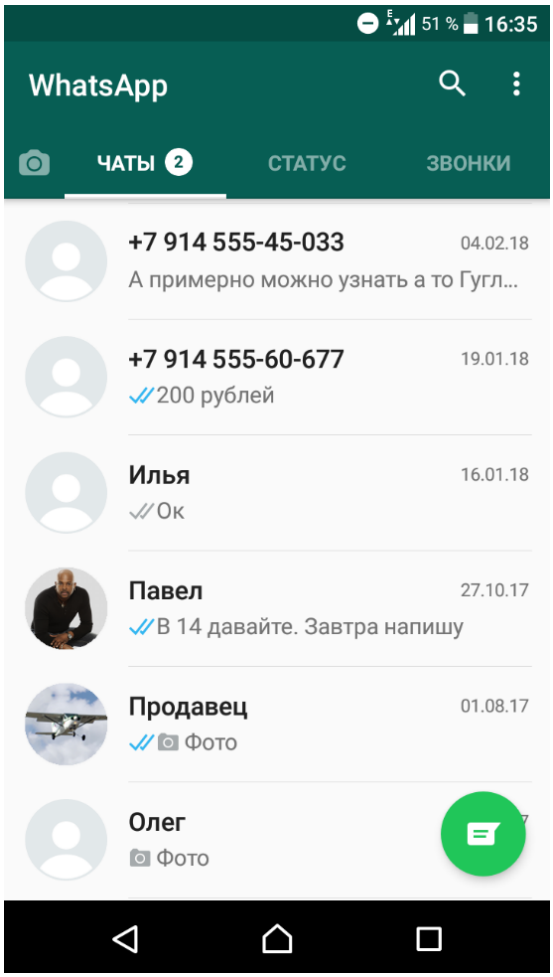


Рисунок 1.3 – Вікно додатку WhatsApp

Враховуючи недоліки зазначених вище програм можна зробити висновок, що розробка власної реалізації є доцільною. Буде розроблено кросплатформений додаток месенджер з використанням асиметричного шифрування для захисту його інформаційної системи.

1.3 Аналіз методів і засобів реалізації програмного продукту

Шифрування – оборотне перетворення інформації з метою приховування від неавторизованих осіб, з наданням, в цей же час, авторизованим особам доступу до неї. Головним чином, шифрування служить завданням дотримання конфіденційності інформації, що передається. Важливою особливістю будь-якого алгоритму шифрування є використання ключа, який стверджує вибір конкретного перетворення з сукупності можливих даного алгоритму[7].

Користувачі є авторизованими, якщо вони мають певний ключ. Вся складність, і, власне, завдання шифрування полягає в тому, як саме реалізований цей процес[7].

Шифрування складається з двох складових – шифрування та розшифрування[7].

За допомогою шифрування забезпечують три стани безпеки інформації:

- Конфіденційність – шифрування використовується для приховування інформації від неавторизованих осіб при передачі або зберіганні.
- Цілісність – шифрування використовується для запобігання зміни інформації при передачі або зберіганні.
- Ідентифікованість – шифрування використовується для аутентифікації джерела інформації та запобігання відмови відправника інформації від того факту, що дані були відправлені саме ним.

Для того, щоб прочитати зашифровану інформацію, приймаючій стороні необхідні ключ і дешифратор (пристрій, що реалізує алгоритм розшифрування).

Ідея шифрування полягає в тому, що зловмисник, перехопивши зашифровані дані і не маючи до них ключа, не може ні прочитати, ні змінити передану інформацію. Крім того, в сучасних криптосистемах (з відкритим ключем) для шифрування, розшифрування даних можуть використовуватися різні ключі. Однак, з розвитком криптоаналізу, з'явилися методики, що дозволяють дешифрувати закритий текст без ключа. Вони засновані на математичному аналізі переданих даних[7].

Шифрування поділяється на 2 типи – симетричне та асиметричне.

Симетричне шифрування - схема шифрування, у якій ключ шифрування, та ключ дешифрування збігаються, або один легко обчислюється з іншого та навпаки, на відміну від асиметричного, де ключ дешифрування важко обчислити[8].

Асиметричне шифрування - ефективні системи криптографічного захисту даних, які також називають криптосистемами з відкритим ключем. В таких системах для зашифровування даних використовують один ключ, а для розшифровування — інший (звідси і назва — асиметричні). Перший ключ є відкритим і може бути опублікованим для використання усіма користувачами системи, які шифрують дані. Розшифровування даних за допомогою відкритого ключа неможливе. Для розшифровування даних отримувач зашифрованої інформації використовує другий ключ, який є секретним (закритим). Зрозуміло, що ключ розшифровування не може бути визначеним з ключа зашифровування[9].

Головна перевага асиметричного шифрування в тому, що воно дозволяє людям, що не мають наперед наявної домовленості про безпеку, обмінюватися секретними повідомленнями. Необхідність відправникові й одержувачеві погоджувати таємний ключ по спеціальному захищеному каналу цілком відпала. Прикладами криптосистем з відкритим ключем є Схема Ель-Гамаля (названа на честь автора, Тахера Ель-Гамаля), RSA (названа на честь винахідників: Рона Рівеста, Аді Шаміра і Леонарда Адлмана), Діффі-Геллмана і DSA, англ. Digital Signature Algorithm (винайдений Девідом Кравіцом)[10].

Перевагою симетричного шифрування над асиметричним полягає в швидкості роботи, але при цьому головним недоліком симетричного шифрування є необхідність публічної передачі ключів «з рук в руки». На цей недолік не можна не звертати увагу, так як при такій системі стає майже неможливим використання симетричного шифрування з необмеженою кількістю користувачів[10].

Отже, після проведеного аналізу методів для реалізації захисту інформаційної системи месенджеру було вирішено використовувати асиметричне шифрування, а саме – використання алгоритму RSA, так як системи з публічним та приватним ключем мають більшу криптографічну стійкість, а також дозволяють зберігати ключі розшифрування інформації децентралізовано (тобто безпосередньо на пристрої користувача).

1.4 Постановка задач магістерської кваліфікаційної роботи

Після аналізу стану месенджерів, порівняння існуючих аналогів та аналізу методів і засобів реалізації програмного продукту було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

- розробити програмний додаток месенджер
- розробити модуль шифрування повідомлень
- розробити модуль дешифрування повідомлень
- розробити серверну частину месенджеру
- розробити клієнтську частину програмного продукту
- провести тестування програмного продукту

1.5 Висновки

В даному розділі було розглянуто аналіз стану месенджерів, що показав що створення месенджеру є актуальним. Було розглянуто такі аналоги як: Viber, Telegram, WhatsApp. Порівняння існуючих аналогів показало, що необхідно

розробити власний додаток. Також було проведено аналіз методів реалізації програмного продукту. Було обрано метод шифрування. Були розглянуті симетричне та асиметричне шифрування. Після порівняння було обрано асиметричне шифрування з використанням алгоритму RSA. В результаті цього було розроблено основні завдання, які необхідно виконати при розробці програмного додатку.

2 РОЗРОБКА МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ МЕСЕНДЖЕРІВ

2.1 Аналіз інформаційного забезпечення

Інформаційне забезпечення – це сукупність реалізованих рішень, нормативної бази, форм документів, і щодо обсягу, розміщення і форм організації інформації, яка знаходиться в системі автоматизованого оброблення економічної інформації чи в інформаційній системі [11].

До основних принципів створення інформаційного забезпечення відносять: достовірність, контроль, цілісність, єдність, гнучкість, захист від несанкціонованого доступу, адаптивність, стандартизація і уніфікація, мінімізація помилок введення-виведення інформації[11].

При розробці інформаційної системи розробка інформаційного забезпечення є одною із найважливіших складових та повинна забезпечувати:

- єдність і зберігання інформації, необхідної для розв’язання задач;
- єдність інформаційних масивів для всіх задач інформаційних систем;
- однократність уведення інформації та її багатоцільове використання;
- різні методи доступу до даних;
- низьку вартість витрат на зберігання та використання даних, а також на внесення змін.

Усі дані у додатку будуть отримувати безпосередньо від користувачів. При реєстрації користувачу необхідно буде вказати власне ім’я, фамілію, прізвисько, а також пошту. Далі при користуванні додатком користувач зможе обмінюватись повідомленнями з іншими користувачами, а також налаштовувати власний профіль, тобто змінювати інформацію про себе, а також оновлювати фото профілю.

Усі дані користувача, окрім фото профілю зберігаються у базі даних, у форматі рядку. Фото профілю користувачів зберігаються безпосередньо на сервері, в той час як в базі даних зберігається лише посилання на файл.

2.2 Розробка моделі системи

Додаток представляє собою систему, що має серверний та клієнтський модулі. Система постійно зберігає всі зміни у базі даних на стороні серверної частини. Для роботи клієнтської частини з серверною потрібна авторизація та реєстрація.

Серверна частина містить базу даних та блок додатку системи. База даних призначена для зберігання, зміни та обробки інформації. Блок додатку включає в себе бізнес логіку системи, нормалізацію даних, API для роботи з клієнтськими модулями.

Основний функціонал виконано на клієнтській частині. UX та UI додатку розроблено таким чином, що дозволяє використовувати повний функціонал системи з першої спроби. На стороні клієнту відбувається шифрування та розшифрування даних, а також обробка усієї інформації.

Усі дані у додатку зберігаються у базі даних для безпеки даних. Повідомлення користувачів зберігаються у зашифрованому вигляді, в той час як уся інша інформація

Бізнес-логіка реалізує бізнес-правила. А що таке бізнес-правило? Бізнес-правило – це положення, яке визначає чи обмежує будь-які сторони предметної області. Його призначення – захистити структуру предметної області, контролювати або впливати на його операції.

Нормалізація даних використовується для коректного подання та швидкісній обробці даних.

2.3 Розробка методів роботи системи

Для удосконалення методу шифрування з використанням асиметричного шифрування було вирішено розробити функцію для шифрування інформації, який у якості аргументів приймає не один ключ шифрування та повідомлення яке

необхідно зашифрувати, а два ключа шифрування та повідомлення яке необхідно зашифрувати.

2.3.1 Розробка методу асиметричного шифрування інформації користувача

Метод шифрування з використанням асиметричних ключів шифрування слугує для захисту інформаційної системи-месенджеру.

Шифрування у додатку використовується для обміну повідомленнями у системі.

Обмін повідомленнями у системі працює наступним чином:

1. Створюється копія відправленого повідомлення
2. Оригінальне повідомлення шифрується публічним ключем користувача, який є відправником повідомлення
3. Копія повідомлення шифрується публічним ключем користувача, який є отримувачем повідомлення
4. До бази даних записуються обидві версії зашифрованого повідомлення, для того щоб потім кожен з учасників діалогу між розшифрувати відправлене повідомлення власним приватним ключем шифрування.

Стандартний метод шифрування як аргумент приймає один публічний ключ та повідомлення, яке необхідно зашифрувати. Тобто, при використанні стандартного методу шифрування потрібно буде спочатку зашифрувати оригінал повідомлення, і тільки після цього ще раз викликати той самий метод, але у аргументи йому передати вже публічний ключ другого користувача. Це значить що шифрування повідомлення таким чином буде виконувати послідовно у 2 етапи, що при відправленні великих повідомлень може створювати враження, що система працює дуже повільно.

Саме тому було вирішено модифікувати стандартний метод шифрування повідомлень таким чином, щоб у якості аргументів він приймав не один ключ

шифрування інформації та повідомлення яке необхідно зашифрувати, а 2 ключа та повідомлення яке ними потрібно зашифрувати, після чого повертав 2 версії зашифрованого повідомлення різними ключами, а шифрування різними ключами виконував паралельно а не послідовно.

У таблиці 2.1 наведено порівняння шифрування повідомлення стандартним методом шифрування та модифікованим.

Таблиця 2.1 – порівняння шифрування повідомлення стандартним методом шифрування та модифікованим

Довжина повідомлення, яке необхідно зашифрувати	Час в мілісекундах який необхідний для шифрування повідомлення	
	Стандартний метод шифрування	Модифікований метод шифрування
15 символів	3.4	2.35
50 символів	10.2	7.07
300 символів	25.7	18
1000 символів	76.34	52.97
2000 символів	147.25	112.68

Як видно з наведеної таблиці, за допомогою використання модифікованого методу шифрування інформації досягається покращення швидкої на 30-35 відсотків, що доводить доцільність його використання.

2.4 Розробка методів шифрування та дешифрування повідомлень

Для шифрування інформації у месенджері використовується алгоритм RSA.

RSA (абревіатура від прізвищ Rivest, Shamir та Adleman) — криптографічний алгоритм з відкритим ключем, що ґрунтується на обчислювальній складності завдання факторизації великих цілих чисел[12].

Криптосистема RSA стала першою системою, придатною і для

шифрування, і цифрового підпису. Алгоритм використовується у великій кількості криптографічних програм, включаючи PGP, S/MIME, TLS/SSL, IPSEC/IKE та інших[12].

Криптографічні системи з відкритим ключем використовують так звані односторонні функції, які мають таку властивість:

- якщо відомо x, то розрахувати f(x) доволі просто;
- якщо відомо y = f(x), то для розрахування x немає ефективного шляху знаходження.

Під однобічністю розуміється не математично доведена односпрямованість, а практична неможливість обчислити зворотне значення, використовуючи сучасні обчислювальні засоби, за доступний для огляду інтервал часу[12].

В основу криптографічної системи з відкритим ключем RSA покладено складність завдання факторизації добутку двох великих простих чисел. Для шифрування використовується операція зведення в ступінь модуля великого числа. Для дешифрування (зворотної операції) за розумний час необхідно вміти обчислювати функцію Ейлера від великого числа, навіщо необхідно знати розкладання числа на прості множники[12].

У криптографічної системі з відкритим ключем кожен учасник має у своєму розпорядженні як відкритий ключ (англ. public key), так і закритий ключ (англ. private key). У криптографічної системі RSA кожен ключ складається з кількох чисел. Кожен учасник створює свій відкритий та закритий ключ самостійно. Закритий ключ кожен з них тримає в секреті, а відкриті ключі можна повідомляти будь-кому або навіть публікувати їх. Відкритий та закритий ключі кожного учасника обміну повідомленнями у криптосистемі RSA утворюють «узгоджену пару» [12].

RSA-ключі генеруються наступним чином:

1. Вибираються 2 випадкових простих числа p і q заданого розміру (наприклад 1024 біта кожне)
2. Розраховується їх добуток n = p * q, який називається їх модулем;

3. Розраховується значення їх функції Ейлера від числа n: f(n) = (p – 1) * (q – 1);
4. Вибирається ціле число e (1 < e < f(n)), взаємно просте зі значенням функції f(n):
    1. Число e називається відкритою експонентою;
    2. Зазвичай в якості числа e беруть прості числа, які містять у собі невелику кількість одиничних біт в двійковому записі, наприклад, прості числа з чисел Ферма: 17, 257, 65537, так як в такому випадку час, необхідний для шифрування з використанням швидкого піднесення до ступеню буде менше; в той час як використанням дуже малих чисел, наприклад 3 потенційно можуть послабити безпечність схеми RSA.
5. Розраховується число d, мультиплікативне числу e по модулю f(n). Число d називається секретною експонентою, зазвичай воно розраховується за допомогою розширеного алгоритму Евкліду.
6. Пара (e, n) публікується у якості відкритого ключа;
7. Пара (d, n) грає роль закритого ключа та тримається в секреті.

Для шифрування повідомлень у месенджері повідомлення отримуються у форматі тексту – те, що користувач безпосередньо вводить з клавіатури у додатку, після чого повідомлення розбивається на частини, так щоб отримати масив стрічок такої довжини, яку може зашифрувати алгоритм RSA. Через певні технічні обмеження використовуючи асиметричні алгоритми шифрування максимальний обсяг інформації, яку потрібно зашифрувати повинен бути меншим ніж сам публічний ключ. Саме тому, для того щоб не обмежувати користувача в довжині повідомлення було вирішено перед шифруванням ділити повідомлення на частини, які гарантовано можуть бути зашифровані алгоритмом, а в базі даних зберігати масив зашифрованих стрічок[12].

Для розшифрування повідомлень необхідно отримати масив зашифрованих стрічок, розшифрувати їх, тобто створити масив з розшифрованими частинами повідомлення, після чого усі частини масиву

об’єднати в один рядок, який і буде розшифрованим повідомленням[12].

2.5 Розробка діаграми класів месенджеру

У додатку використовується бібліотека React.

React (іноді React.js або ReactJS) — JavaScript-бібліотека з відкритим вихідним кодом для розробки інтерфейсів користувача.

React розробляється та підтримується Facebook, Instagram та спільнотою окремих розробників та корпорацій.

React може використовуватися для розробки односторінкових та мобільних додатків. Його мета — надати високу швидкість, простоту та масштабованість. Як бібліотека для розробки інтерфейсів користувача React часто використовується з іншими бібліотеками, такими як MobX, Redux і GraphQL.

В даному додатку React використовується разом з бібліотекою MobX.

MobX - бібліотека JavaScript з відкритим вихідним кодом, призначена для керування станом програми. Найчастіше використовується у зв'язці з React чи Angular для розробки клієнтської частини. Містить низку інструментів, що дозволяють значно спростити передачу даних сховища через контекст[13].

Будь який додаток на React складається з компонентів і має деревовидну структуру, тобто існує один батьківський компонент, в якого може бути безліч дочірніх компонентів, в кожного з яких може бути безліч дочірніх компонентів і так далі. В React у кожного з цих компонентів є 2 стани – внутрішній (state) та зовнішній (props). Зовнішній стан компонент отримує від свого батьківського компоненту, в той час як внутрішній стан задається всередині самого компоненту. Тобто в React досить просто передавати дані зверху вниз, тобто від батьківського компоненту до дочірнього, а передати дані напряму від дочірнього компоненту до батьківського можливо лише з використанням callback функцій, які також потрібно передати з батьківського компоненту в дочірній. Такий підхід є досить зручним, коли додаток є невеликим і вкладеність компонентів є невеликою, а коли у додатку багато інформації яку потрібно передавати з одного

компонента в інший і вкладеність компонентів є досить великою використання такого підходу стає недоцільним. Саме тому було вирішено використовувати бібліотеку MobX, яка дозволяє об’єднати стан усієї програми а не кожного компоненту окремо в один клас (сховище), що дає змогу керувати даними додатку з будь якого компоненту без написання додаткового коду та логіки програми для передавання даних знизу – вверх з використанням callback функцій. А також такий підхід дозволяє тримати усі дані в одному місці а не розкидано по всьому додатку.

Клас MainStore – основний клас програми який об’єднує у собі усі інші класи (сховища) а також має такі поля та методи:

- privateKey – приватний ключ користувача, поле типу String;
- publicKey – публічний ключ користувача, поле типу String;
- searchPeopleInput – поле, що містить дані, що користувач ввів для пошуку інших користувачів у системі поле типу String;
- foundedUsers – поле, що містить інформацію по користувачам, яких шукав користувач, поле типу Array;
- userObj – об’єкт, що містить інформацію про користувача, таку як:
    - id: ідентифікатор користувача у базі даних, поле типу ObjectID;
    - email: пошта користувача, поле типу String;
    - name: ім’я користувача, поле типу String;
    - lastName: прізвище користувача, поле типу String;
    - middleName: по-батькові користувача, поле типу String;
    - nickname: прізвисько користувача у системі, поле типу String;
    - avatar: посилання на фотографію профілю користувача, поле типу String;
    - role: роль користувача у системі, поле типу String;
    - onlineStatus: статус користувача у мережі, поле типу String;
    - status: статус користувача у системі, поле типу String;
    - profileDescription: опис профілю користувача, поле типу String;

    - frozenUntil: поле, яке вказує до якого числа профіль користувача заблокований, якщо його заблокували, поле типу String;
    - createdAt: дата створення профілю користувача, поле типу Date;
    - updatedAt: дата оновлення профілю користувача, поле типу Date;
- sendCodeStore – клас, що містить інформацію з екрану першого етапу авторизації, поле типу Store;
- loginStore – клас, що містить інформацію з екрану другого етапу авторизації, поле типу Store;
- registerStore – клас, що містить інформацію з екрану реєстрації в системі, поле типу Store;
- dialogPageStore – клас, що містить інформацію з екрану переписки з користувачем, поле типу Store;
- rooms – поле, що містить інформацію про діалоги користувача, поле типу Array;
- setFoundedUsers – метод для занесення до сховища інформації по знайденим юзерам через пошук;
- setRoomAndUserAndKeys – метод для занесення до сховища інформації про діалоги користувача, інформації про користувача а також ключів шифрування;
- setUser – метод для занесення до сховища інформації про юзера;
- updateAvatar – метод для оновлення фотографії профілю користувача;
- setPrivateKey – метод для задання приватного ключа користувача;
- setPublicKey – метод для задання публічного ключа користувача;
- changeSearchPeopleValue – метод для оновлення у сховищі інформації про те, що ввів користувач для пошуку інших користувачів;

- getRooms – метод для отримання інформації по діалогам користувача;
- setRooms – метод для занесення до сховища інформації про діалоги користувача;
- updateRooms – метод для оновлення інформації по діалогам користувачів

Діаграма класу MainClass наведена на рисунку 2.1.

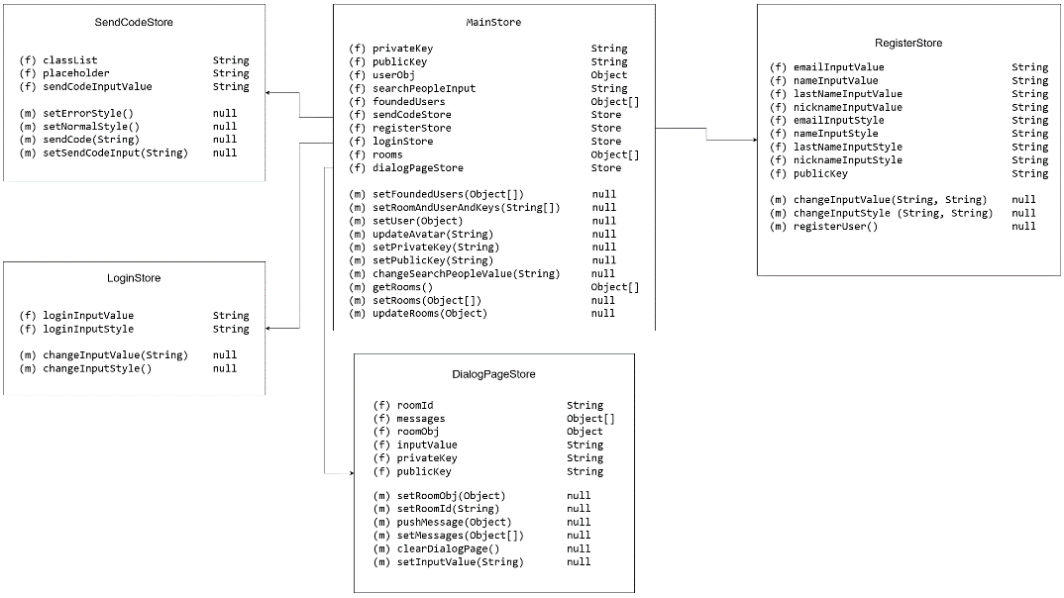


Рисунок 2.1 – діаграма класу MainClass.

Клас SendCodeStore слугує для обробки та зберігання інформації на екрані першого етапу авторизації і містить такі поля та методи:

- classList – поле для зберігання поточного стану стилізації полів форми, поле типу String;
- placeholder – поле для зберігання інформації, яка буде відображатись у підказках до поля, поле типу String;
- sendCodeInputValue – поле для зберігання інформації про те, що ввів у полі користувач, поле типу String;
- setErrorStyle – метод для видачі помилки у додатку;
- setNormalStyle – метод для повернення до нормального стану додатку;

– sendCode – метод для відправлення користувачу коду для авторизації на пошту;
– setSendCodeInput – метод для оновленні у сховищі інформації про те, що ввів користувач;

Діаграма класу SendCodeStore наведена на рисунку 2.2.

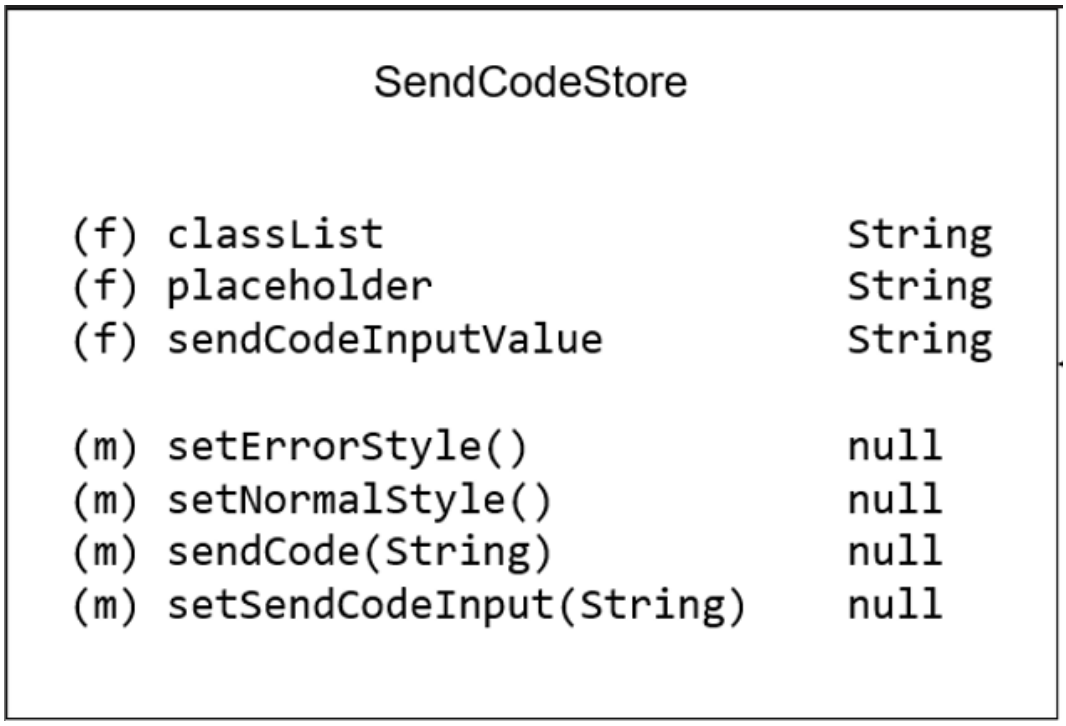


Рисунок 2.2 – діаграма класу SendCodeStore.

Клас LoginStore слугує для обробки та зберігання інформації на екрані другого етапу авторизації і містить такі поля та методи:

– loginInputValue – поле для зберігання інформації по тому, шо користувач ввів у поле, поле типу String;
– loginInputStyle – поле що зберігає інформацію про стилізацію поля вводу коду для логіну, поле типу String;
– changeInputValue – метод для оновлення у сховищі інформації про те, що ввів користувач;
– changeInputStyle – метод для виведення користувачу помилки, якщо користувач ввів некоректні дані;

Діаграма класу LoginStore наведена на рисунку 2.3.

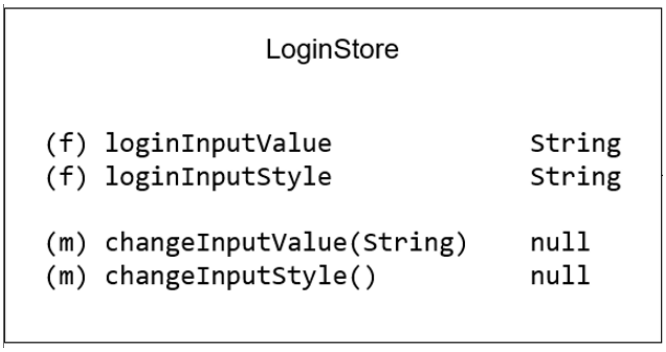


Рисунок 2.3 – діаграма класу LoginStore.

Клас DialogPageStore слугує для обробки та зберігання інформації на екрані переписки з іншим користувачем і містить такі поля та методи:

- roomId – поле, що зберігає інформацію про ідентифікатор діалогу, в якому знаходиться користувач, поле типу String;
- messages – поле, що зберігає масив повідомлень, що були відправлені у діалозі, поле типу Array;
- roomObj – поле з інформацією про відкритий діалог, поле типу Object;
- inputValue – поле з інформацією, про те, що ввів користувач, поле типу String;
- privateKey – поле з інформацією про приватний ключ користувача, поле типу String;
- publicKey – поле з інформацією про публічний ключ користувача, поле типу String;
- setRoomObj – метод для задання об’єкту з інформацією про відкритий діалог;
- setRoomId – метод для занесення до об’єкту інформації про ідентифікатор відкритого діалогу;
- pushMessage – метод для додавання нового повідомлення до сховища;
- setMessages – метод для додавання до сховища інформації про написані у діалозі повідомлення;

– clearDialogPage – метод для очистки інформації про кімнату;
– setInputValue – метод для оновлення у сховищі даних про те, що ввів користувач;

Діаграма класу DialogPageStore наведена на рисунку 2.4.

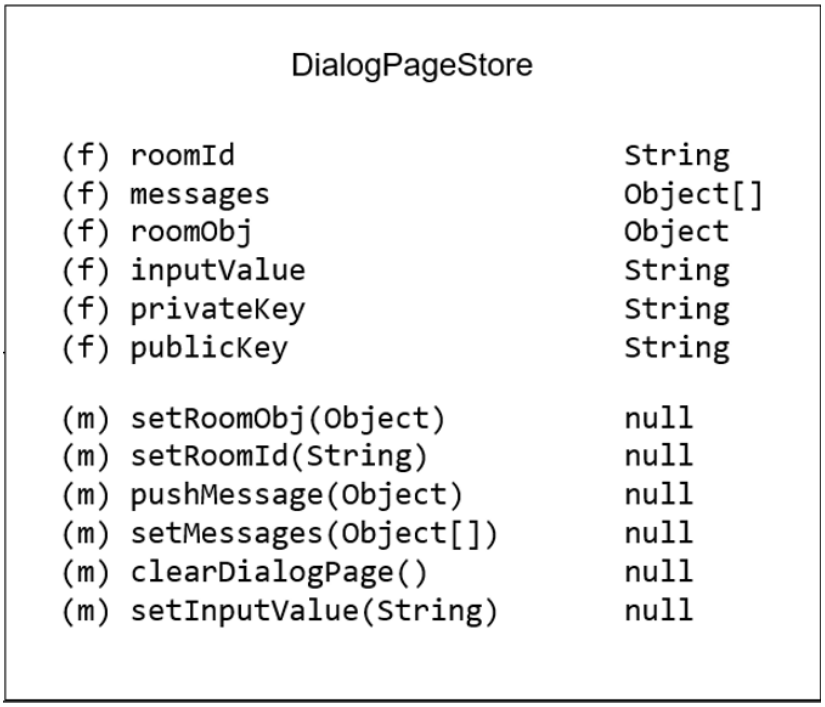


Рисунок 2.4 – діаграма класу DialogPageStore.

Клас RegisterStore слугує для обробки та зберігання інформації на екрані реєстрації користувача у системі і містить такі поля та методи:

– emailInputValue – поле, що містить інформацію про те, що користувач ввів у поле вводу електронної пошти, поле типу String;
– nameInputValue - поле, що містить інформацію про те, що користувач ввів у поле вводу імені, поле типу String;
– lastNameInputValue - поле, що містить інформацію про те, що користувач ввів у поле вводу прізвища, поле типу String;
– nicknameInputValue - поле, що містить інформацію про те, що користувач ввів у поле вводу прізвиська, поле типу String;
– emailInputStyle – поле, в якому міститься інформація про стилізацію поля вводу електронної пошти, поле типу String;

- nameInputStyle - поле, в якому міститься інформація про стилізацію поля вводу імені користувача, поле типу String;
- lastNameInputStyle - поле, в якому міститься інформація про стилізацію поля вводу прізвища, поле типу String;
- nicknameInputStyle - поле, в якому міститься інформація про стилізацію поля вводу прізвиська, поле типу String;
- publicKey – поле, в якому міститься приватний ключ користувача;
- changeInputValue – метод для оновлення інформації про те, що користувач вводив у поля форми;
- changeInputStyle – метод для зміни стилізації полів форми;
- registerUser – метод для реєстрації користувача у системі;

Діаграма класу RegisterStore наведена на рисунку 2.5.

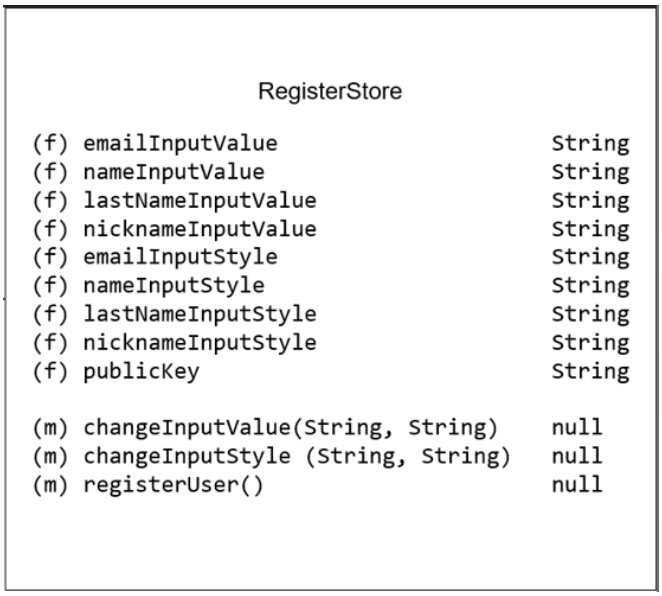


Рисунок 2.5 – діаграма класу RegisterStore

2.6 Розробка алгоритмів

Основними алгоритмами є алгоритм шифрування повідомлення та розшифрування повідомлення.

На рисунку 2.6 представлена блок-схема алгоритму методу шифрування повідомлення, який викликається коли користувач відправляє повідомлення

другому користувачу.

Даний алгоритм отримує повідомлення користувача як параметр, розбиває його на масив типу String довжиною по 50 символів, по кожному елементу масиву зашифрувати його, використовуючи публічний ключ користувача та повернути масив зашифрованих

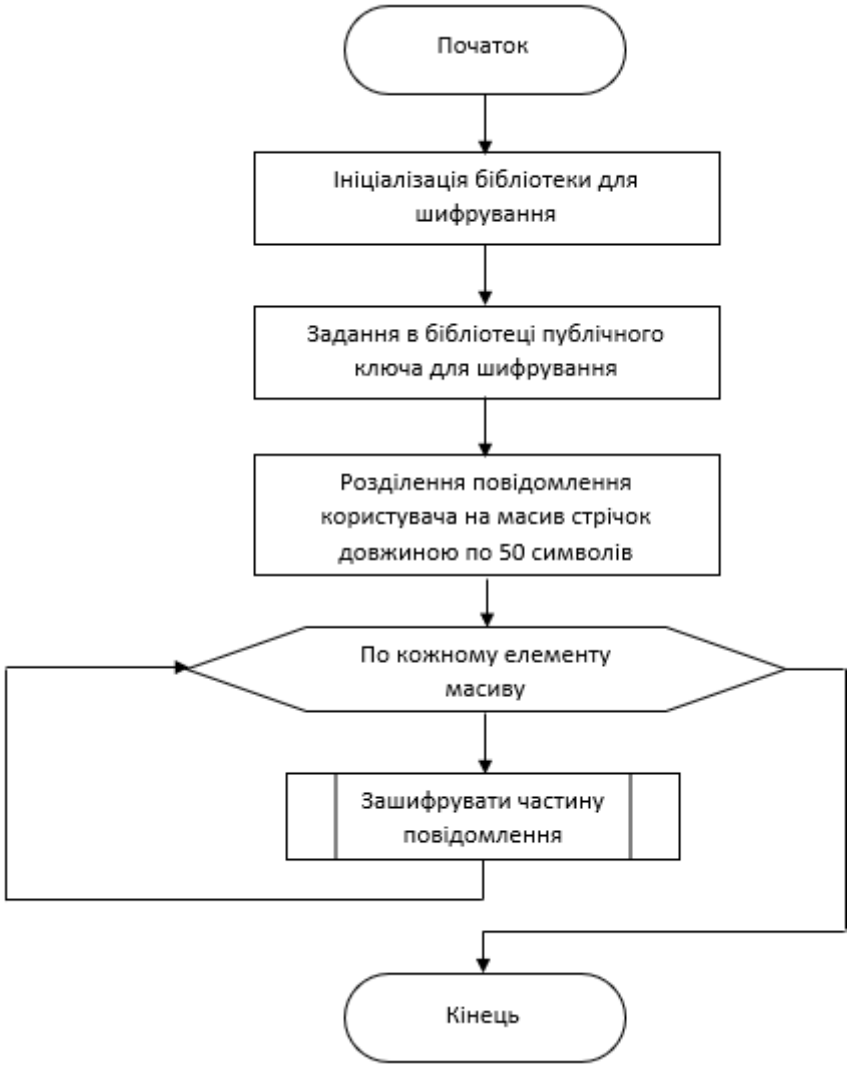


Рисунок 2.6 – блок-схема алгоритму методу шифрування повідомлення

На рисунку 2.7 представлена блок схема алгоритму методу дешифрування повідомлень, який викликається коли користувач завантажує повідомлення з кімнати.

Даний алгоритм отримує як аргумент повідомлення як масив зашифрованих стрічок, розшифровує кожен з них за допомогою приватного

ключа після чого об’єднує всі елементи масиву в розшифроване повідомлення.

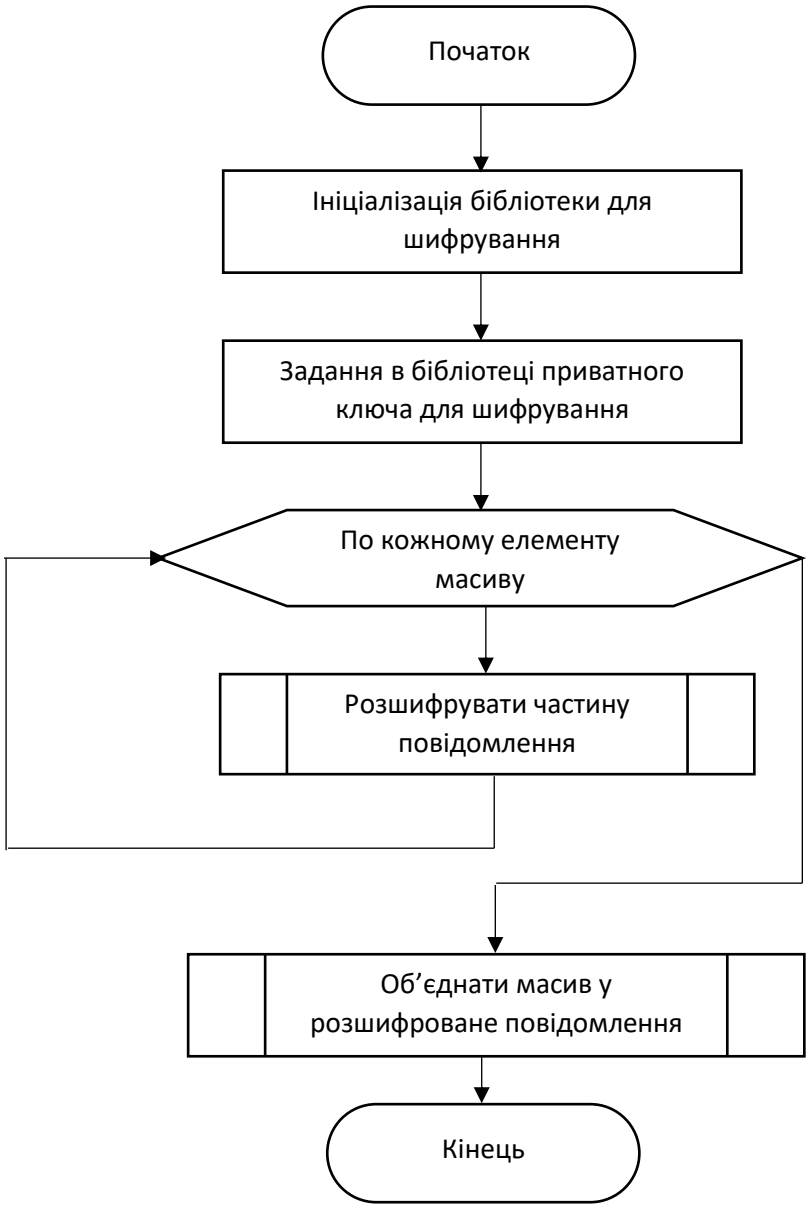


Рисунок 2.7 – блок схема алгоритму розшифрування повідомлень

2.7 Висновки

В даному розділі було проаналізовано вхідні та вихідні дані, розроблено діаграми класів: RegisterStore, LoginStore, DialogPageStore, SendCodeStore, MainClass.Також алгоритм шифрування повідомлення та розшифрування повідомлень.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИНИ МЕСЕНДЖЕРУ

3.1 Варіантний аналіз і обґрунтування вибору засобу реалізації програмного засобу

У якості варіантів мов програмування для розробки серверної частини було обрано PHP, C# та його платформа веб-додатків ASP.NET, JavaScript та програмну платформу NodeJS.

PHP – скриптова мова загального призначення, що інтенсивно застосовується для розробки веб-додатків. В даний час підтримується більшістю хостинг-провайдерів і є однією із лідерів серед мов програмування, що використовується для створення динамічних веб-сайтів[15].

Мова та його інтерпретатор (Zend Engine) розробляються групою ентузіастів у рамках проекту з відкритим кодом. Проект розповсюджується під власною ліцензією, несумісною з GNU GPL[15].

Переваги мови PHP:

- Потужний та гнучкий. Як ми вже писали, ця мова здатна обслужити невеликий особистий блог, але при цьому спокійно почувається у великих ресурсах: інтернет-магазинах, соцмережах, порталах і т.д.
- Безкоштовність. Ця мова є повністю безкоштовною та поширюється з вільною ліцензією, тому її сміливо можуть застосовувати як приватні особи, так і комерційні організації.
- Має велику спільноту розробників, а отже просто знайти необхідну інформацію.

Але також є і недоліки, серед яких:

- на ньому неможливо створити десктопну програму або якийсь системний компонент;
- у додатків на PHP нижча захищеність, ніж із використанням інших мов;

– слабка можливість роботи із глобальними винятками.

JavaScript - мультипарадигменна мова програмування. Підтримує об'єктно-орієнтований, імперативний та функціональний стилі. Є реалізацією специфікації ECMAScript[16].

JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найширше застосування знаходить у браузерах як мову сценаріїв для надання інтерактивності веб-сторінок[16].

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу[16].

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожою на Java. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє її від низки мов програмування, які використовуються у веб-розробці[16].

Назва JavaScript є зареєстрованим товарним знаком корпорації Oracle в США[16].

Node або Node.js - програмна платформа, заснована на двигуні V8 (що транслює JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API, написаний на C++, підключати інші зовнішні бібліотеки, написані різними мовами, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js та десктопні віконні програми (за допомогою NW.js, AppJS або Electron для Linux, Windows та macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js та espruino). В основі Node.js лежить подієво-орієнтоване та асинхронне (або реактивне) програмування з неблокуючим введенням/виводом[17].

Переваги платформи NodeJS:

– Повна нативна підтримка JSON формату;

- Має сотні тисяч бібліотек від сторонніх розробників, в тому числі від великих компаній, таких як Google та Facebook;
- Найбільша швидкість обробки запитів серед конкурентів;
- Дозволяє використовувати модулі написані на C++;

Недоліки платформи NodeJS:

- Низька продуктивність при роботі з важкими для процесора задачами

Ruby - динамічна, рефлективна, інтерпретована високорівнева мова програмування. Мова володіє незалежною від операційної системи реалізацією багатопоточності, сильною динамічною типізацією, збирачем сміття та багатьма іншими можливостями. За особливостями синтаксису він близький до мов Perl та Eiffel, за об'єктно-орієнтованим підходом – до Smalltalk. Також деякі риси мови взяті з Python, Lisp, Dylan та Клу[18].

Кросплатформна реалізація інтерпретатора мови є повністю вільною.

Переваги мови Ruby:

- високорівневий — зручний для розробників, із сильною абстракцією та використанням конструкцій натуральної чи людської мови;
- динамічний - типи даних визначаються на етапі виконання програми, це збільшує швидкість розробки;
- інтерпретований - код на Ruby обробляється інтерпретатором у момент виконання без попередньої компіляції, це забезпечує незалежність від платформ і зменшує розмір програм, що виконуються;
- мова програмування загального призначення — на відміну предметно-ориентированных, мови програмування загального призначення не створюються до застосування у специфічних областях.

Недоліки мови Ruby:

- маю низьку продуктивність в порівнянні з конкурентами;

– не призначений для розробки надійних рішень корпоративних рішень

Як видно з порівняння, найбільше переваг і найменше серйозних недоліків в порівнянні з конкурентами має JavaScript та програмна платформа NodeJS, так як має безліч сторонніх бібліотек, високу швидкість та надійність а також дозволяє використовувати модулі написані на мові C++.

У якості мови програмування для розробки клієнтської частини також було обрано JavaScript та програмну платформу NodeJS з фреймворком Electron для створення десктопних програмних додатків.

Electron.js — це фреймворк, який дозволяє користувачеві створювати настільні програми з HTML5, CSS і JavaScript. Це проект з відкритим кодом, започаткований Ченгом Чжао, інженером GitHub[19].

Будь-яка веб-програма, яку ви написали, може працювати на Electron.js. Аналогічно, будь-яка програма Node.JS, яку ви пишете, може використовувати цю технологію[19].

Electron JS використовує веб-технології, такі як простий HTML, CSS і JavaScript.Крім цих технологій більше нічого знати не потрібно, щоб написати додаток на Electron. Він може бути розроблений для одного браузера. Його файлова система належить до API Node.js і працює в Linux, Mac OS X, Windows[19].

Він використовує модулі npm, які широко використовуються для JavaScript. Він складається з рідного меню для діалогових вікон і сповіщень. Інсталятори Windows не потребують ніякої конфігурації[19].

Він також має можливість автоматичного оновлення та звітів про аварійне завершення роботи на Windows і Mac за допомогою Squirrel. Звіти про аварійне завершення роботи надсилаються на віддалений сервер для подальшого аналізу. Chromium контролює такі дії з відстеження вмісту, як налагодження та профілювання[19].

Electron базується на Chromium.

Chromium - веб-браузер з відкритим вихідним кодом, розробляється

спільнотою The Chromium Authors, компанією Google і некоторими іншими компаніями (Opera Software, Яндекс, NVIDIA, Microsoft і іншими). За підтвердженням розробників, Chromium призначений для надання користувача швидкого, безпечного та надтожного доступу в Інтернет, а також для зручної платформи для веб-прикладів. На основі Chromium створений браузер Google Chrome (при цьому в рамках Chromium також доступні попередні попередні версії, які можна ознайомитися з новими, ще не включеними в склад Chrome), а також ряд інших альтернативних веб-переглядачів[20].

Згідно слів розробників Chromium вони намагаються створити найбезпечний браузер, розробники приділяють велику увагу з впровадження нових функцій для захисту браузера. Для забезпечення безпеки в Chromium була обрана модель «песочниці», що дозволить обмежити простір для атаки користувацького комп'ютера через використану вразливість. Дослідники Google прийшли до висновку, що майже 70% загроз «працюють» у рухомих зображеннях, які взаємодіють з ненадійним вмістом. Саме тому розробники перевели всю роботу движка у «пісочницю». У більшості операційних систем Linux цей режим у браузері увімкнений, однак деякі неофіційні зборки дистрибутива Chromium Slackware відключають режим «пісочниці» спеціально. Тем не менш на офіційно підтримуваних системах Google Linux, починаючи з версії 23, Chromium використовує можливості ядра для використання додаткових компонентів, таких як фільтри seccomp-bpf, що дозволяє значно обмежити можливість використання зловмисниками специфічних викликів ядра. У збірці 66022 розробники перенесли в «пісочницю» (визначення стосується системи під ОС Microsoft Windows) також виконується підключаючий модуль Adobe Flash Player, який досить часто стає об'єктом пристальної уваги хакерів. У Chromium немає дієвого захисту від XSS-атак, але, завдяки тому, що Chromium підтримує файли cookie, призначені лише для HTTP, небезпека міжсайтового скриптингу значно знижується. Також активно тестується функція XSS Auditor, впроваджена в список експериментальних функцій у 7 версії. Даний компонент значно збільшує захист від міжсайтового скриптингу. Перший XSS Auditor був

використаний у 4-й версії Chromium, а також у зв’язку зі численними помилками та падінням функції у версії 4.1. Проблему з продуктивністю та стабільністю розробників вдалося вирішити, але функція до цього часу є експериментальною, так як не всі сайти, здатні з нею працювати. Також у 7 версії Chromium в якості експерименту з'явилася можливість нагляду за підключеними модулями. Браузер отримав можливість запропонувати відключення технічних плагінів, які мають незакриті вразливості до тих пір, поки не вийде оновлена версія модуля з помилками керування, остаточно функція стала доступною в 10 версії браузера. Для забезпечення криптографічної безпеки при роботі з конфіденційною інформацією користувачів Chromium надає можливість працювати із надійним протоколом передачі даних (HTTPS), які можуть бути упаковані відповідно до криптографічних протоколів SSL 3.0 і TLS 1.0. Для додаткової захисту Chromium може використовувати експериментальний відкритий протокол HSTS, що дозволяє встановити з сайтами форсований режим захисту з’єднання. Час набудови безпеки користувач може регулювати сам. У Chromium вбудований компонент Безпечний засіб, що забезпечує захист від фішинга та шкідливого ПО. Chromium при першому запуску протягом перших п’ятих хвилин завантажує базу визначення шкідливих і шахрайських сайтів, які потім оновлюються через 30 хвилин, при цьому ніяка особиста інформація в компанії Google не повертається. Сам компонент може бути відключений в налаштуваннях браузера. Додатково Chromium дає змогу виконувати гнучке налаштування вмісту веб-сторінок: редагувати політику запуску плагінів, використовувати JavaScript та файли cookie, а також очищати конфіденційні дані за визначений період. Середи інших механізмів захисту можна виділити:

- Специфікація HTML5 у вигляді заголовка Origin забезпечує захист від розділів міжсайтових запитів (CSRF), блокуючи неправильні запити сайтів.
- Chromium підтримує X-Frame-Options API, що захищає браузер від клікджекінгу, а також X-Content-Type-Options API, що надає браузеру можливість захищатися від MIME-сніффінга.

– Chromium підтримує «режим інкогніто», при якому історія наявного та завантаження не записується в журнали, а файли cookies видаляються після закриття браузера[20].

При цьому Google закликає користувачів при виявленні вразливостей в браузері повідомляти про них розробникам, взамін отримуючи грошову винагороду[20].

Серед варіантів середовища розробки були Atom, JetBrains WebStorm, Visual Studio.

Microsoft Visual Studio — лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення та інших інструментів. Дані продукти дозволяють розробляти як консольні програми, так і ігри та програми з графічним інтерфейсом, у тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому коді для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework та Silverlight[21].

Visual Studio включає редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторингу коду. Вбудований налагоджувач може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Інші вбудовані інструменти включають редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати та підключати сторонні доповнення (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як, наприклад, Subversion та Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування та візуального проектування) коду предметно-орієнтованими мовами програмування) або інструментами для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server) [21].

Переваги Visual Studio:

- безкоштовність у версії Community Edition;
- велика спільнота розробників.

Недоліки Visual Studio:

- відсутня повна підтримка інтелектуального підсвічування синтаксису для JavaScript
- відсутність автоімпорту сторонніх бібліотек для JavaScript;

Atom – безкоштовний текстовий редактор з відкритим вихідним кодом для Linux, macOS, Windows з підтримкою плагінів, написаних на JavaScript, та вбудованих під керуванням Git. Більшість плагінів мають статус вільного програмного забезпечення, розробляються та підтримуються спільнотою[22].

Atom заснований на Electron - фреймворку крос-платформної розробки з використанням Chromium та io.js. Редактор написаний на CoffeeScript та LESS. Версія 1.0 була випущена 25 червня 2015 року[22].

Переваги Atom:

- Безкоштовність;
- Кросплатформеність;
- Зручний пошук файлів у відкритому проекті;

Недоліки Atom:

- Повільна швидкість роботи;
- Низька стабільність роботи;
- відсутність автоімпорту сторонніх бібліотек для JavaScript;
- Мала відносно конкурентів кількість сторонніх плагінів для додатку.

JetBrains WebStorm — інтегроване середовище розробки JavaScript, CSS & HTML від компанії JetBrains, розроблене на основі платформи IntelliJ IDEA[23].

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію за кодом, рефакторинг, налагодження та інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі рефакторинг коду JavaScript, що знаходиться в різних файлах та папках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладено скрипт на Javascript,

в який вкладено інший код HTML, всередині якого вкладений JavaScript) - тобто в таких конструкціях підтримується коректний рефакторинг[23].

Переваги WebStorm:

- Автоімпорт бібліотек;
- Зручний пошук по файлам усього проекту;
- Зручна робота з системою контролю версій;
- Можливість працювати з командою безпосередньо через IDE;
- Велика кількість плагінів від сторонніх розробників, як платних так і безкоштовних;

Недоліки WebStorm:

- Безкоштовна версія лише для студентів;
- Великі відносно конкурентів системні вимоги для стабільної роботи;

Як видно із порівняння найкращим середовищем розробки для розробки на JavaScript є JetBrains WebStorm, так як головними недоліками його є безкоштовна версія лише для студентів а також високі системні вимоги відносно конкурентів, але водночас він пропонує автоімпорт бібліотек, зручний пошук по файлам а також зручну роботу з системою контролю версій.

Висновок. В даному розділі було визначено що мовою програмування для клієнтської та серверної частини було обрано JavaScript а також програмну платформу NodeJS, через її швидкість роботи, велику кількість бібліотек від сторонніх розробників та можливість використання модулів написаних на C++. В якості середовища розробки було обрано WebStorm через автоімпорт бібліотек, зручний пошук по файлам а також зручну роботу з системою контролю версій, які відсутні у представлених конкурентів.

3.2 Розробка загальної моделі роботи програми

Додаток розділений на клієнтську та серверну частину. Для отримання даних з серверу та запису нових даних до бази даних використовуються HTTP-запити а також сокет з використанням бібліотеки socket.io для серверної частини

та socket.io-client для клієнтської частини.

Після запуску клієнтської частини додатку користувач потрапляє на екран логіну, з якого можна зайти в свій профіль або зареєструватись, якщо у користувача ще немає профілю. Коли користувач увійшов до свого профілю він може отримувати повідомлення від інших користувачів або сам надсилати їх комусь.

Отже, спочатку потрібно увійти до свого аккаунту, після чого отримати дані про аккаунт користувача, а також отримати список його діалогів, перейти в будь-який чат щоб надіслати або отримати повідомлення.

На рисунку 3.1 представлена діаграма послідовності загальної роботи програми.

Рисунок 3.1 – Діаграма послідовності загальної моделі роботи програми

3.3 Розробка модулю створення зашифрованих повідомлень

Для того щоб реалізувати шифрування повідомлення спочатку потрібно отримати введені користувачем дані. Для цього було створено React-компонент

DialogBottom з полем для вводу та кнопкою «Надіслати повідомлення». Код компоненту DialogBottom наведено на рисунку 3.2.

```
const DialogBottom = observer(({store}) => {
    return <div className='dialog-bottom'>
        <input
            className="message-input"
            type="text"
            onChange={(e) => {
                store.setInputValue(e.target.value);
            }}
            value={store.inputValue}
            placeholder="Input your message"
            onKeyDown={
                async event => {
                    if (event.key === 'Enter') {
                        await sendMessage(store);
                    }
                }
            }
        />

        <div
            className="send-message-btn"
            onClick={
                async () => {
                    await sendMessage(store);
                }
            }
        >
            <img src="./build/assets/arrowforward.png" alt=""/>
        </div>
    </div>
});
```

Рисунок 3.2 – Код компоненту DialogBottom

Повідомлення шифрується коли користувач натискає на кнопку відправлення повідомлення або коли поле вводу знаходиться в фокусі і користувач натискає на клавішу Enter. В обох випадках викликається функція sendMessage, код якої наведено на рисунку 3.3.

```
async function sendMessage(store) {
    if (store.inputValue.length > 0) {
        let messageCreateObj = {
            room: store.roomId,
            messageObj: {}
        };

        for (let user of store.roomObj.users) {
            messageCreateObj.messageObj[user.id] = {message: cryptMessage(store.inputValue, user.publicKey)};
        }

        let [createdMessage, createdMessageError] = await createMessageApi(messageCreateObj);

        if (createdMessageError) history.push('/error');

        // store.pushMessage(createdMessage.data);
        store.setInputValue("");
    }
}
```

Рисунок 3.3 – Код функції sendMessage

У функції sendMessage для кожного користувача в діалозі викликається функція cryptMessage, параметрами якої є введене користувачем повідомлення, яке він збирається надіслати, та публічний ключ користувача, яким повідомлення і буде зашифроване. Код функції cryptMessage наведено на рисунку 3.4.

```
export function cryptMessage(message, publicKey) {
    let crypt = new JsEncrypt();

    crypt.setPublicKey(publicKey);

    let Parts = message.match(/.{1,50}/g),
        encryptedArr = [];

    for (let el of Parts) {
        encryptedArr.push(crypt.encrypt(el));
    }

    return encryptedArr;
}
```

Рисунок 3.4 – Код функції cryptMessage

В функції cryptMessage спочатку ініціалізується бібліотека для шифрування/дешифрування. Потім в бібліотеці задається публічний ключ користувача та повідомлення розбивається на рівні частини по 50 символів, після чого кожен елемент масиву шифрується публічним ключем користувача.

3.4 Розробка модулю розшифрування повідомлень

.

У програмі розшифровані повідомлення виводять на двох екранах: на екрані виводу діалогів та на екрані відкритого діалогу. На екрані діалогів повідомлення виводиться у компоненті RoomObj. Код компоненту RoomObj наведено на рисунку 3.5.

```
export default function RoomObj({room}) {
    let history = useHistory();
    let imageStyleObj = {
        backgroundImage:
            room.otherUser.avatar ? "url(" + process.env.BACKEND_URL + "/" + room.otherUser.avatar + ")"
            :
            "url('./build/assets/defaultAvatar.png')"
    }
    return (
        <div
            className="roomClass"
            onClick={
                () => {
                    socket.emit('joinRoom', `${room.id}${localStorage.getItem('id')}`);
                    history.push(`/room/${room.id}`)
                }
            }
        >
            <div className="roomClass-avatar" style={imageStyleObj}>
            </div>
            <div className="roomClass-info">
                <p className="room-user-name">{room.otherUser.name + " " + room.otherUser.lastName}</p>
                <p className="room-last-message">{room.lastMessage?.messageObj.message}</p>
                <p className="room-last-message-time">{room.lastMessage?.createdAt}</p>
            </div>
            <div className="clear">
            </div>
        </div>
    )
}
```

Рисунок 3.5 – Код компоненту RoomObj

В компоненті RoomObj відсутня будь яка логіка, відбувається лише рендер інформації, яка в нього передається. Отримання інформації з бази даних та розшифрування інформації відбувається у компоненті MainView у методі компоненту useEffect, який викликається при першому рендері компоненту. Код методу useEffect компоненту MainView наведено на рисунку 3.6.

```
useEffect(async () => {
    console.log('effect');
    let [user, userError] = await getUserDataApi(localStorage.getItem('id'));

    socket.off('new-message');
    socket.off('new-room-message');

    socket.on('new-message', (data) => {
        data = JSON.parse(data);
        let updatedRooms = [];
        let neededRoom = {};
        for (let i = 0; i < store.rooms.length; i++) {
            let obj = JSON.parse(JSON.stringify(store.rooms[i]));
            if (obj.id === data.room) {
                obj.lastMessage = data;
                neededRoom = store.decryptOneRoom(obj);
            } else {
                updatedRooms.push(obj);
            }
        }

        updatedRooms = [neededRoom, ...updatedRooms];

        store.setRoomsWithoutDecryption(updatedRooms);
    })

        store.setRoomsWithoutDecryption(updatedRooms);
    })

    socket.on('new-room', (data) => {
        store.updateRooms(data);
    })

    if (userError) {
        switch (userError.status) {
            case 404:
                history.push('/not-found');
                break;
            default:
                history.push('/error');
                break;
        }
    }

    let [rooms, error] = await getRoomsApi();

    if (error) {
        switch (error.status) {
            case 404:
                history.push('/not-found');
                break;
            default:
                history.push('/not-found');
                break;
            default:
                history.push('/error');
                break;
        }
    }

    let privateKey = await Axios.get('./private.pem');

    store.setRoomAndUserAndKeys(user.data, privateKey.data, user.data.publicKey, rooms.data.data);

    crypt.setPublicKey(store.publicKey);
    crypt.setPrivateKey(store.privateKey);
}, []);
```

Рисунок 3.6 – Код методу useEffect компоненту MainView

Для розшифрування повідомлень викликається метод setRooms, який викликає функцію decryptMessagesFromRoomObj. Ця функція отримує як параметр масив об’єктів кімнат та приватний ключ користувача. Код функції

decryptMessagesFromRoomObj наведено на рисунку 3.7.

```
export function decryptMessagesFromRoomObj(roomArr, privateKey) {
    let crypt = new JsEncrypt();

    crypt.setPrivateKey(privateKey);

    for (let i = 0; i < roomArr.length; i++) {
        if (!roomArr[i].lastMessage) continue;
        let decryptedArr = [];

        for (let elMes of roomArr[i].lastMessage.messageObj.message) {
            decryptedArr.push(crypt.decrypt(elMes));
        }

        roomArr[i].lastMessage.messageObj.message = messageShortener(decryptedArr.join(''));
    }

    return roomArr;
}
```

Рисунок 3.7 – Код функції decryptMessagesFromRoomObj

В даній функції спочатку ініціалізується бібліотека для шифрування та розшифрування, після чого в бібліотеці задається приватний ключ користувача. Після цього в циклі по кожній кімнаті отримується об’єкт повідомлення і розшифровується.

На екрані відкритого діалогу повідомлення виводяться в компоненті DialogMessageBox. Код компоненту DialogMessageBox наведено на рисунку 3.8.

```
const DialogMessageBox = observer(({store}) => {
    let messagesEnd = React.useRef < HTMLInputElement > null;

    useEffect(() => {
        messagesEnd.scrollIntoView({behavior: "auto"})
    });

    return <div className="message-box">
        {
            store.messages.map(el => {
                return <MessageComp messageObj={el} key={el.id}/>
            })
        }

        <div style={{float: "left", clear: "both"}}
             ref={(el) => {
                 messagesEnd = el;
             }}>
        </div>
    </div>
});
```

Рисунок 3.8 – код компоненту DialogMessageBox

В компоненті по кожному повідомленню з бази даних створюється компонент MessageComp. Код компоненту MessageComp наведено на рисунку 3.9.

```
export default function MessageComp({messageObj}) {
    let myMessage = messageObj.sender === localStorage.getItem('id') ? 'my-message' : 'inner-message';
    return <div className={myMessage}>
        <p className="message">
            {messageObj.messageObj.message}
        </p>
        <p className="message-time">
            {messageObj.createdAt}
        </p>
    </div>
}
```

Рисунок 3.9 – код компоненту MessageBox

Даний компонент не містить ніякої логіки, а слугує лише для відображення повідомлення у відформатованому форматі.

Отримуються з бази даних та розшифровуються повідомлення у компоненті DialogPage при рендері та викликі функції useEffect.Код функції useEffect компоненту DialogPage наведено на рисунку 3.10.

```
useEffect(async () => {
    socket.on('new-room-message', (data) => {
        data = JSON.parse(data);
        store.pushMessage(data);
    });

    let [apiDetail, apiDetailError] = await getRoomDetailApi(id);

    if (apiDetailError) {
        switch (apiDetailError.status) {
            case 404:
                history.push('/not-found');
                break;
            default:
                history.push('/error');
                break;
        }
    }

    store.setRoomId(id);

    store.setRoomObj(apiDetail.data);

    let [messages, messagesError] = await getMessagesApi(id);

    if (messagesError) history.push('/error');

    store.setMessages(messages.data);

    function listener(event) {
        if (event.keyCode === 27) {
            socket.emit('leaveRoom', `${store.roomObj?.id}${localStorage.getItem('id')}`);
            store.clearDialogPage();
            history.goBack();
            document.removeEventListener('keydown', listener);
        }
    }

    document.addEventListener('keydown', listener);
}, []);
```

Рисунок 3.10 – код функції useEffect компоненту DialogPage

У методі useEffect компоненту DialogPage для задання до компоненту та розшифрування повідомлень використовується метод setMessages який для розшифрування повідомлень викликає функцію decryptMessages, яка як параметри отримує масив повідомлень та приватний ключ користувача. Код функції decryptMessages наведено на рисунку 3.11.

```
export function decryptMessages(messageArr, privateKey) {
    let crypt = new JsEncrypt();

    crypt.setPrivateKey(privateKey);

    for (let i = 0; i < messageArr.length; i++) {
        let decryptedArr = [];

        for (let elMes of messageArr[i].messageObj.message) {
            decryptedArr.push(crypt.decrypt(elMes));
        }

        messageArr[i].messageObj.message = decryptedArr.join('');
    }

    return messageArr;
}
```

Рисунок 3.11 – код функції decryptMessages

Спочатку у функції decryptMessages ініціалізується бібліотека для шифрування/дешифрування, після цього у бібліотеці ініціалізується приватний ключ користувача. Потім по кожному повідомленню в массиві повідомлень в циклі розшифровується повідомлення, яке розбите на частини, після чого розшифровані частини повідомлення об’єднуються в одну стрічку.

3.5 Розробка модулю реєстрації

Для розробки даного модулю було створено компонент Register а також MobX сховище RegisterStore для того щоб зберігати інформації введену

користувачем. Код компоненту Register наведено на рисунку 3.12.

```
const Register = observer(({store}) => {
    let backImageStyleObj = {
        backgroundImage: `url('./build/assets/defaultAvatar.png')`
    }
    let history = useHistory();

    return (
        <div className={'absolute-block'}>
            <Popup behaviour={() => { history.push('/login') }}
                info={'Your private and public key were generated successfully, you can find them in root directory of application'}
                display={store.showPopup} />
            <div className="block-with-inputs">
                <div className="register-image" style={backImageStyleObj}>
                </div>
                <input type="text" className={`${store.emailInputStyle} email-reg`} value={store.emailInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'emailInputValue')
                }}
                        placeholder={"Enter your email"}
                />
                <input type="text" className={`${store.nameInputStyle} name-reg`} value={store.nameInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'nameInputValue')
                }}
                        placeholder={"Enter your name"}
                />
                <input type="text" className={`${store.lastNameInputStyle} last-name-reg`} value={store.lastNameInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'lastNameInputValue')
                }}
                        placeholder={"Enter your lastName"}
                />
                <input type="text" className={`${store.nicknameInputStyle} nickname-reg`} value={store.nicknameInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'nicknameInputValue')
                }}
                        placeholder={"Enter your nickname"}
                />

                <button ...>Register
                </button>

                <p>Or you can authorize by sending code on your email if you already have account using <Link
                    to="/send-code" className="link">this</Link> link</p>
            </div>
        </div>
    )
})
```

Рисунок 3.12 – Код компоненту Register

При натисненні на кнопку викликається функція registerBtnClick. При натисненні на кнопку спочатку ініціалізується бібліотека для створення ключів шифрування, генеруються ключі та записуються до внутрішнього сховища програми. Фрагмент цього коду наведено на рисунку 3.13.

```
let crypt = new JSEncrypt({ default_key_size: 512, log: true });
crypt.getKey();
store.setPublicKey(crypt.getPublicKey());
store.setPrivateKey(crypt.getPrivateKey());
```

Рисунок 3.13 – Фрагмент коду для створення та запису до сховища ключів шифрування

Після створення ключів необхідно перевірити чи користувач заповнив усі обов'язкові поля для вводу – електронна пошта, ім'я, прізвище та прізвисько користувача, якщо поля пусті, то виконання функції припиняється а незаповнені поля форми виділяються червоним. Фрагмент коду перевірки валідності полів наведено на рисунку 3.14.

```
let hasErrors = false;

if (store.emailInputValue === '' || !validate(store.emailInputValue)) {
    store.changeInputStyle('warning', 'emailInputStyle');
    hasErrors = true;
}

if (store.nameInputValue === '') {
    store.changeInputStyle('warning', 'nameInputStyle');
    hasErrors = true;
}

if (store.lastNameInputValue === '') {
    store.changeInputStyle('warning', 'lastNameInputStyle');
    hasErrors = true;
}

if (store.nicknameInputValue === '') {
    store.changeInputStyle('warning', 'nicknameInputStyle');
    hasErrors = true;
}

if (hasErrors) return;
```

Рисунок 3.14 – Фрагмент коду перевірки валідності полів

Після перевірки полів на валідність приватний та публічний ключ зберігаються до файлів private.pem для приватного ключа та public.pem для публічного ключа а також введені поля передаються на сервер для реєстрації користувача. Після виконання запиту перевіряється статус відповіді, якщо статус

200 – то показується модальне вікно з детальною інформацією про те, де знаходиться приватний ключ, щоб користувач міг зберегти його щоб потім мати можливість авторизації на інших пристроях. Якщо код 409 – значить у системі є користувач з такою поштою або таким прізвиськом і показується інформаційне модальне вікно. Фрагмент коду зі збереженням файлів та посиланням запиту на сервер наведено на рисунку 3.15.

```
Electron.files.keySave('keySave', store.publicKey, 'public');
Electron.files.keySave('keySave', store.privateKey, 'private');

let [response, error] = await store.registerUser();

if (error) {
    console.log(error);
    switch (error.status) {
        case 409:
            switch (error.errors.field) {
                case 'nickname':
                    store.changeInputStyle('warning', 'nicknameInputStyle');
                    Electron.errors.showError('error', 'User with such nickname already exist!', 'Registration error');
                    break;
                case 'email':
                    store.changeInputStyle('warning', 'emailInputStyle');
                    Electron.errors.showError('error', 'User with such email already exist!', 'Registration error');
                    break;
            }
            break;
        case 400:
            history.push('/error');
            break;
    }
}

if (response) store.showPopup = 'block';
```

Рисунок 3.15 - Фрагмент коду зі збереженням файлів та посиланням запиту на сервер

3.6 Розробка модулів завантаження/видалення фото профілю

Модуль завантаження/зміни фото профілю а також модуль видалення фото профілю реалізовано у компоненті MainViewProfile. Код компоненту MainViewProfile наведено на рисунку 3.16.

```
const MainViewProfile = observer(({store}) => {
    let history = useHistory();
    let styleObj = {
        backgroundImage: store.userObj?.avatar
            ?
            "url(" + process.env.BACKEND_URL + "/" + store.userObj.avatar + ")"
            :
            "url('./build/assets/defaultAvatar.png')"
    }

    let exitBtnStyleObj = {
        backgroundImage: `url('./build/assets/exitbtn.png')`
    }

    const inputFile = useRef(null);

    return <div className="main-view-profile">
        <div
            className="exit-btn"
            style={exitBtnStyleObj}
            onClick={exitBtnClick}
        >
        </div>
        <input
            type='file'
            id='file'
            ref={inputFile}
            style={{ display: 'none' }}
            onChange={inputFileOnChange}
        />

        <div
            ...
        >
        </div>
        <p className="main-view-user-name">
            {store.userObj?.name + " " + store.userObj?.lastName}
        </p>

        <p
            className="main-view-profile-options"
        >
            Profile settings
        </p>

        <p
            className="main-view-profile-options"
            onClick={deleteAvatar}
        >
            Delete profile image
        </p>
        <p
            className="main-view-profile-options"
        >
            Delete profile
        </p>
    </div>
})
```

Рисунок 3.16 – Код компоненту MainViewProfile

При натисненні на фото профілю користувача викликається функція inputFileOnChange, яка слугує для зміни/встановлення нового фото профілю користувача. Код функції inputFileOnChange наведено на рисунку 3.17.

```
async function inputFileOnChange(event) {
        event.stopPropagation();
        event.preventDefault();

        if (!event.target.files) return;

        let file = event.target.files[0];

        let formData = new FormData();

        formData.append("avatar", file, file.name);

        let saveImage;

        let [request, requestError] = await axios(
            formData,
            `${process.env.BACKEND_URL}/user/${store.userObj?.id}/image`,
            "PATCH",
            {
                token: localStorage.getItem('token'),
                "Content-Type": "multipart/form-data"
            });

        if (requestError) saveImage = null;

        saveImage = request.data;

        if (saveImage) {
            store.updateAvatar(saveImage.avatar);
        }
}
```

Рисунок 3.17 – Код функції inputFileOnChange

При виклікі функції на серверну частину відправляється запит з новим фото профілю користувача, після чого на екрані відображатиметься нове фото профілю користувача.

При натисненні на кнопку “Delete profile image” викликається функція deleteAvatar, яка слугує для видалення фото профілю користувача. Код функції

deleteAvatar наведено на рисунку 3.18.

```
async function deleteAvatar(event) {
    let [deleteImage, deleteImageError] = await axios({},
        `${process.env.BACKEND_URL}/user/${localStorage.getItem('id')}/image`,
        'DELETE', { token: localStorage.getItem('token') });

    if (deleteImageError) return history.push('/error');

    store.userObj.avatar = null;
}
```

Рисунок 3.18 – код функції deleteAvatar

При викликі функції deleteAvatar виконується запит на серверну частину для видалення фото профілю користувача, після чого файл фото видаляється на сервері а посилання на фото видаляється з бази даних.

3.7 Висновки

В даному розділі було проведено аналіз таких мов програмування як JavaScript, PHP та Ruby. Враховуючи переваги мови JavaScript як для серверної так і для клієнтської частини було обрано саме її. В якості середовища розробки порівнювались Visual Studio, WebStorm та Atom. WebStorm виявилась найкращою серед представлених конкурентів, тому було обрано саме її. Також було розроблено загальну модель програми, також було розроблено: модуль шифрування повідомлень, модуль розшифрування повідомлень а також модулі завантаження та видалення фото профілю користувача.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Аналіз методів тестування

Тестування програмного забезпечення (англ. Software Testing) – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і програмних складових з метою оцінки[24]. Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники;
- правильна відповідь для усіх можливих вхідних даних;
- виконання функцій за прийнятний час;
- практичність;
- сумісність з програмним забезпеченням та операційними системами;
- відповідність задачам замовника[24].

Навіть для простого програмного забезпечення кількість можливих тестів може бути практично нескінченно. Тому тактика тестування полягає у тому, щоб провести найважливіші тести з урахуванням часу та доступних ресурсів. Результатом цього є те, що розроблене програмне забезпечення необхідно тестувати звичайним виконанням програми, метою якого є виявлення та пошук багів (дефектів роботи)[24].

Тестування можна проводити навіть тоді, коли створено код, який можна виконати не обов'язково повністю. Під час розробки слід вводити тестування на план і передбачати його. При поетапному процесі кількість та вид тестів задається після визначення вимог до програмного забезпечення, після чого починається розробка[24].

Статичне тестування – це тестування, діяльність якого пов'язана з аналізом та перевіркою результатів розробки програмного забезпечення. Такий вид

тестування виконується без запуску програми на пристрої, тому передбачає перевірку і контроль коду програми[24].

Динамічне тестування – це тестування, на противагу статичному, передбачає запуск програмного забезпечення на пристрої з його подальшою перевіркою[24].

Статичне та динамічне тестування зазвичай виконуються в комплексі.

При статичному тестуванні провадиться перевірка документації, яка є результатом життєвого циклу програми. До цього можна зарахувати код, специфікація, технічне завдання. Документація, отримана, аналізується відповідно до вимог стандартів програмування. Після статичної перевірки можна отримати результат того, наскільки розроблене програмне забезпечення відповідає тим критеріям та вимогам, які були поставлені замовником[24].

Методи динамічного тестування використовуються під час виконання програми. Правильна робота програми перевіряється на заданій кількості тестів. При виконанні тесту його результати збираються та аналізуються[24].

На сьогодні тестування програмного забезпечення – один з найбільш дорогих етапів життєвого циклу програмного забезпечення, на нього відводиться від 50% до 65% загальних витрат[24].

4.2 Тестування модулю реєстрації

Перед тим як користувач зможе користуватись додатком, необхідно зареєструватись у системі. Для тесту при реєстрації декілька раз будуть введені невалідні дані в різних комбінаціях а також валідний набір даних для перевірки працездатності програми. Щоб розпочати процес реєстрації спочатку необхідно запустити додаток, після чого користувач опиниться на екрані першого етапу авторизації, який наведено на рисунку 4.1. Для переходу на екран реєстрації необхідно натиснути на посилання виділене синім кольором. Екран реєстрації наведено на рисунку 4.2.

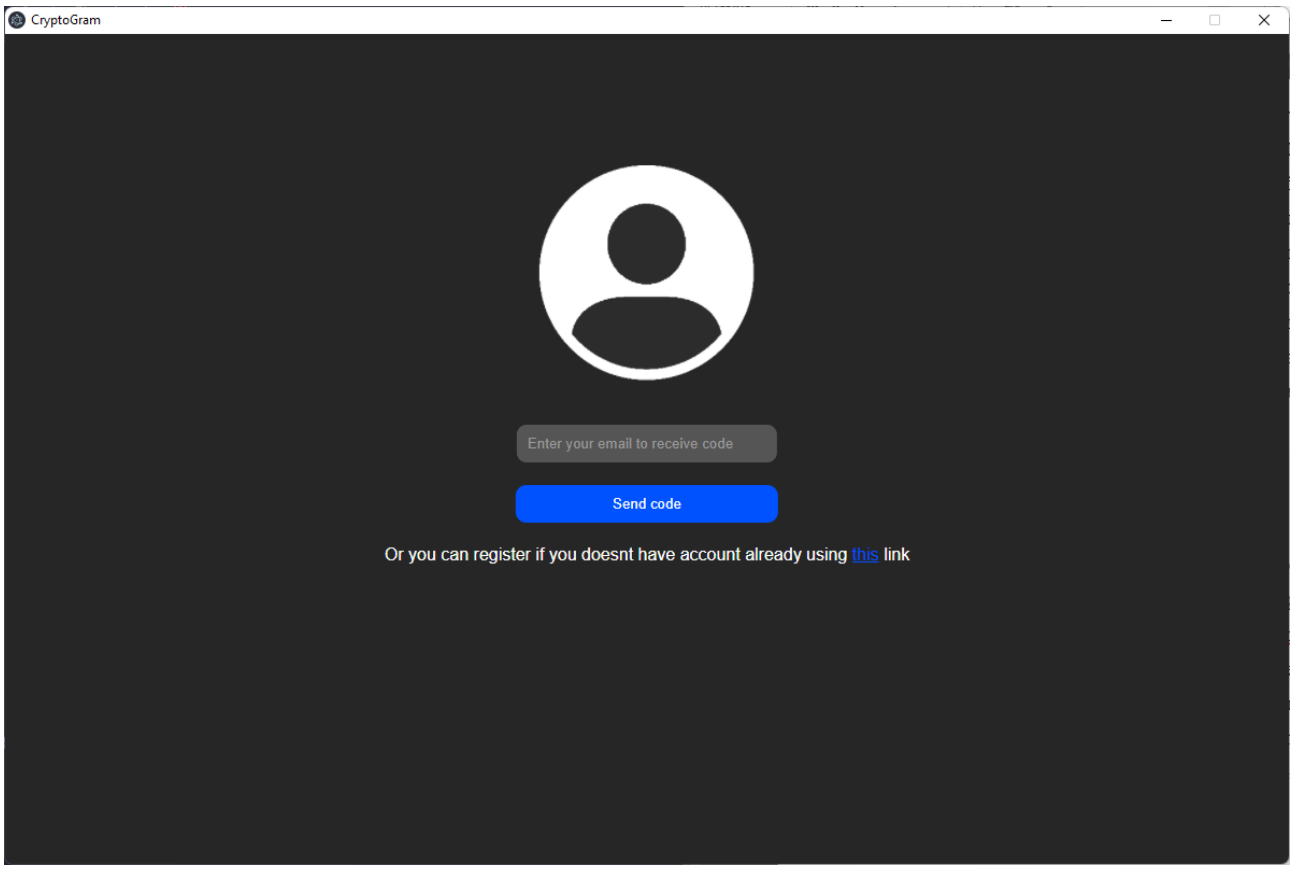


Рисунок 4.1 – Екран першого етапу авторизації

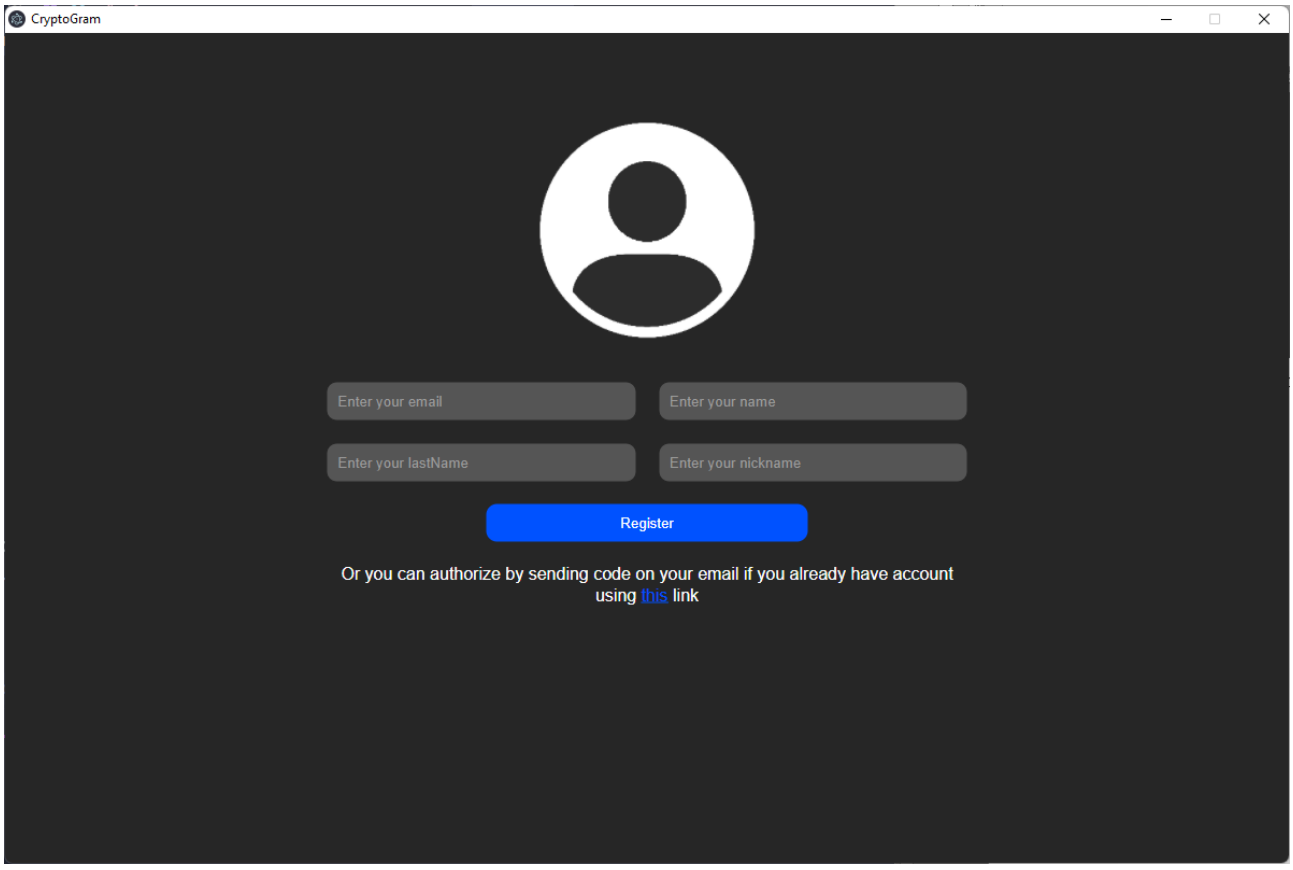


Рисунок 4.2 – Екран реєстрації

Після переходу на екран реєстрації необхідно заповнити таку інформацію про користувача: електронна пошта, ім’я, фамілію та прізвисько користувача. Для тесту спочатку усі поля будуть незаповнені. При натисненні на кнопку “Register” програма підсвітить незаповнені поля червоним кольором (рисунок 4.3).

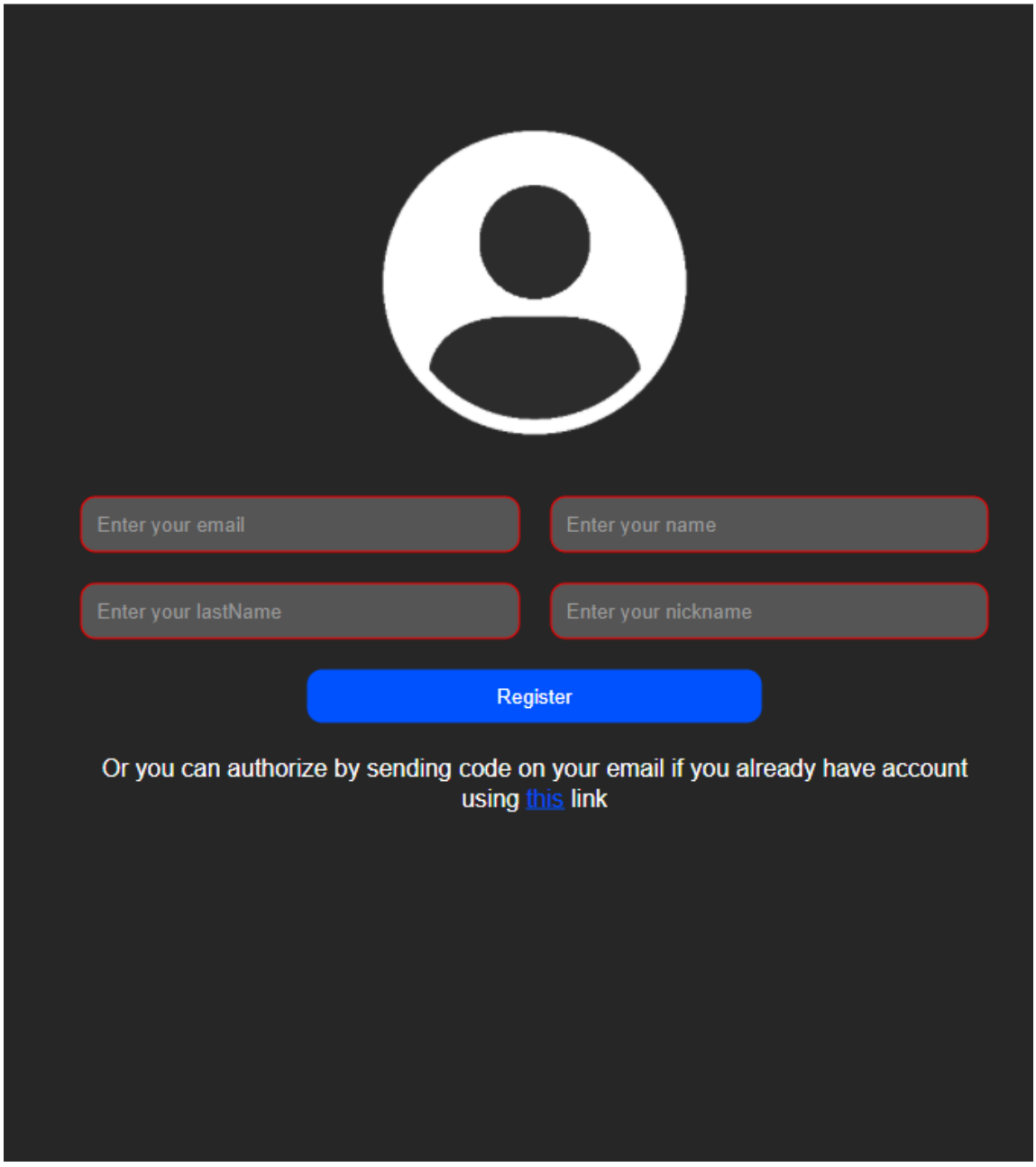


Рисунок 4.3 – Екран реєстрації з незаповненими даними

Якщо заповнити поля, але не всі, то червоним будуть підсвічені лише ті поля, які не заповнені. Приклад наведено на рисунку 4.4.

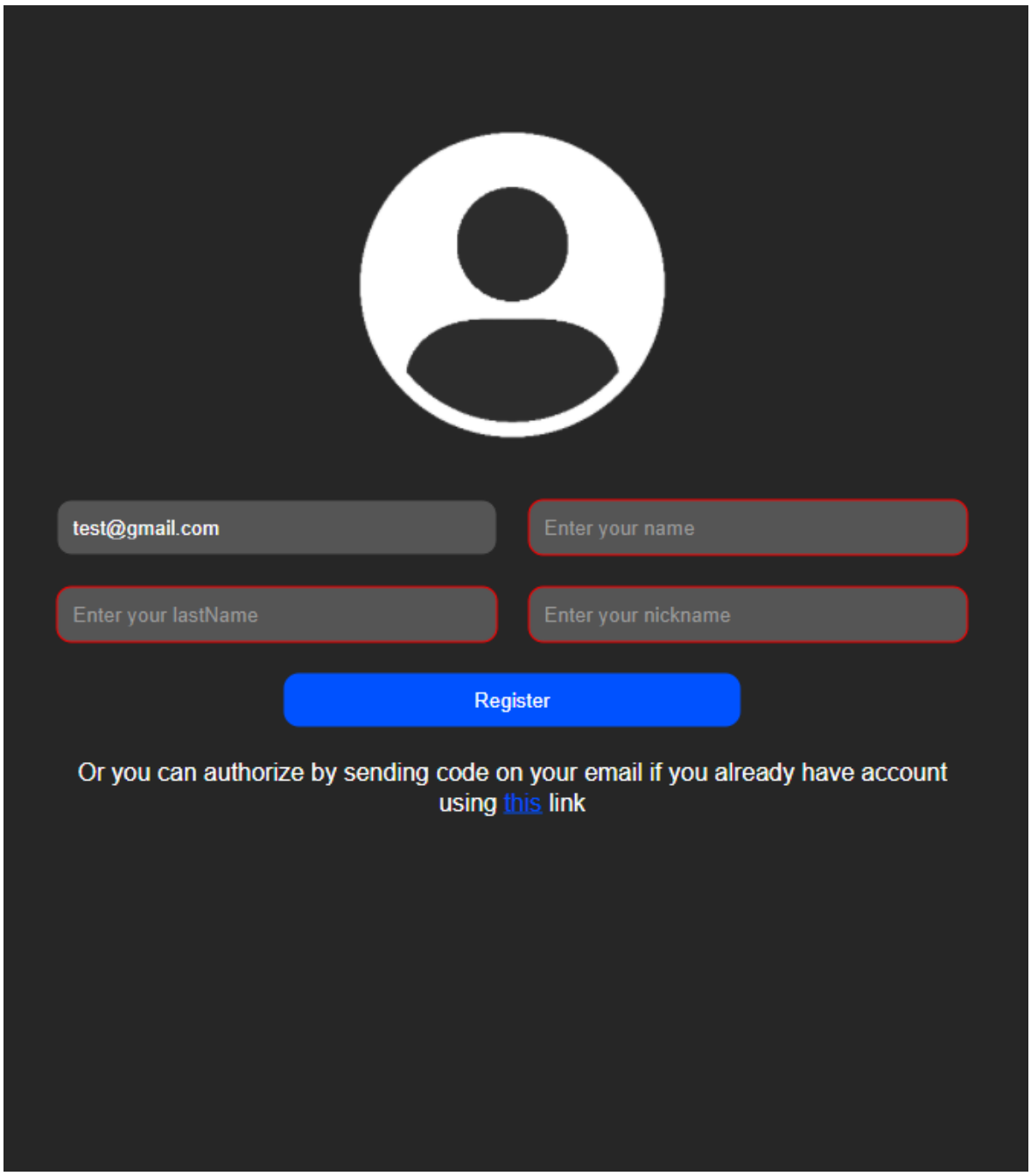


Рисунок 4.4 –Екран реєстрації з частково незаповненими даними

Також якщо ввести невалідні дані, наприклад неправильну електронну пошту користувача то неправильно заповнене поле також буде підсвічене червоним кольором, що наведено на рисунку 4.5.

Якщо заповнити поля коректними даними, тобто усі дані заповнені у правильному форматі, то відбудеться запит на сервер з заданими даними, і якщо у системі уже зареєстрований користувач з такою електронною поштою, або з таким прізвиськом, то буде показано модальне вікно з інформацією про те,

користувач з яким з цих полів уже існує в базі даних (рисунок 4.6).

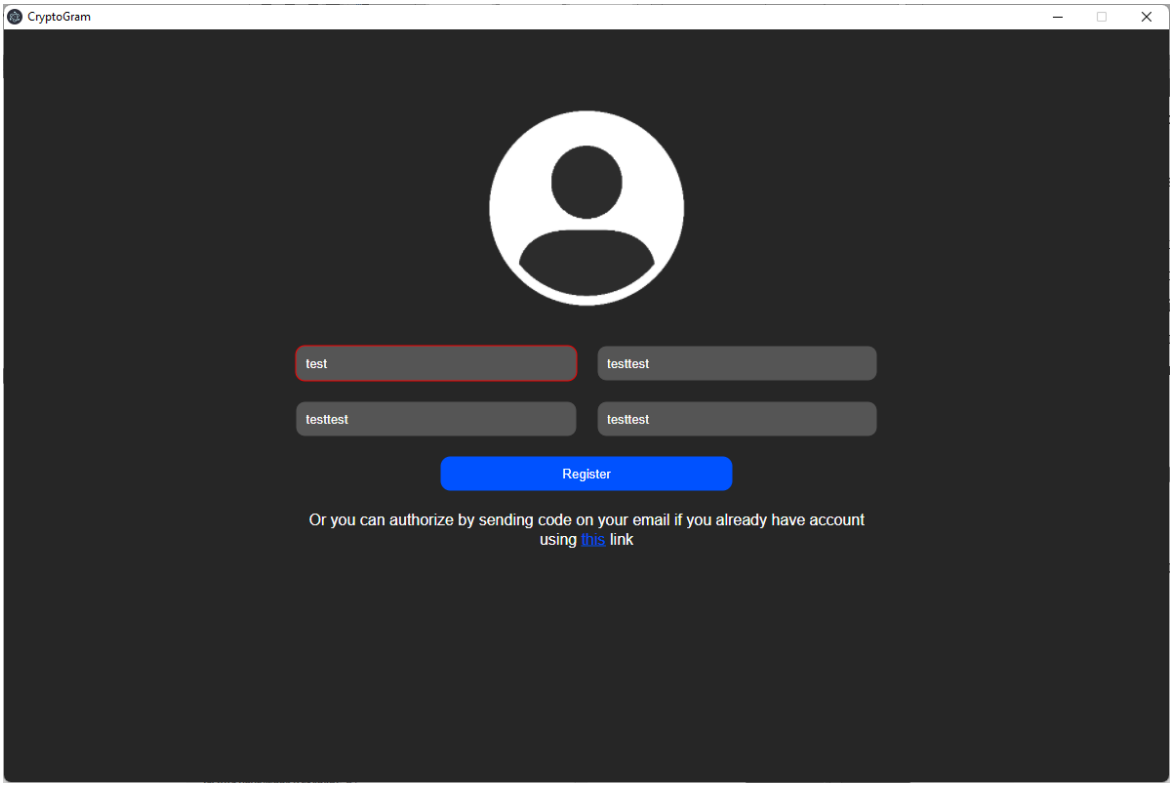


Рисунок 4.5 – Екран реєстрації з невалідними даними

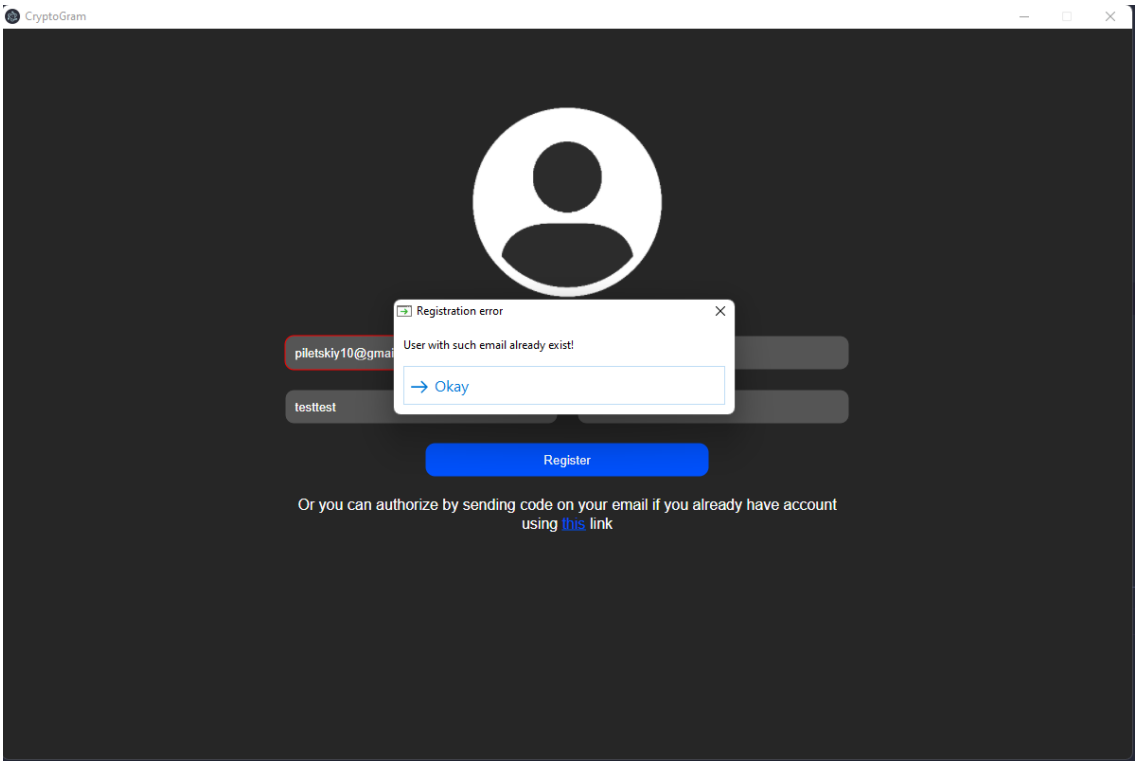


Рисунок 4.6 – Модальне вікно з інформацією про те, що користувач з такою електронною поштою вже зареєстрований у системі

Протестувавши те, як система реагує на невалідні або неповні дані необхідно перевірити як система реагує, коли усі дані заповнені правильно. Після заповнення усіх даних та натиснення на кнопку “Register” користувачу буде показано модальне вікно з інформацією про успішну реєстрацію та місцезнаходження згенерованих ключів шифрування, після чого користувач перейде на другий етап авторизації. Приклад правильно заповнених даних наведено на рисунку 4.7, модальне вікно з інформацією про успішну реєстрацію наведено на рисунку 4.8.

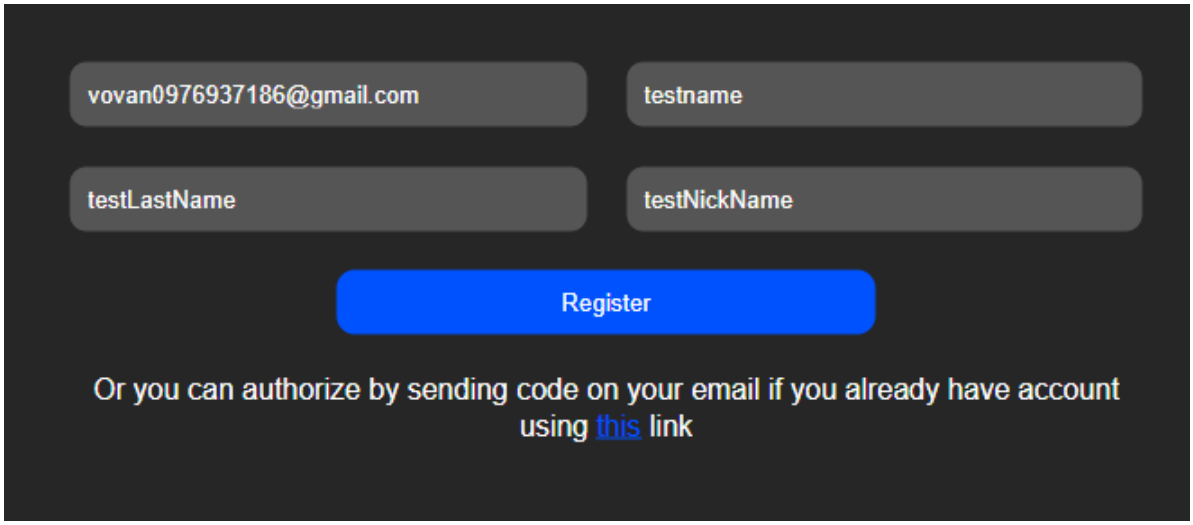


Рисунок 4.7 – Приклад правильно заповнених даних для реєстрації

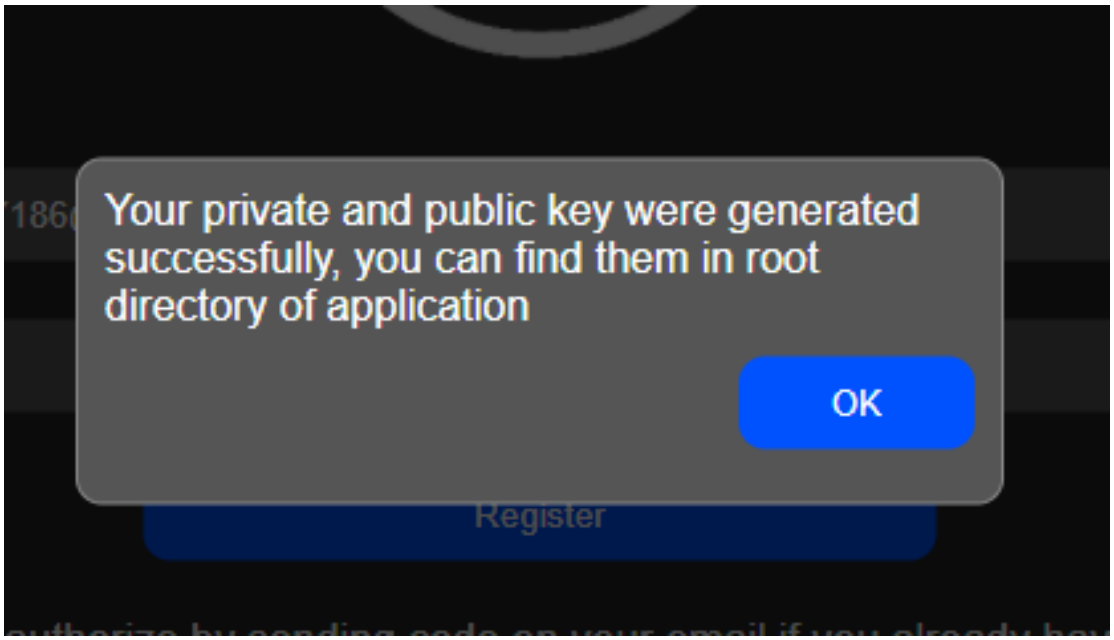


Рисунок 4.8 – Модальне вікно з інформацією про успішну реєстрацію

4.3.Тестування модулю першого етапу авторизації

Після запуску додатку користувач потрапляє на екран першого етапу авторизації, який було наведено на рисунку 4.1. Для переходу на другий етап авторизації необхідно ввести електронну пошту користувача. Якщо не ввести електронну пошту користувача і натиснути кнопку “Send code” поле вводу електронної пошти буде виділено червоним кольором, а також підказка у полі вводу буде змінена на підказку ввести електронну пошту користувача. Приклад екрану першого етапу авторизації з незаповненим полем для вводу наведено на рисунку 4.9.

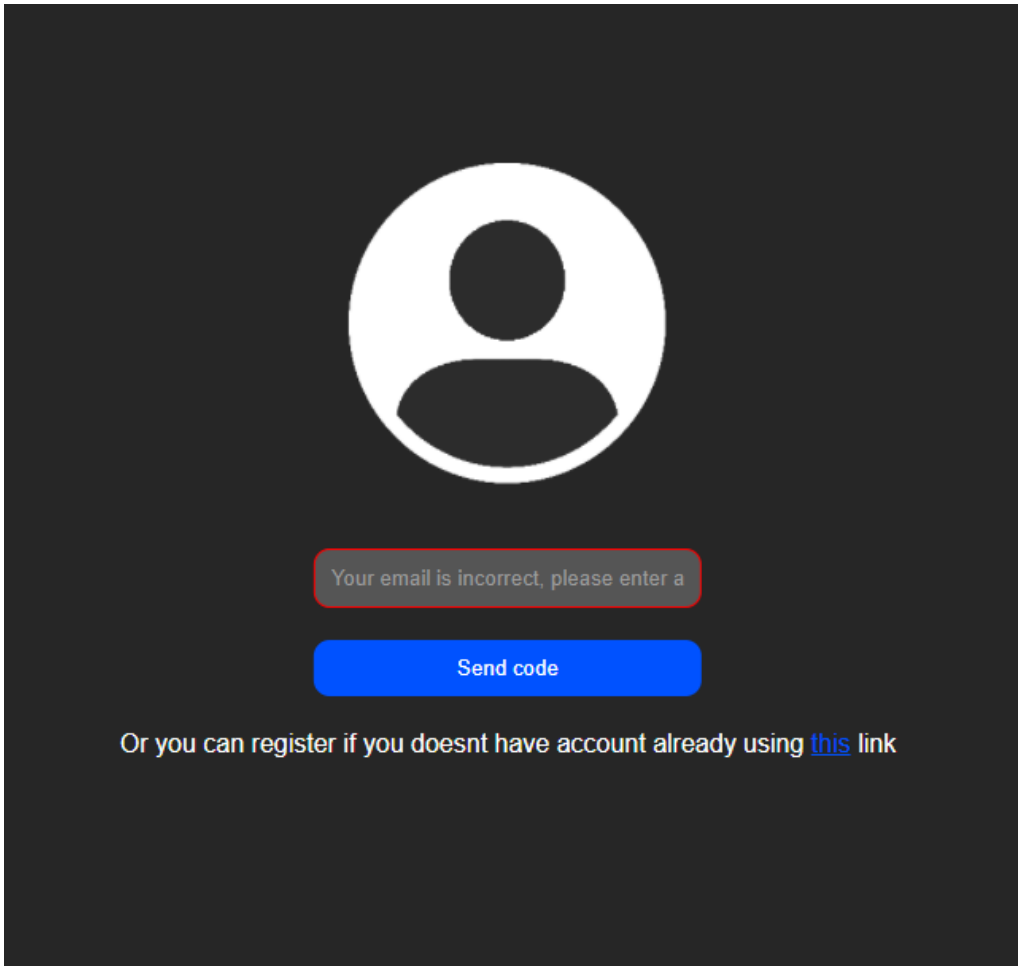


Рисунок 4.9 – Екран першого етапу авторизації з незаповненим полем для вводу

Якщо ввести у поле вводу електронну пошту в неправильному форматі, то поле також буде виділено червоним кольором. Приклад екрану першого етапу

авторизації з введеною електронною поштою в неправильному форматі наведено на рисунку 4.10.

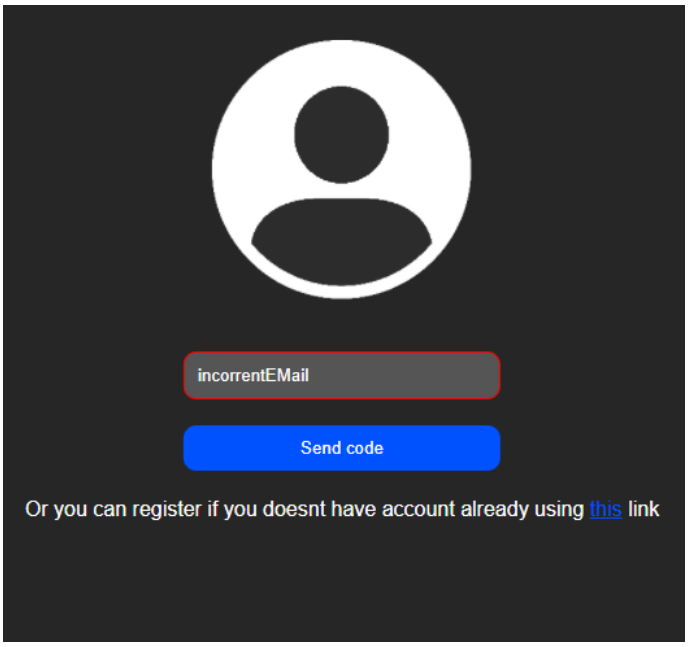


Рисунок 4.10 – Екран першого етапу авторизації з електронною поштою у неправильному форматі

Якщо ввести електронну пошту користувача в правильному форматі, але користувача з такою електронною поштою в системі не буде знайдено, буде показане модальне вікно з інформацією про те, що користувача не знайдено. Модальне вікно з інформацією про те, що користувача не знайдено в системі наведено на рисунку 4.11.

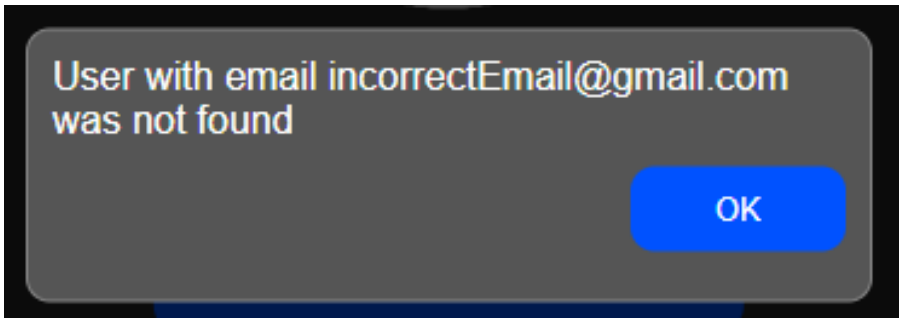


Рисунок 4.11 – модальне вікно з інформацією про те, що користувача з такою електронною поштою не знайдено у системі

Якщо ввести електронну пошту у правильному форматі і користувача з такою електронною поштою буде знайдено у системі, то користувач перейде на другий етап авторизації.

4.4 Тестування модулю другого етапу авторизації

На другому етапі авторизації користувач потрапляє на екран другого етапу авторизації, а також користувачу на електронну пошту приходить лист з кодом для авторизації. Екран другого етапу авторизації наведено на рисунку 4.12, лист з кодом з електронної пошти користувача наведено на рисунку 4.13.

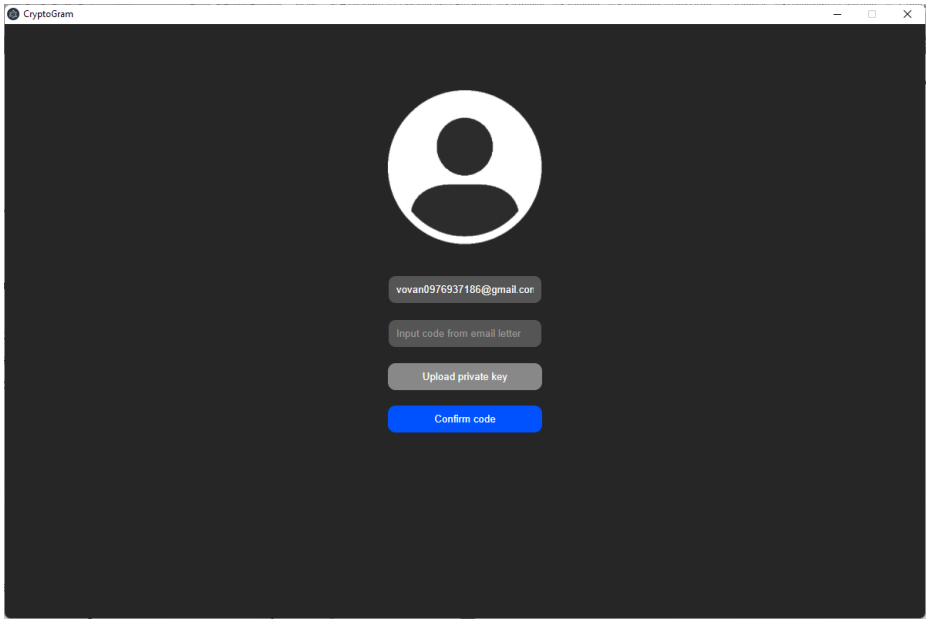


Рисунок 4.12 – Екран другого етапу авторизації

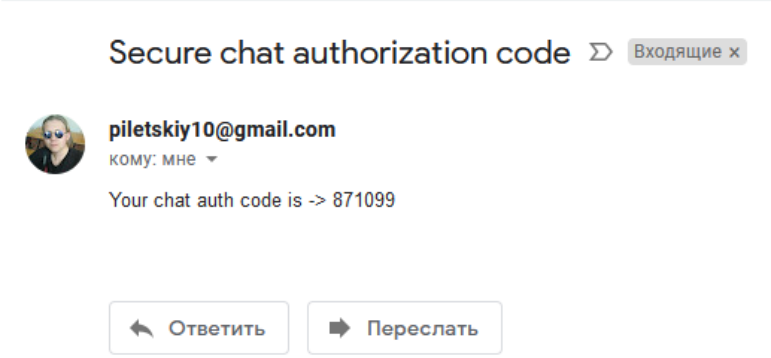


Рисунок 4.13 – Лист з кодом з електронної пошти користувача

Після того як користувачу на електронну пошту прийшов код для авторизації користувачі потрібно ввести цей код на екрані другого етапу авторизації а також вказати власний приватний ключ.

Якщо не вказати приватний ключ буде показано модальне вікно з інформацією про те, що користувач забув вказати приватний ключ користувача. Модальне вікно з інформацією про те, що користувач не вказав приватний ключ користувача наведено на рисунку 4.14.

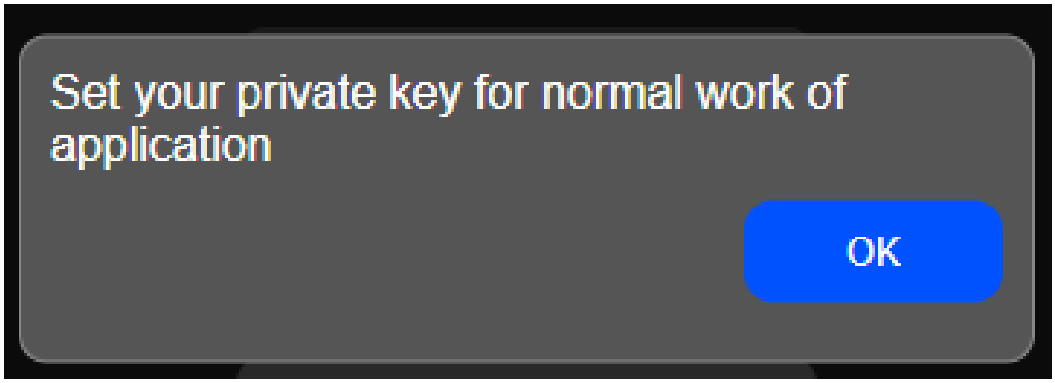


Рисунок 4.14 – Модальне вікно з інформацією про те, що користувач не вказав приватний ключ

Якщо вказати приватний ключ користувача і не вказати код з електронної пошти, буде показано модальне вікно з інформацією про те, що необхідно вказати код з електронної пошти. Модальне вікно з інформацією про те, що необхідно вказати код з електронної пошти наведено на рисунку 4.15.

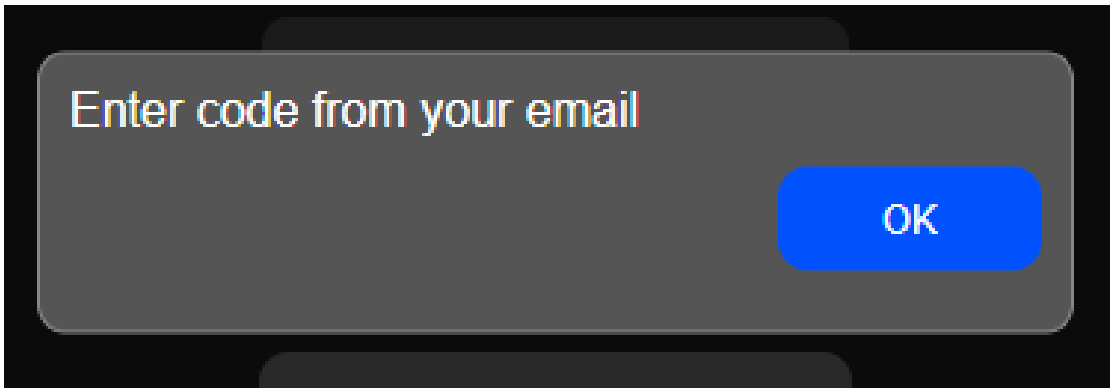


Рисунок 4.15 – Модальне вікно з інформацією про те, що необхідно вказати код з електронної пошти

Якщо вказати неправильний код користувача буде показано модальне вікно з інформацією про те, що користувач ввів неправильний код. Модальне вікно з інформацією про те, що користувач ввів неправильний код наведено на рисунку 4.16.

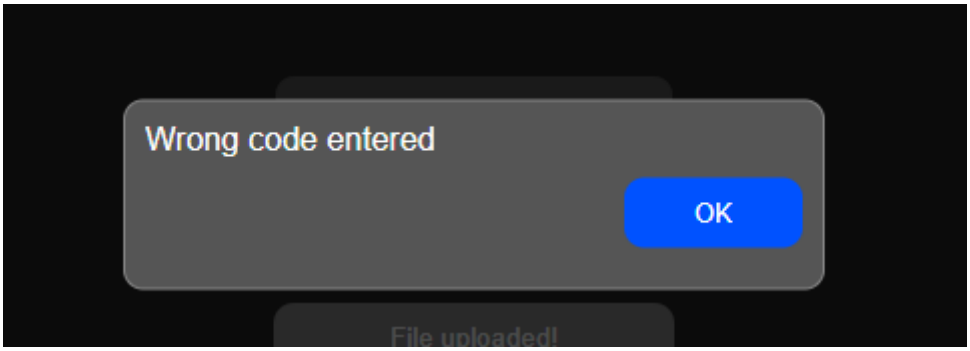


Рисунок 4.16 – Модальне вікно з інформацією про те, що користувач ввів неправильний код

Якщо ввести правильний код та вказати приватний ключ користувача, то після натиснення кнопки користувач потрапить на основний екран додатку. Основний екран додатку наведено на рисунку 4.17.

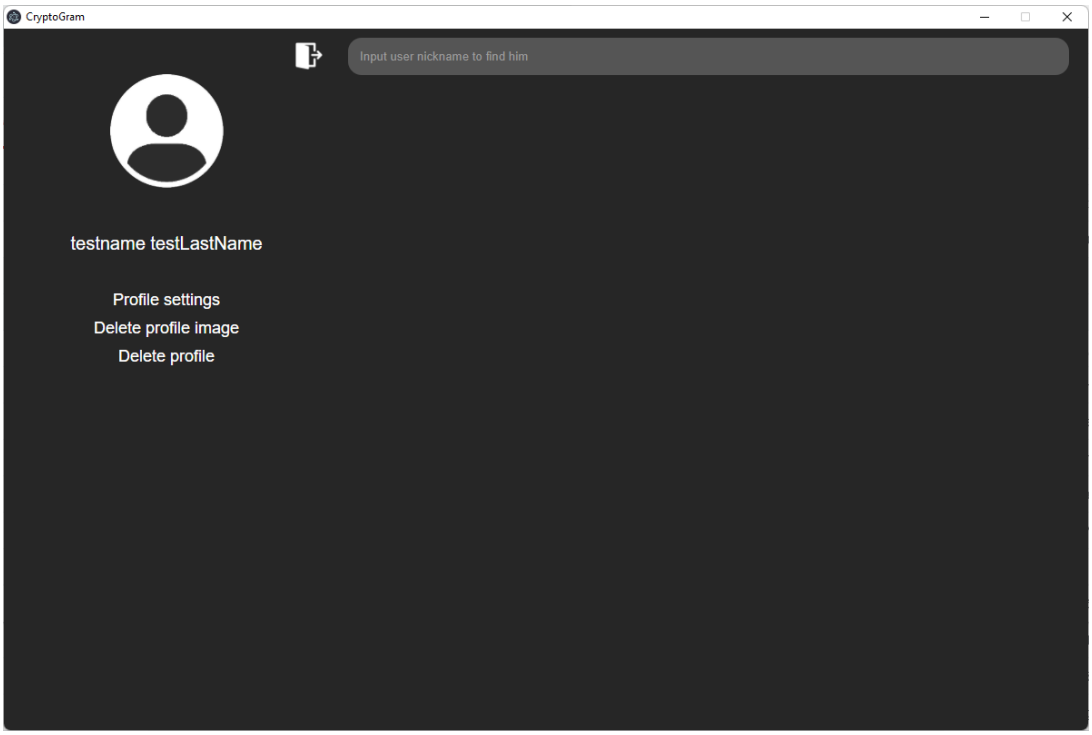


Рисунок 4.17 – Основний екран додатку

4.5 Тестування модулів оновлення та видалення фотографії профілю

Для того щоб змінити фото профілю потрібно натиснути на поточну фотографію профілю користувача. Якщо у користувача ще не встановлено ніякої фотографії профілю, то у користувача буде встановлено стандартну фотографію профілю. Стандартну фотографію профілю користувача наведено на рисунку 4.18.

Рисунок 4.18 – Стандартна фотографія профілю користувача

Після натиснення на фотографію профілю користувача відкриється файловий менеджер Підтримуються лише формати png та jpeg. Якщо обрати файл правильного формату, то фотографію профілю користувача буде оновлено на вибрану. Якщо вибрати файл неправильного формату, то фотографію профілю змінено не буде. Файловий менеджер з обраним новим фото зображено на рисунку 4.19. Оновлену фотографію профілю користувача наведено на рисунку 4.20.

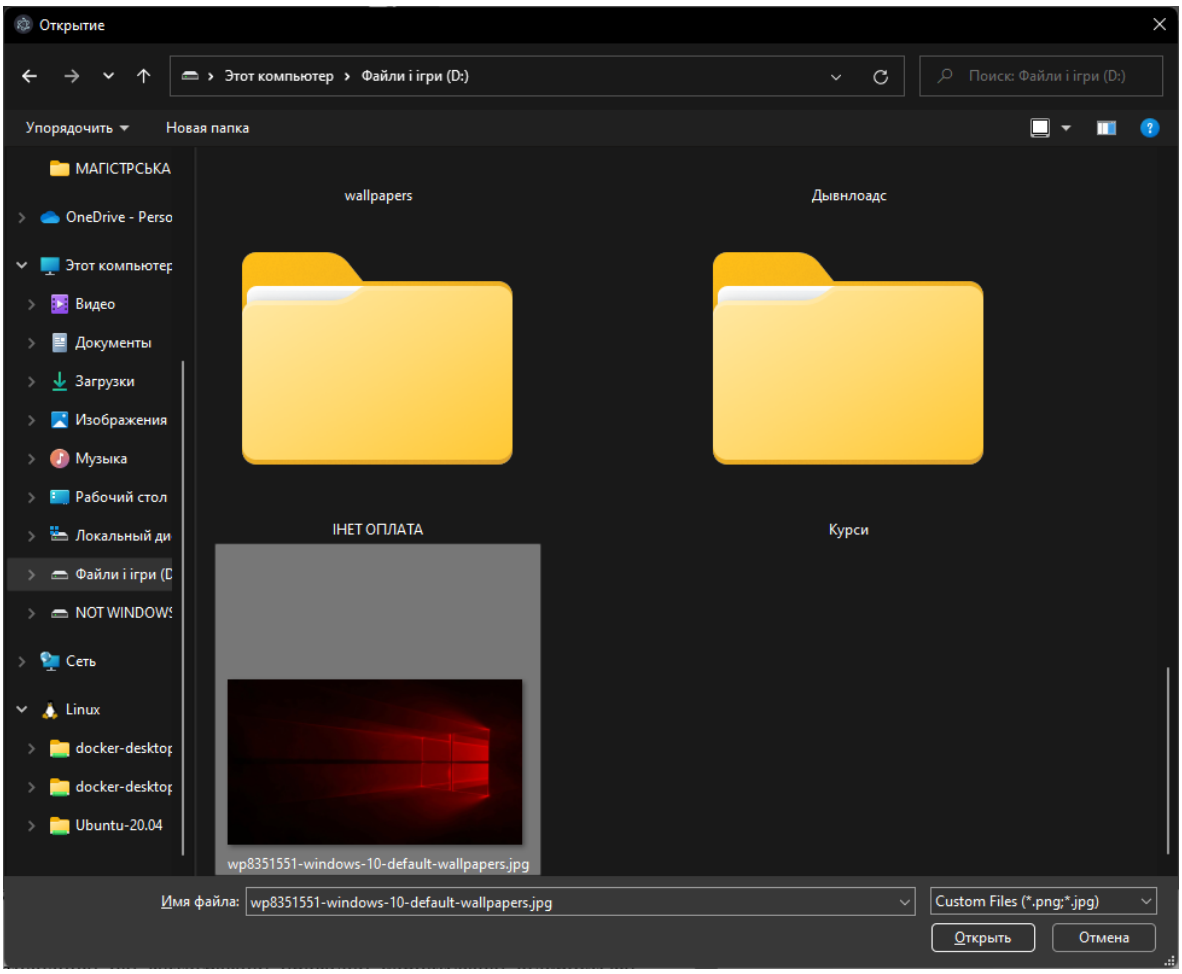


Рисунок 4.19 – Файловий менеджер з обраним новим фото

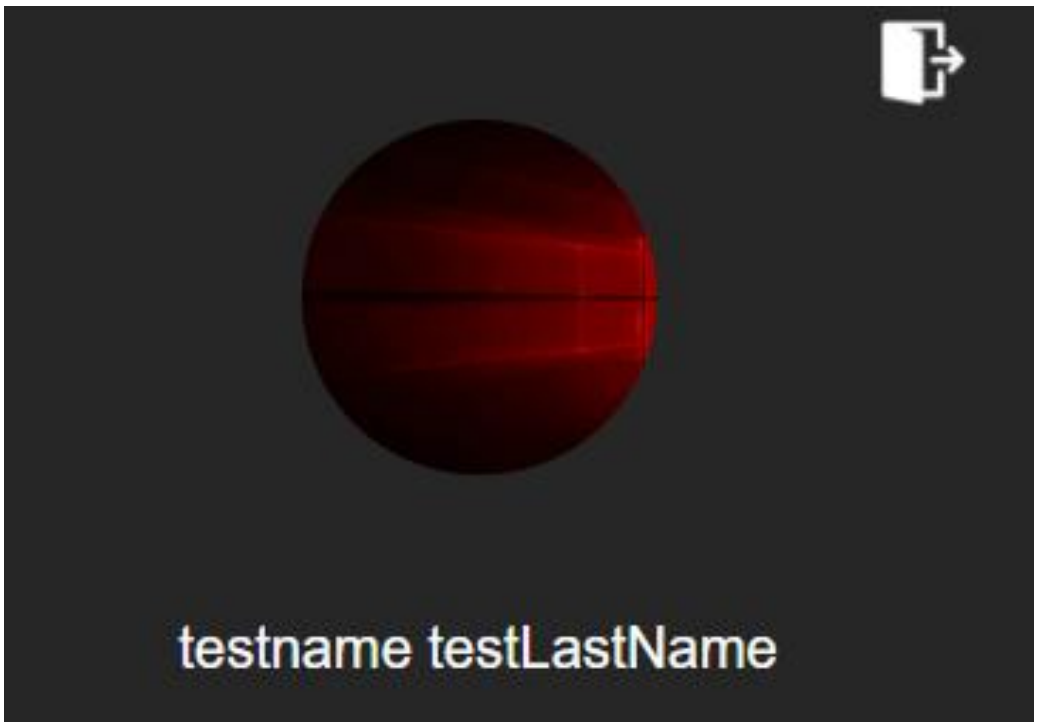


Рисунок 4.20 Оновлена фотографія профілю користувача

Для того щоб видалити фотографію профілю користувача і повернути стандартну фотографію необхідно натиснути на кнопку “Delete profile image”. Кнопка “Delete profile image” при наведенні стає синього кольору. Кнопку “Delete profile image” виділену синім кольором наведено на рисунку 4.21.

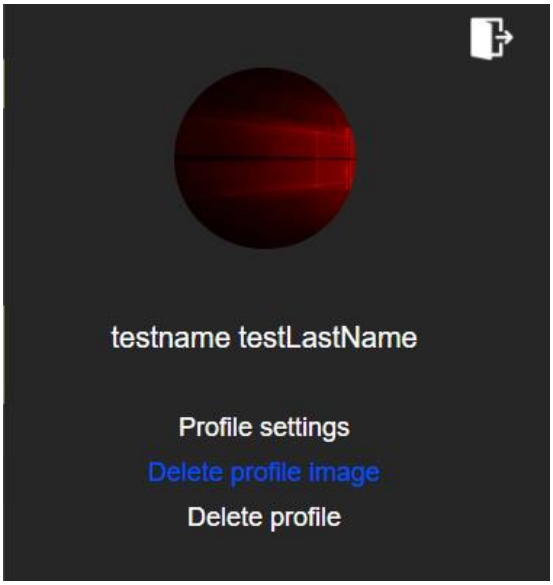


Рисунок 4.21 – Кнопка “Delete profile image” виділена синім кольором

Після натиснення на кнопку “Delete profile image” фотографію профілю змінюється на стандартну, що наведено на рисунку 4.22.

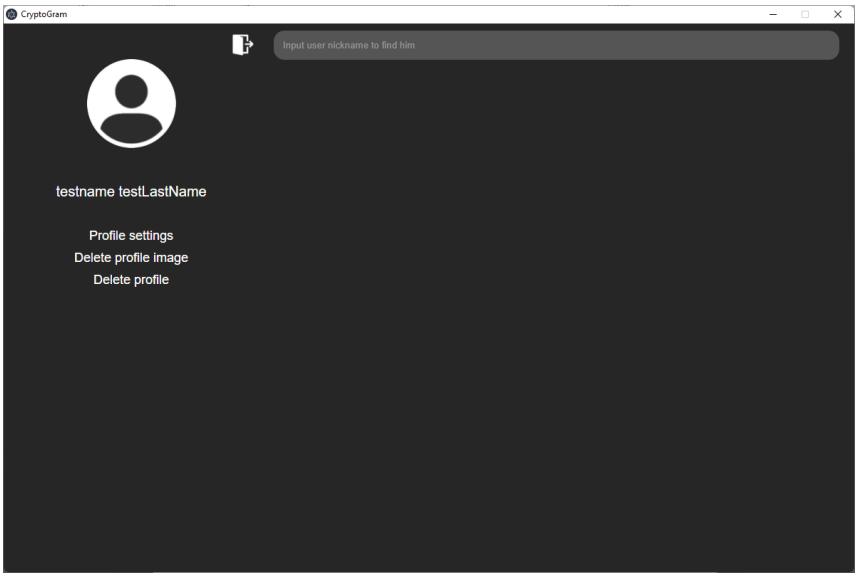


Рисунок 4.22 – Фотографію профілю користувача змінено на стандартну після натиснення кнопки “Delete profile image”

4.6 Тестування пошуку користувачів по нікнейму

У додатку реалізовано пошук інших користувачів по їх нікнейму. Поле для пошуку користувачів по нікнейму наведено на рисунку 4.23.

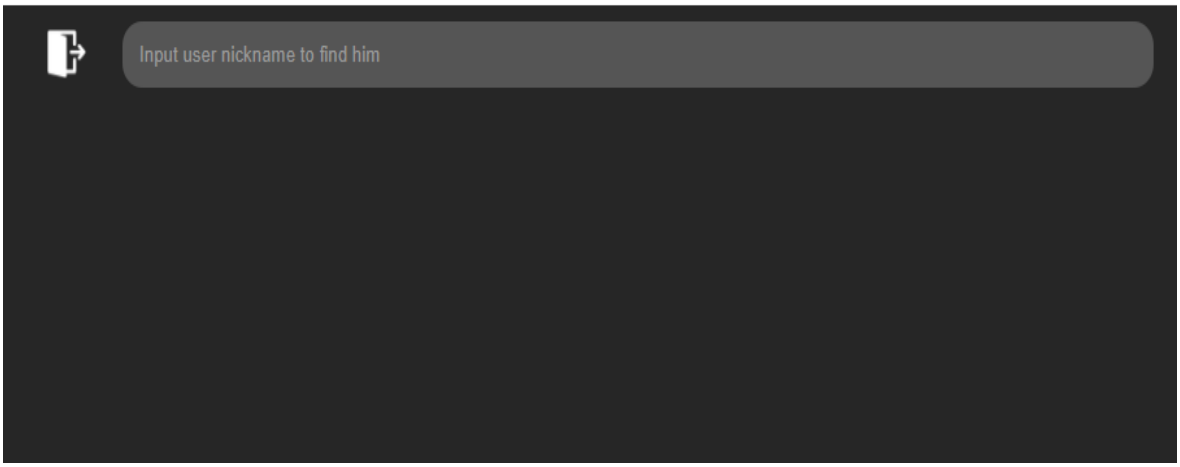


Рисунок 4.23 – Поле для пошуку користувачів по нікнейму

Пошук користувачів відбувається по частині строки, тобто якщо ввести будь яку букву, то виведуться усі користувачі, нікнейм яких містить цю букву на початку нікнейму, в кінці або по середині нікнейму. На момент тестування у базі даних зареєстровані 5 користувачів з такими нікнеймами: pilya, vinogradik, test, test2, testNickName. Тестування у прикладі проводиться під користувачем з нікнеймом testNickName. Це важливо, тому що алгоритм пошуку реалізований таким чином, щоб користувач не міг знайти самого себе. Якщо ввести букву “v” повинен показатись лише 1 користувач. Якщо ввести “test” повинно показатись 3 користувача, але так як пошук виконується з під користувача, який сам попадає під параметри пошуку, то у результатах пошуку буде показано лише 2 користувача. Якщо ввести “p” повинен показатись лише 1 користувач. Якщо ввести символ “i” повинно бути показано три користувача, але як і у випадку з пошуком по “test”, так як сам користувач потрапляє під параметри пошуку, буде показано лише 2 користувача так як сам користувач у пошукі відображений не буде. Результати тестування пошуку по символам та частині нікнейму наведено на рисунках 4.24, 4.25, 4.26, 4.27.

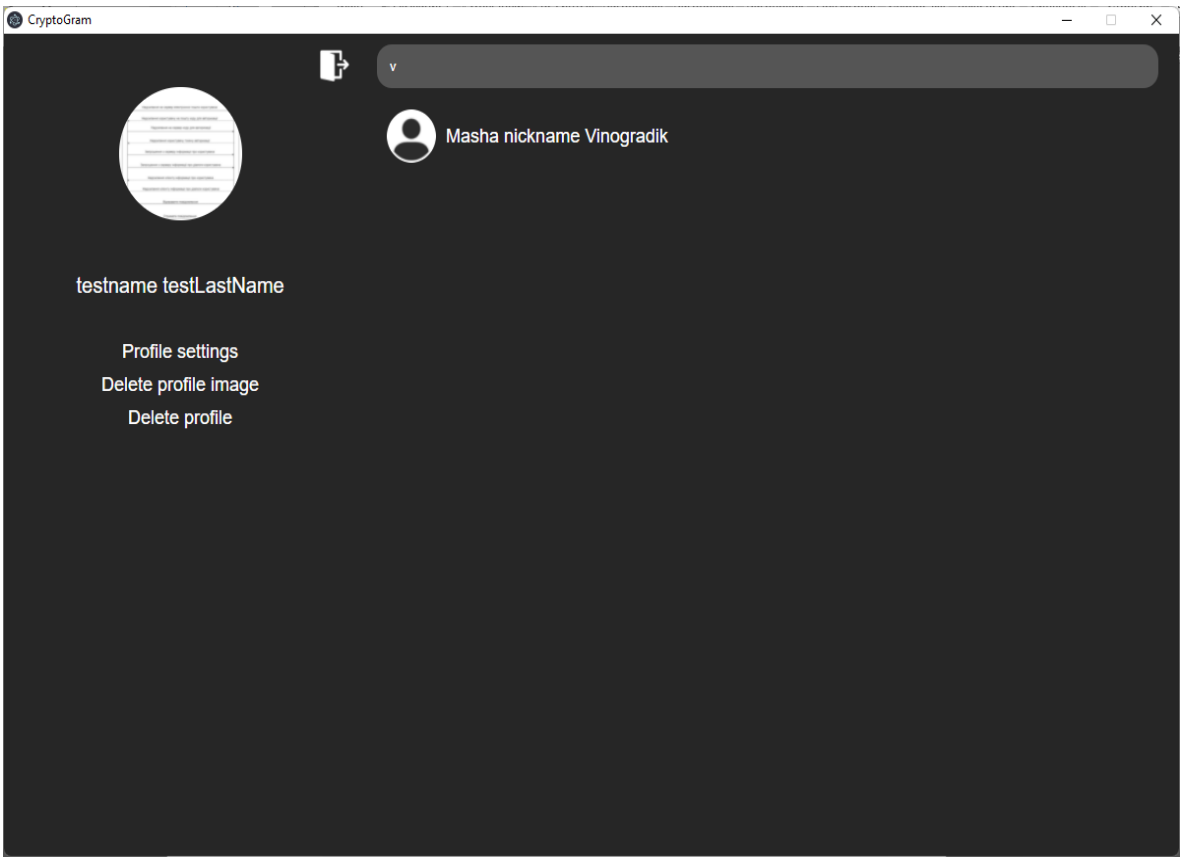


Рисунок 4.24 – Результати пошуку по букві “v”

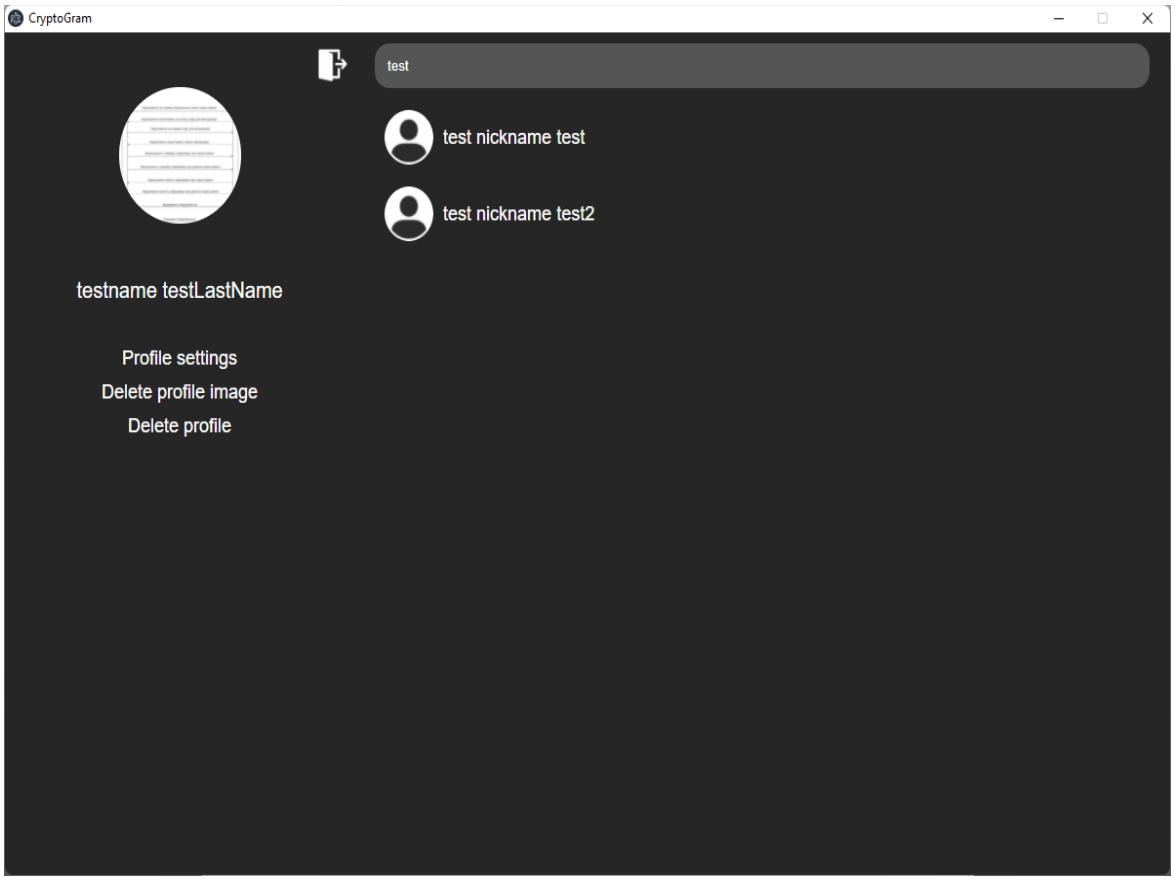


Рисунок 4.25 – Результати пошуку по слову “test”

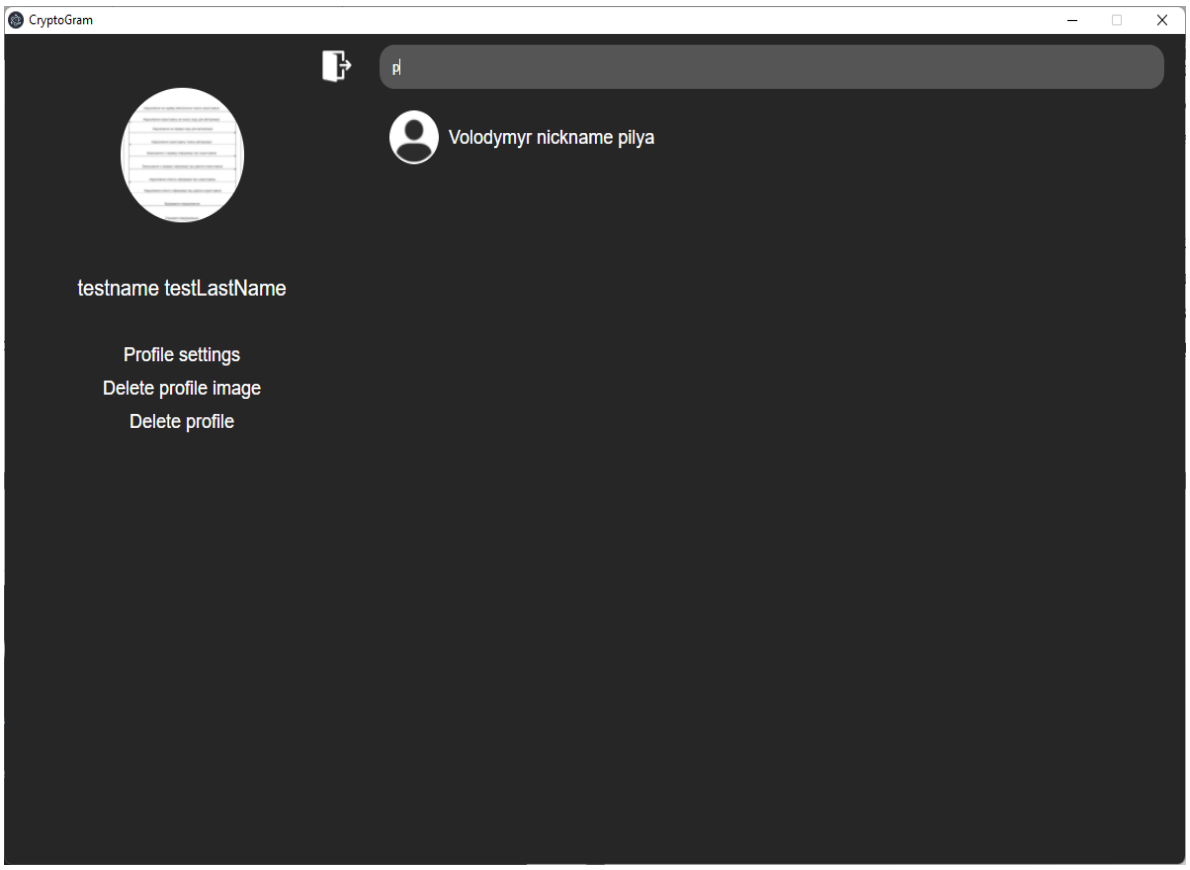


Рисунок 4.26 – Результати пошуку по букві “p”

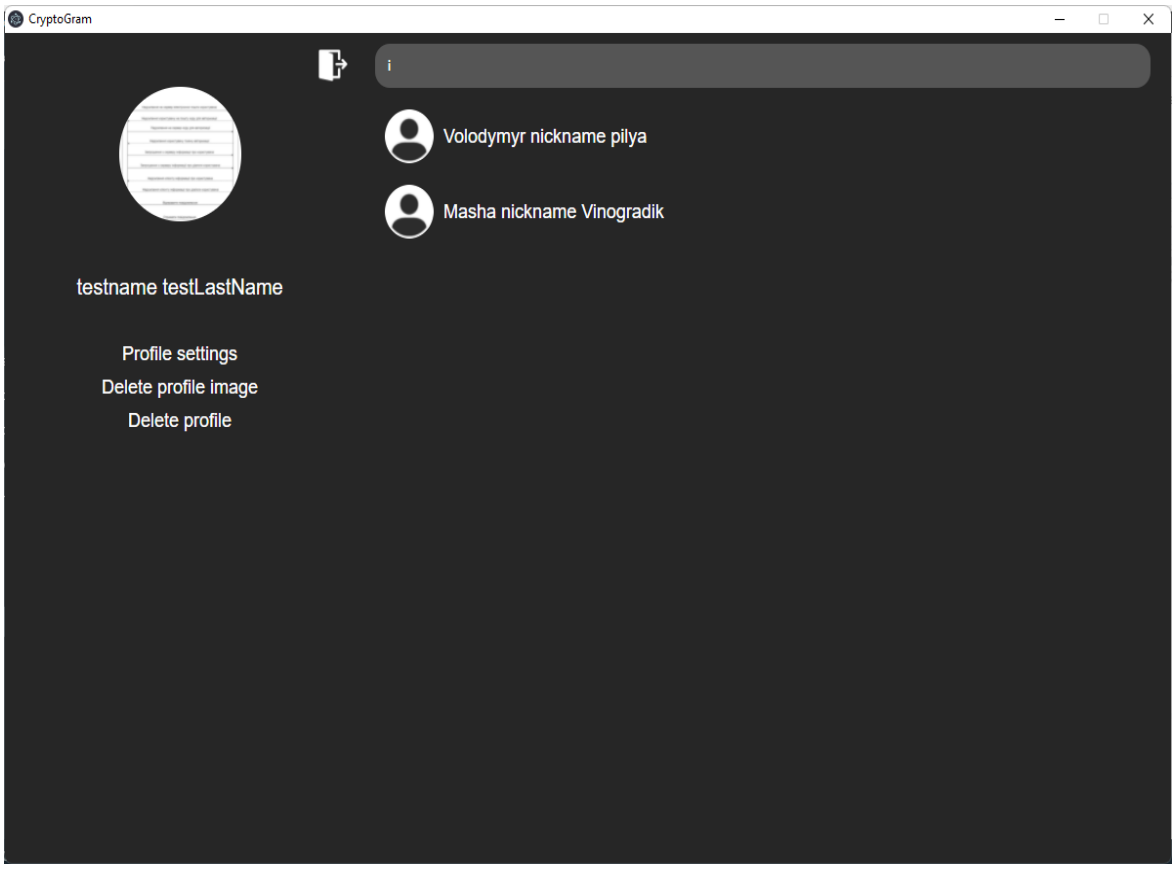


Рисунок 4.27 – результати пошуку по букві “i”

Для того щоб створити діалог зі знайденим користувачем потрібно натиснути на карточку потрібного користувача. Після чого користувач потрапить на екран діалогу з користувачем, а також діалог з обраним користувачем буде відображатись на головному екрані. Екран діалогу з користувачем після натиснення на карточку користувача зображено на рисунку 4.28. Діалог з користувачем на головному екрані зображено на рисунку 4.29.

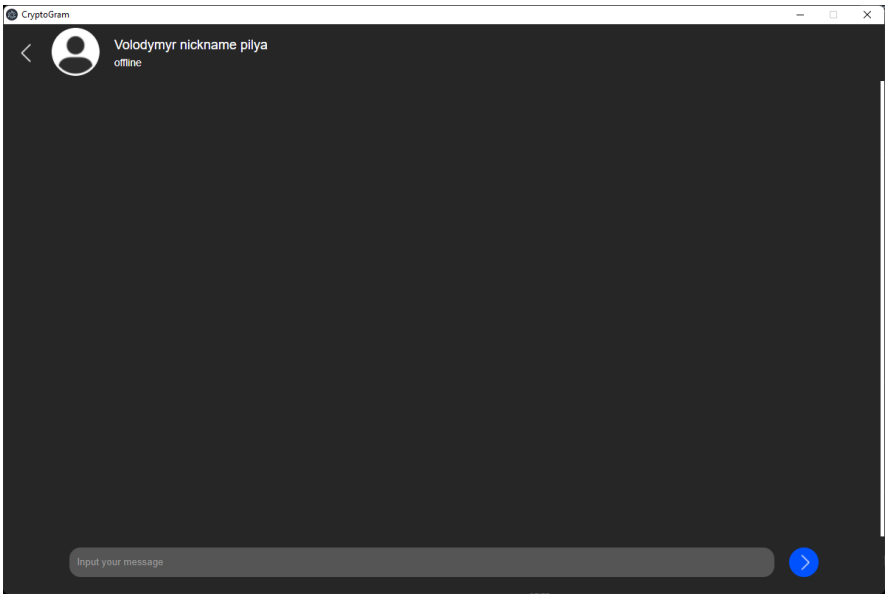


Рисунок 4.28 - Екран діалогу з користувачем після натиснення на карточку користувача

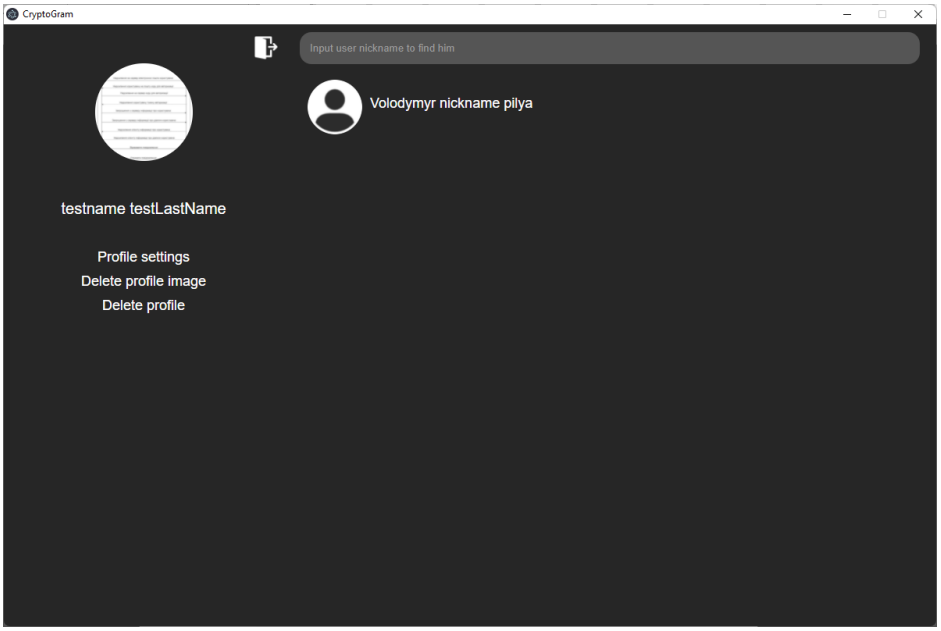


Рисунок 4.29 - Діалог з користувачем на головному екрані

4.7 Тестування відправки та отримання повідомлень

Для відправки повідомлення необхідно перейти в діалог з користувачем та у поле вводу ввести повідомлення. Поле для вводу повідомлень наведено на рисунку 4.30.

Рисунок 4.30 – Поле для вводу повідомлень

Для відправлення набраного повідомлення необхідно натиснути кнопку Enter або лівою кнопкою миші натиснути на синю кнопку справа від поля вводу повідомлення. Повідомлення відправлені користувачем відображаються справа, а повідомлення відправлені користувачем, з яким ведеться діалог відображаються зліва.

Екран діалогу з користувачем з відправленим повідомленням наведено на рисунку 4.31.

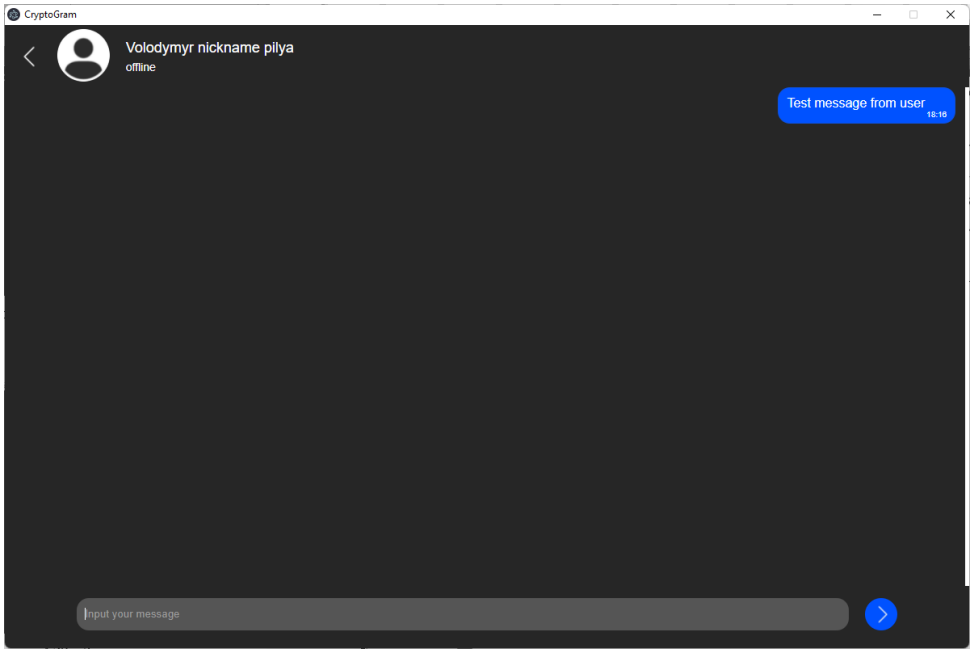


Рисунок 4.31 - Екран діалогу з користувачем з відправленим повідомленням

Для тестування тепер необхідно авторизуватись під аккаунтом користувача, якому було відправлено повідомлення і перейти в той самий діалог. Головний екран нового авторизованого користувача наведено на рисунку 4.32.

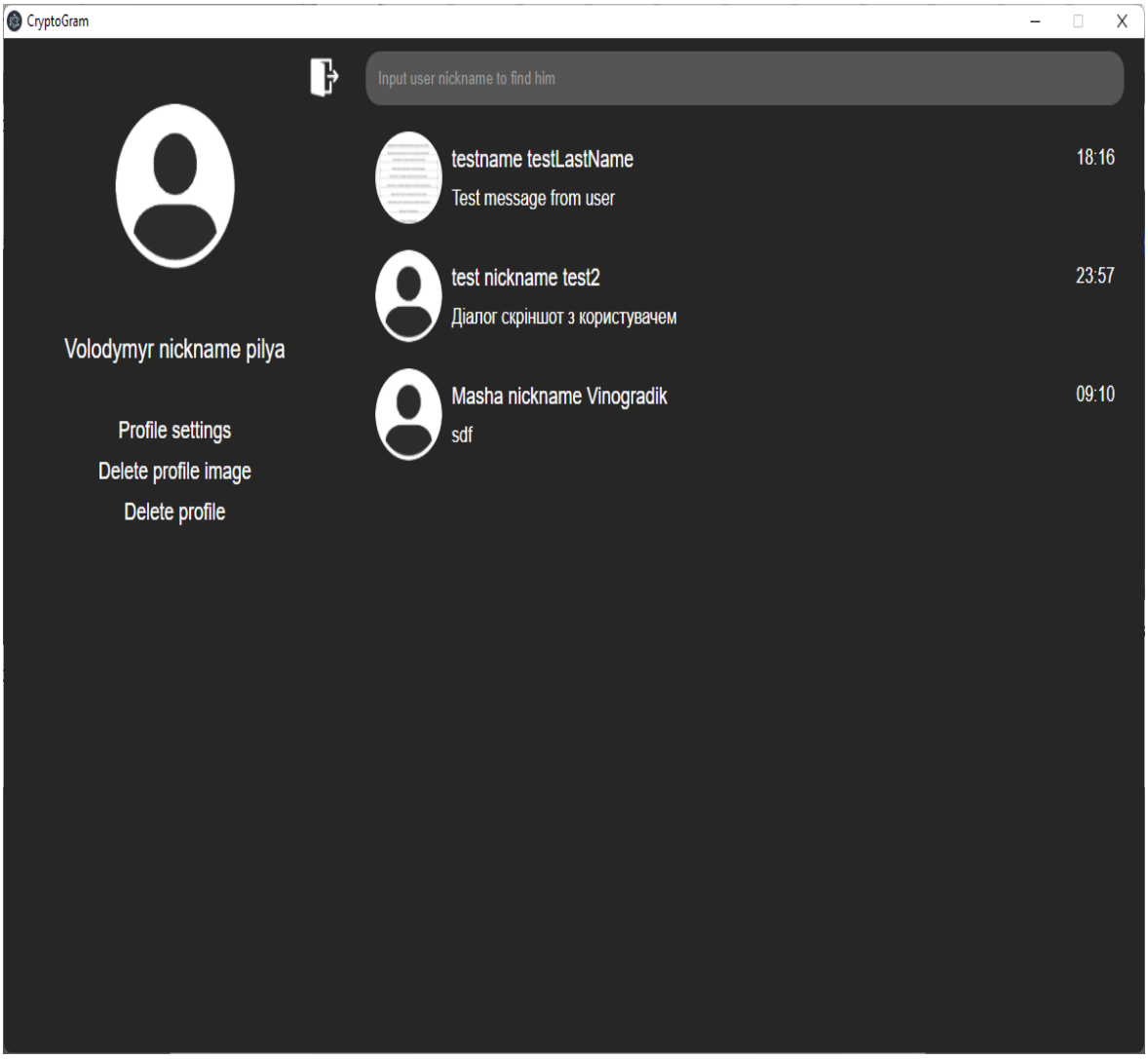


Рисунок 4.32 – Головний екран користувача з нікнеймом pilya

Як видно з рисунку 4.32 першим у списку діалогів є діалог з користувачем testname testLastName, з аккаунту якого і було відправлено повідомлення “Test message from user”. Якщо з аккаунту користувача Volodymyr nickname pilya то відправлене повідомлення буде відображене не справа, а зліва, так як тепер це повідомлення відправлене іншим користувачем. Діалог від лиця користувача Volodymyr nickname pilya наведено на рисунку 4.33.

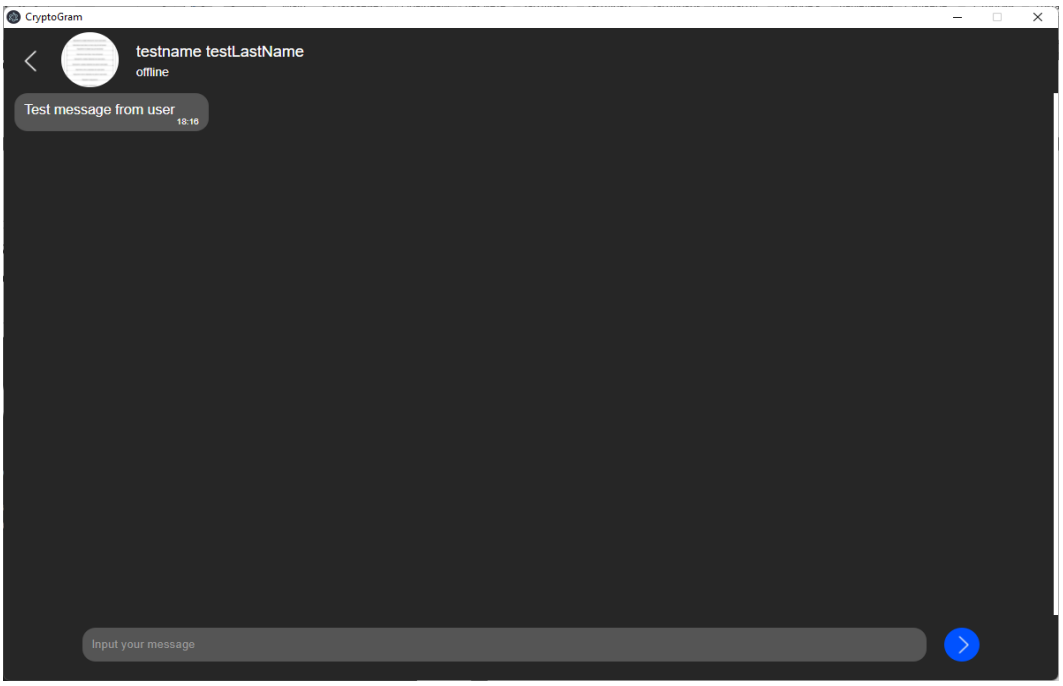


Рисунок 4.33 – Діалог з користувачем з аккаунту користувача Volodymyr nickname pilya

4.8 Висновки

В даному розділі було проведено аналіз існуючих методів тестування та було проведено ручне тестування. Ручне тестування включало в себе перевірку основних функцій: реєстрація користувача, перший та другий етапи авторизації, відправка та отримання повідомлень, зміна та видалення фотографії профілю а також пошук користувачів по нікнейму. Дані тестування показали, що додаток працює коректно при правильних і неправильних даних.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу впровадження розробки програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів: Майданюк В.П., ВНТУ, Катєльніков Д.І., ВНТУ, Стахов А.Ю., ASTA.MOBI.

Технологічний аудит проводився з використанням таблиці 5.1 [25].

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 5.1

Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки.

Критерії	Прізвище, ініціали, посада експерта		
	Майданюк В.П., ВНТУ	Катєльніков Д.І., ВНТУ	Стахов А.Ю., ASTA.MOBI
	Бали, виставлені експертами:		
1	3	3	4
2	2	2	2
3	4	4	4
4	3	3	2
5	3	3	4
6	4	4	4
7	2	2	2
8	4	3	4
9	1	1	1
10	4	4	4
11	4	4	4
12	4	4	4
Сума балів	$СБ_1$=38	$СБ_2$=37	$СБ_3$=39
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{38+37+39}{3} = 38$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 38 балів, що згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} * t \text{ (грн)} \quad (5.1)$$

де $М$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_Р$ – число робочих днів в місяці; приблизно $Т_Р \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Для розробки методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування необхідно залучити інженера програміста з посадовим окладом 9000 грн. Кількість робочих днів у місяці складає 22, а кількість робочих днів програміста складає 21. Зведемо сумарні розрахунки до таблиця 5.4.

Таблиця 5.4 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	15000	681,8	5	3409
Інженер-програміст	9000	409,1	21	8591
Всього				12000

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (5.2)$$

$$З_д = 0{,}11 * 12000 = 1320 \text{ (грн)}$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.3):

$$Н_{ЗП} = (З_о + З_д) * \frac{\beta}{100} \text{ (грн)} \quad (5.3)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітни

ків, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді**:**

$$Н_{ЗП} = (12000 + 1320) * \frac{22}{100} = 2930{,}4 \text{ (грн)}$$

4. Витрати на матеріали М та комлектуючі К, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн.,} \quad (5.4)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Витрати на комплектуючі вироби, які використовують при виготовленні одиниці продукції, розраховуються, згідно їх номенклатури, за формулою:

$$К = \sum_{i=1}^{n} Н_i \cdot Ц_i \cdot К_i, \quad (5.5)$$

де $Н_i$ – кількість комплектуючих *і*-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих *і*-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Інформацію про використані матеріалита комлектуючі подамо у вигляді табл. 5.5.

Таблиця 5.5 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	125	1	125
Ручка	14	1	14
CD-диск	12	1	12
Флешка	140	1	140
Всього			291
З врахуванням коефіцієнта транспортування			320,1

5. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{5.6}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункта 137.3.3 Податкового кодекса амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувався персональний комп'ютер вартістю 10000 грн.

$$А = \frac{22000 \cdot 1}{2 \cdot 12} = 916{,}67$$

6. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \tag{5.7}$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}3 \cdot 155 \cdot 4{,}1 \cdot 0{,}5}{0{,}8} = 119{,}15$$

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати

Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.8)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 12000 \cdot \frac{100}{100\%} = 12000 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В = 120000+1320+2930,4+320,1+916,67+119,15+12000=29606,1 \text{ грн.}$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (5.9)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,7.

Звідси:

$$ЗВ = \frac{29606,1}{0,7} = 42294,75 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \quad (5.10)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2021 році – 18%.

Прогнозується, що прибуток ми будемо отримувати за рахунок реклами від впроваджених результатів нашої розробки. Припустимо, що при впроваджені результатів наукової розробки покращується якість, що дозволяє підвищити ціну на рекламу на 150 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 1000 шт., протягом другого року – на 2000 шт., протягом третього року на 3000 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна 200 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [150 \cdot 1 + (200 + 150) \cdot 1000] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 59814{,}89 \text{ грн.}$$

$$\Delta П_2 = [150 \cdot 1 + (200 + 150) \cdot (1000 + 2000)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 179517{,}83 \text{ грн.}$$

$$\Delta П_З = [150 \cdot 1 + (200 + 150) \cdot (1000 + 2000 + 3000)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 358885{,}65 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \tag{5.11}$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 2 \cdot 42294{,}75 = 84589{,}49$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \tag{5.12}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{5.13}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$ПП = \frac{59814,89}{(1+0,2)^1} + \frac{179517,83}{(1+0,2)^2} + \frac{358885,65}{(1+0,2)^3} = 383165,35 \text{ грн.}$$

$$Е_{абс} = (383165,35 - 84589,49\ ) = 298575,86 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (5.14)$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{298575,86}{84589,49}} - 1 = 1 = 100\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \qquad (5.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0,18 + 0,05 = 0,23.$$

Так як $E_в > \tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \tag{5.16}$$

$$\mathrm{T_{ок}} = \frac{1}{1} = 1 \text{ рік}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

5.5 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу розробки програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування, який є на вище середньому рівні.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 29606,1 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 42294,75 грн.

Вкладені інвестиції в даний проект окупляться через 1 рік при прогнозованому прибутку 383165,35 грн. за три роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі був розроблений програмний додаток чату з використанням асиметричного шифрування на платформі NodeJS з використанням мови JavaScript.

Виконано аналіз основних аналогів, виявлено їх переваги та недоліки. На основі отриманих результатів було прийнято рішення про створення власного додатку, який би вирішував проблеми що присутні у аналогів.

Також в результаті проведеного варіантного аналізу було обрано мову програмування JavaScript та програмну платформу NodeJS для серверної та клієнтської частини та середовище розробки Intellij WebStorm.

Було вирішено наступні задачі:

- проведено аналіз існуючих методів і засобів захисту інформаційної системи месенджеру з використанням асиметричного шифрування;
- запропоновано новий метод використання асиметричного шифрування для захисту інформаційної системи месенджеру;
- розроблено програмні компоненти та систему захисту інформаційної системи месенджеру з використанням асиметричного шифрування на основі запропонованих методів;
- проведено експериментальні дослідження розроблених засобів захисту інформаційної системи месенджеру.

Було розроблено алгоритми шифрування повідомлення, розшифрування повідомлень, алгоритми зміни та видалення фото профілю користувача.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню.

Магістерську кваліфікаційну роботу було оформлено відповідно до встановлених вимог.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Д.І. Катєльніков, В.Д. Пілецький – ВИКОРИСТАННЯ АСИМЕТРИЧНОГО ШИФРУВАННЯ ДЛЯ ЗАХИСТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ МЕСЕНДЖЕРУ. Збірник тез доповідей Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», Вінниця, 2021. –С.154-156.

2. Instant messaging [Електронний ресурс] – Режим доступу: shorturl.at/yFJLY.

3. Системи миттєвого обміну повідомленнями [Електронний ресурс] – Режим доступу: shorturl.at/fkqJ0.

4. Telegram [Електронний ресурс] – Режим доступу: shorturl.at/kpFL9.

5. Viber [Електронний ресурс] – Режим доступу: shorturl.at/mxzDZ.

6. WhatsApp [Електронний ресурс] – Режим доступу: shorturl.at/cprsS.

7. Шифрування [Електронний ресурс] – Режим доступу: shorturl.at/ruRV8.

8. Симетричне шифрування [Електронний ресурс] – Режим доступу: shorturl.at/yBHO0.

9. Асиметричне шифрування [Електронний ресурс] – Режим доступу: shorturl.at/jyGJ7.

10. Переваги та недоліки симетричного та асиметричного типів шифрування [Електронний ресурс] – Режим доступу: shorturl.at/iosPY

11. Інформаційні системи і технології в статистиці. [Електронний ресурс] – Режим доступу: https://bit.ly/2FPN1Mc.

12. RSA [Електронний ресурс] – Режим доступу: shorturl.at/vyBW4.

13. React [Електронний ресурс] – Режим доступу: shorturl.at/bdwGT.

14. MobX [Електронний ресурс] – Режим доступу: shorturl.at/dkFOR.

15. PHP [Електронний ресурс] – Режим доступу: shorturl.at/nFHLT.

16. JavaScript [Електронний ресурс] – Режим доступу: shorturl.at/ksCT1.

17. NodeJS [Електронний ресурс] – Режим доступу: shorturl.at/jnrPT.

18. Ruby [Електронний ресурс] – Режим доступу: shorturl.at/cuRU9.

19. Electron [Електронний ресурс] – Режим доступу: shorturl.at/beDKO.

20. Chromium [Електронний ресурс] – Режим доступу: shorturl.at/bkxCN.

21. Microsoft Visual Studio [Електронний ресурс] – Режим доступу: shorturl.at/puL17.

22. Atom [Електронний ресурс] – Режим доступу: shorturl.at/vBN05.

23. Intellij WebStorm [Електронний ресурс] – Режим доступу: shorturl.at/dtBPZ.

24. Тестування програмного забезпечення shorturl.at/jnIQY.

25. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

26. Д.І. Катєльніков, В.Д. Пілецький – ВПРОВАДЖЕННЯ НОВИХ ТЕХНОЛОГІЙ JAVA В НАВЧАЛЬНИЙ ПРОЦЕС. Збірник тез доповідей Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», Вінниця, 2021. –С.74-77.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"__" _____ 2021 р.

Технічне завдання
на магістерську кваліфікаційну роботу
«Розробка методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____к.т.н., доцент кафедри ПЗ, Катєльніков Д.І.
«_» _____ 2021 р.
Виконав:
_____студент гр. 1ПІ-20м, Пілецький В.Д.
«_» _____ 2021 р.

ВНТУ – 2021

Додаток Б. Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Розробка методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 20м

Науковий керівник: к.т.н. доц. Катєльніков Д.І.

Unicheck	
Оригінальність	**90,8%**
Схожість	9,2 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлена з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування».

Автор _____ Пілецький Володимир Данилович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток В. Лістинг коду серверної частини

```
const
    mongoose = require('./Config/database'),
    Logs = mongoose.model('Logs'),
    {queryParser} = require('express-query-parser'),
    fileUpload  = require('express-fileupload'),
    crypto = require('crypto'),
    path = require('path'),
    socketIo = require('socket.io'),
    express = require('express'),
    http = require('http'),
    config = require('./Config/config.json'),
    logger = require('morgan'),
    cors = require('cors'),
    bodyParser = require('body-parser'),
    port = config.port,
    app = express(),
    server = http.createServer(app);

const io = socketIo(server, {
    path: "/socket.io"
});

require('./Socket/socket')(io);

app.io = io;

app.use(fileUpload({
    createParentPath: true,
    parseNested: true
}));

app.use('/uploads', express.static('./uploads'));

app.use(bodyParser.urlencoded({extended: false, limit: '100mb'}));
app.use(bodyParser.json({limit: '100mb'}));
app.use(bodyParser.text());

app.use(logger('dev'));

app.use(cors());

app.get('/', (req, res) => {
    res.send('App is working');
})

app.use('/', require('./Routes/index'));

app.use(
    queryParser({
        parseNull: true,
        parseBoolean: true
    })
)

app.all('*', (req, res) => {
    res.status(404).send({
        err: "Path or method are wrong"
    });
```

```
});

app.use(async function (err, req, res, next) {
    console.log("=========================ERROR IN ROUTE=========================");
    console.error(err.stack);
    let status = err.status || 500;
    await Logs.create({message: err.message, status: err.status, body: err.body});
    res.status(status);
    res.json({
        status: status,
        message: err.message,
        code: err.code,
        errorBody: err.errorBody
    });
    console.log("================================================================");
});

server.listen(port, () => {
    console.log(`Server has started on port: ${port}`);
});
module.exports = class Error {
    constructor(code, field, message) {
        this.code = code;
        this.field = field;
        this.message = message;
    }
}
const
    moment = require('moment'),
    emailValidator = require('email-validator'),
    {passwordValidation} = require('../Utils/utils'),
    Error = require('./Error');

// ERROR CONSTANTS

const
    CODE_OPTIONAL_ERROR = 'REQUIRED_ERROR',
    CODE_TYPE_ERROR = 'INVALID_ERROR';

function fieldsValidator (params) {
    let errors = [], obj = {};
    for (let param of params) {
        let isError = false;
        if ((param.value == null || param.value === '') && param.optional === false) {
            errors.push(new Error(CODE_OPTIONAL_ERROR, param.name, `FIELD ${param.name}
IS REQUIRED`));
            continue;
        }

        if ((param.value === undefined || param.value === null) && param.optional ===
true) {continue;}

        if (param.allowedValues) {
            if (param.allowedValues.indexOf(param.value) === -1) {
                errors.push(new Error(CODE_TYPE_ERROR, param.name, `FIELD ${param.name}
IS INVALID`));
                continue;
            }
        }

        switch (param.type) {
            case 'arrayNumber': {
```

```
    for (let el of param.value) {
        if (typeof +el !== 'number') {
            isError = true;
        }
    }
    break;
}
case 'arrayString': {
    for (let el of param.value) {
        if (typeof el !== 'string') {
            isError = true;
        }
    }
    break;
}
case 'error': {
    isError = true;
    break;
}
case 'arrayToStr': {
    param.value = param.value.join(',');
    break;
}
case 'email': {
    if (!emailValidator.validate(param.value)) {
        isError = true;
    }
    break;
}
case 'password': {
    if (!passwordValidation(param.value)) {
        isError = true;
    }
    break;
}
case 'string': {
    if (typeof param.value !== 'string') {
        isError = true;
    }
    break;
}
case 'number': {
    console.log(isNaN(param.value))
    if (isNaN(param.value)) {
        isError = true;
    }
    break;
}
case 'date': {
    if (!moment(param.value).isValid()) {isError = true;}
    break;
}
case 'boolean': {
    if (typeof param.value !== 'boolean') {isError = true;}
    break;
}
case 'file': {
    if (!param.value) isError = true;
    break;
}
case 'searchString': {
    if (param.value !== null) {
```

```
            if (typeof param.value !== "string") isError = true;

            obj[param.dbName] = {$regex: new RegExp(param.value), $options: "i"}
        }

        break;
    }
    case 'searchNumber': {
        if (param.value !== null) {
            let splattedArray = param.value.split(',');

            if (splattedArray.length === 0) break;

            for (let el of splattedArray) {
                if (isNaN(el)) isError = true;
            }

            obj[param.dbName] = {$in: splattedArray};
        }

        break;
    }
    case 'searchDate': {
        if (param.value !== null) {
            let splattedArray = param.value.split(',');

            if (splattedArray.length === 0) break;

            obj[param.dbName] = {
                $gte: splattedArray[0],
                $lte: splattedArray[1]
            }
        }

        break;
    }
    case 'searchBool': {
        if (param.value !== null) {
            let splattedArray = param.value.split(',');

            if (splattedArray.length === 0) break;

            obj[param.dbName] = splattedArray.map(el => {
                return {$eq: el}
            });
        }

        break;
    }
    case 'messageObj': {
        for (let prop in param.value) {
            if (typeof prop !== 'string') {
                isError = true;
                break;
            }

            let {message, files} = param.value[prop];

            if (!message && !files) isError = true;

            if (message) {
                if (!(message instanceof Array)) {
```

```
                            isError = true;
                            break;
                        }

                        for (let el of message) {
                            if (typeof el !== 'string') {
                                isError = true;
                                break;
                            }
                        }
                    }

                    if (files) {
                        if (!(files instanceof Array)) {
                            isError = true;
                            break;
                        }

                        if (!isError) {
                            for (let el of files) {
                                let {fileName, value} = el;
                                if (typeof fileName !== 'string') isError = true;

                                if (!(value instanceof Array)) {
                                    isError = true;
                                    break;
                                }

                                if (!isError) {
                                    for (let part of value) {
                                        if (typeof part !== 'string') isError = true;
                                    }
                                }
                            }
                        }
                    }

                    break;
                }
            }

            if (isError) {
                errors.push(new Error(CODE_TYPE_ERROR, param.name, `FIELD ${param.name} IS
INVALID`))
            } else if (param.value != null && param.value !== '' && param.type !==
'searchString' && param.type !== 'searchNumber' && param.type !== 'searchDate' &&
param.type !== 'searchBool' && param.dbName != null) {
                obj[param.dbName] = param.value;
            }
        }

        if (errors.length > 0) console.log(errors);

        return {
            errors: errors,
            obj: obj
        }
    }

    module.exports = fieldsValidator;
    const
```

```
    moment = require('moment'),
    mongoose = require('../Config/database'),
    Message = mongoose.model('Message'),
    User = mongoose.model('User');

const fs = require('fs');

function normalizeTimeOutput (value) {
    if ((value + "").length < 2) return "0" + value;
    else return value;
}

function timeOutput(time) {
    return         normalizeTimeOutput(time.getHours())         +         ":"         +
normalizeTimeOutput(time.getMinutes());
}

function dateOutput(date) {

}

function dateTimeOutput (dateTime) {

}

function userObj (data) {
    return {
        id: data._id,
        email: data.email,
        name: data.name,
        lastName: data.lastName,
        middleName: data.middleName,
        nickname: data.nickname,
        avatar: data.avatar,
        role: data.role,
        onlineStatus: data.onlineStatus,
        status: data.status,
        profileDescription: data.profileDescription,
        frozenUntil: data.frozenUntil,
        createdAt: moment(data.createdAt).format("YYYY-MM-DD hh:mm:ss"),
        updatedAt: moment(data.updatedAt).format("YYYY-MM-DD hh:mm:ss"),
        publicKey: data.publicKey
    };
}

async function roomObj (data, user) {
    let resultObj = {
        id: data._id,
        lastMessage: data.lastMessage ? messageObj(data.lastMessage, user) : null,
        users: data.users,
        createdAt: moment(data.createdAt).format("YYYY-MM-DD hh:mm:ss"),
        updatedAt: moment(data.updatedAt).format("YYYY-MM-DD hh:mm:ss")
    }

    let otherUserId = user === data.users[0].id ? data.users[1].id : data.users[0].id;

    resultObj.otherUser = userObj(await User.findById(otherUserId).lean().exec());

    return resultObj;
}

function messageObj (message, user) {
```

```
    let createdAtTime = new Date(message.createdAt);
    return {
        id: message._id,
        sender: message.sender,
        room: message.room,
        messageObj: message.messageObj[user],
        createdAt: timeOutput(createdAtTime)
    };
}

module.exports = {
    userObj,
    roomObj,
    messageObj
}
function generateRoomKey (users) {
    return Buffer.from(users.join(',')).toString('base64');
}

module.exports = {
    generateRoomKey
}
const
    jwt = require('jsonwebtoken'),
    config = require('../Config/config.json');

async function checkTokenForSocket(socket, next) {
    let token = socket.handshake.query.token;

    if (token) {
        jwt.verify(token, config["jwt-token"],function (err, decoded) {
            if (err) {
                next(new Error('Invalid token'));
            } else {
                socket.user = decoded;
                next();
            }
        });
    } else {
        let err = new Error('No authorization token was found');
        next(err);
    }
}

module.exports = {
    checkTokenForSocket
}
const
    jwt = require('jsonwebtoken'),
    config = require('../Config/config.json');

const handle = (promise) => {
    return promise
        .then(data => ([data, undefined]))
        .catch(error => Promise.resolve([undefined, error]));
}

function generateToken(userId, email, userRole) {
    return   jwt.sign({id:   userId.toString(),   email:   email,   userRole:   userRole},
config["jwt-token"]);
}
```

```
function generateCode() {
    let min = 100000, max = 999999;
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

function hasPasswordUpperCase (string) {
    for (let el of string) {
        if (el === el.toUpperCase() && isNaN(+el)) {
            return true;
        }
    }

    return false;
}

function hasPasswordLowerCase (string) {
    for (let el of string) {
        if (el === el.toLowerCase()) {
            return true;
        }
    }

    return false;
}

function hasPasswordNumber (string) {
    let hasNumber = /\d/;
    return hasNumber.test(string);
}

function passwordValidation (string) {
    if (string.length < 8) {
        return false;
    }

    return    hasPasswordLowerCase(string)    &&    hasPasswordUpperCase(string)    &&
hasPasswordNumber(string);
}

function searchParams (req) {
    let limit = 10,
        offset = 0,
        sortField = "_id",
        sortType = 0;

    if (req.query.sortType !== undefined) {
        switch (req.query.sortType) {
            case '1':
            case 'asc':
            case "ASC": {
                sortType = 0;
                break;
            }
            case '0':
            case 'desc':
            case "DESC": {
                sortType = 1;
                break;
            }
        }
```

```
    }

    if (req.query.sortField !== undefined) {
        sortField = req.query.sortField;
    }

    if (req.query.limit !== undefined && !isNaN(req.query.limit)) {
        limit = req.query.limit;
    }

    if (req.query.offset !== undefined && !isNaN(req.query.offset)) {
        offset = req.query.offset;
    }

    return {limit, offset, sortField, sortType};
}

module.exports = {
    handle,
    generateCode,
    generateToken,
    passwordValidation,
    searchParams
}
const
    {checkTokenForSocket}       = require('../Utils/socket'),
    mongoose                    = require('../Config/database'),
    User                        = mongoose.model('User'),
    Room                        = mongoose.model('Room');

module.exports = io => {

    io.use(checkTokenForSocket);

    io.on('connection', async socket => {

        socket.join(socket.user.id);

        console.log(socket.user.id + " connected");

        socket.on('joinRoom', (data) => {
            console.log('user joined');
            socket.join(data);
        })

        socket.on('leaveRoom', (data) => {
            console.log('user left');
            socket.leave(data);
        })

        socket.on('disconnect', async () => {
            socket.leave(socket.user.id);
            await User.findByIdAndUpdate(socket.user.id, {onlineStatus: 'offline'});
        })
    });
};
const
    router = require('express').Router(),
    {auth, sendCode} = require('../Controllers/auth');

router.post('', auth);
router.post('/send-code', sendCode);
```

```
module.exports = router;
const
    router = require('express').Router(),
    jwt = require('../Config/jwt');

router.use(jwt.unless({
    path: [
        /\/uploads/,
        /\/auth/,
        {url: /\/user/, methods: ['POST']}
    ]
}));

router.use('/user',             require('./user'));
router.use('/auth',             require('./auth'));
router.use('/room',             require('./room'));
router.use('/message',          require('./message'));

module.exports = router;
const router = require('express').Router();
const {createMessage, changeMessage, deleteMessage, getMessagesList, messageDetail,
getMessagesFromRoom} = require('../Controllers/message');

router.get('',              getMessagesList);
router.get('/:id',          messageDetail);
router.get('/:id/room',     getMessagesFromRoom);
router.post('',             createMessage);
router.patch('/:id',        changeMessage);
router.delete('/:id',       deleteMessage);

module.exports = router;
const
    router = require('express').Router(),
    {getRoomsList, createRoom, roomDetail, deleteRoom, updateUsersList, getRoomWithUser}
= require('../Controllers/room');

router.get('/:id/user', getRoomWithUser);
router.get('', getRoomsList);
router.post('', createRoom);
router.get('/:id', roomDetail);
router.patch('/:id', updateUsersList);
router.delete('/:id', deleteRoom);

module.exports = router;
const
    router = require('express').Router(),
    {getUsersList, getUserInfo, createUser, updateUser, deleteUser, blockUnblockUser,
deleteImage, loadImage, changeEmail, changeNickname} = require('../Controllers/user');

router.get('', getUsersList);
router.get('/:id', getUserInfo);
router.post('', createUser);
router.patch('/:id', updateUser);
router.delete('/:id', deleteUser);
router.patch('/:id/block-unblock', blockUnblockUser);

router.patch('/:id/email', changeEmail);
router.patch('/:id/nickname', changeNickname);

router.route('/:id/image')
    .patch(loadImage)
```

```
    .delete(deleteImage);

module.exports = router;
const mongoose = require('../Config/database');
const User = mongoose.model('User');
const {handle} = require('../Utils/utils');

const
    ACCESS_DENIED_ERROR = 'ACCESS DENIED',
    USER_NOT_FOUND_ERROR = 'USER WAS NOT FOUND',
    USER_NOT_INITIALIZED = 'USER IS NOT INITIALIZED',
    ENTITY_NOT_INITIALIZED = 'ENTITY IS NOT INITIALIZED'

class Rights {
    id;
    user;
    entity;
    status;

    constructor(id, user, entity) {
        this.id = id;
        this.user = user;

        this.entity = entity;

        this.status = user ? user.role : null;
    }

    async initializeUser() {
        let [user, userError] = await handle(User.findById(this.id).lean().exec());

        if (userError) throw Error(userError);

        if (!user) throw Error(USER_NOT_FOUND_ERROR);

        this.user = user;
        this.status = user.role;
    }

    async whatCanRead(ids) {
        let result = [];
        if (this.user.role === 'user') {
            let [unhiddenUsers, unhiddenUsersError] = await handle(User.find({hidden:
false}).lean().exec());

            if (unhiddenUsersError) return {message: unhiddenUsersError, status: 500,
can: false}

            result = unhiddenUsers.map(el => el._id);

            if (ids && ids.length > 0) {
                let checked = true;

                ids.forEach(el => {
                    if (result.indexOf(el) === -1) checked = false;
                })

                if (!checked) {
                    return null;
                } else {
                    return ids;
                }
```

```
            } else {
                return result;
            }
        } else {
            return ids;
        }
    }

    async checkRights(field) {
        if (!this.user) return {
            status: 500,
            can: false,
            message: USER_NOT_INITIALIZED
        }

        if (!this.entity) return {
            status: 500,
            can: false,
            message: ENTITY_NOT_INITIALIZED
        }

        switch (field) {
            case 'read': {
                if (this.user.role === 'user' && this.entity.hidden && this.id !==
this.entity.id) return {
                    status: 403,
                    can: false,
                    message: ACCESS_DENIED_ERROR
                }
                else return {status: 200, can: true, message: null}
            }
            case 'delete':
            case 'change': {
                if (this.user.role === 'user' && this.user._id !== this.entity._id) {
                    return {
                        can: false,
                        status: 403,
                        message: ACCESS_DENIED_ERROR
                    }
                } else return {
                    can: true,
                    status: 200,
                    message: null
                }
            }
            case 'block':
            case 'unblock': {
                let can = this.user.role === 'admin';

                return {
                    can,
                    status: can ? null : 403,
                    message: ACCESS_DENIED_ERROR
                }
            }
        }
    }
}

module.exports = Rights;
```

```
const
    mongoose = require('mongoose'),
    Schema = mongoose.Schema;

const Logs = new Schema({
    message: String,
    body: String,
    status: Number
}, {
    timestamps: true,
    collection: 'Logs'
});

module.exports = Logs;
const
    mongoose = require('mongoose'),
    Schema = mongoose.Schema;

const Message = new Schema({
    sender: {type: mongoose.Schema.ObjectId, ref: 'User'},
    readBy: [{type: mongoose.Schema.ObjectId, ref: 'User'}],
    room: {type: mongoose.Schema.ObjectId, ref: 'Room', required: true},
    messageObj: Schema.Types.Mixed
}, {
    timestamps: true,
    collection: 'Message'
});

module.exports = Message;
const mongoose = require('mongoose'),
    Schema = mongoose.Schema;

const userSchema = new Schema({
    id: {type: String},
    publicKey: {type: String}
}, {
    _id: false
})

const Room = new Schema({
    users: [userSchema],
    lastMessage: {type: Object}
}, {
    timestamps: true,
    collection: 'Room'
})

module.exports = Room;
const
    mongoose = require('mongoose'),
    Schema = mongoose.Schema;

const User = new Schema({
    name: {type: String, required: true},
    lastName: {type: String, required: true},
    email: {type: String, required: true},
    code: {type: String},
    codeExpiresIn: {type: Date},
    onlineStatus: {type: String, enum: ['online', 'offline']},
    lastActive: {type: Date, default: null},
    avatar: {type: String, default: null},
    nickname: {type: String, required: true},
```

```
    role: {type: String, enum: ['admin', 'user', 'operator'], default: 'user'},
    status: {type: String, enum: ['blocked', 'normal', 'frozen']},
    frozenUntil: {type: Date, default: null},
    profileDescription: String,
    hidden: {type: Boolean},
    publicKey: {type: String}
}, {
    timestamps: true,
    collection: 'User'
});

module.exports = User;
class ValidationField {
    constructor(name, value, type, optional, dbName, allowedValues) {
        this.name = name;
        this.value = value;
        this.type = type;
        this.optional = optional;
        this.dbName = dbName;
        this.allowedValues = allowedValues;
    }
}

module.exports = ValidationField;
const
    mongoose = require('../Config/database'),
    sendMail = require('../Config/nodemailer'),
    config = require('../Config/config.json'),

    User = mongoose.model('User'),

    moment = require('moment'),

    ValidationField = require('../Models/validationField'),

    FieldsValidator = require('../Utils/fieldsValidator'),
    {userObj} = require('../Utils/modelObjects'),
    {handle, generateCode, generateToken} = require('../Utils/utils');

async function sendCode(req, res) {
    /* fields
        * email
    */

    let {email} = req.body;

    let params = [
        new ValidationField('email', email, 'email', false, 'email')
    ]

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({errors: errors});
    }

    let [user, userError] = await handle(User.findOne(obj).lean().exec());

    if (userError) {
        return res.status(403).send({
            error: userError
        })
```

```
    }

    if (!user) {
        return res.status(404).end();
    }

    let code = generateCode(),
        codeExpiresIn    =    moment(moment().valueOf()    +    600000).format('YYYY-MM-DD
HH:mm:ss');

    let [updatedUser, updatedUserError] = await handle(User.findOneAndUpdate(obj, {
        code: code,
        codeExpiresIn: codeExpiresIn
    }));

    if (updatedUserError) {
        return res.status(403).send({
            error: updatedUserError
        })
    }

    let   [sendEmail,   sendEmailError]   =   await   handle(sendMail(config.email.sender,
user.email, 'Secure chat authorization code', 'Your chat auth code is -> ' + code));

    if (sendEmailError) {
        return res.status(403).send({
            error: sendEmailError
        })
    }

    return res.status(200).end();
}

async function auth(req, res) {
    /* fields
        * email
        * code
    */

    let {email, code} = req.body;

    let params = [
        new ValidationField('email', email, 'email', false, 'email'),
        new ValidationField('code', code, 'number', false, 'code')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({errors: errors});
    }

    let     [findUser,     findUserError]     =     await     handle(User.findOne({email:
obj.email}).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (!findUser) {
```

```
        return res.status(404).end();
    }

    if (+findUser.code !== +obj.code && obj.code + "" !== "111111") {
        return res.status(401).end();
    }

    if (moment(findUser.codeExpiresIn).valueOf() < moment().valueOf()) {
        return res.status(401).end();
    }

    let [userAuthUpdate, userAuthUpdateError] = await handle(User.findByIdAndUpdate(findUser._id, {
        code: null,
        codeExpiresIn: null
    }));

    if (userAuthUpdateError) {
        return res.status(403).send({
            error: userAuthUpdateError
        })
    }

    res.status(200).json({
        id: userAuthUpdate._id,
        key: findUser.email,
        token: generateToken(findUser._id, findUser.email, findUser.role)
    });
}

module.exports = {
    sendCode,
    auth
}
const {searchParams, handle} = require('../Utils/utils');
const mongoose = require('../Config/database');
const Message = mongoose.model('Message');
const ValidationField = require("../Models/validationField");
const FieldsValidator = require('../Utils/fieldsValidator');
const {messageObj} = require('../Utils/modelObjects');
const Room = mongoose.model('Room');

async function getMessagesList(req, res) {
    let user = req.user.id;

    let params = [
        new ValidationField('id', req.query.id, 'searchString', true, '_id'),
        new ValidationField('sender', req.query.sender, 'searchString', true, 'sender'),
        new ValidationField('room', req.query.room, 'searchString', true, 'room'),
        new ValidationField('forUser', req.query.user, 'string', true, 'forUser')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    let [messages, messagesError] = await handle(Message.find(obj).lean().exec());

    if (messagesError) return res.status(500).send(messagesError);

    let formattedMessages = messages.map(el => {
        return messageObj(el, user)
```

```
    })

    return res.status(200).send(formattedMessages);
}

async function getMessagesFromRoom(req, res) {
    let user = req.user.id;

    let params = [
        new ValidationField('room', req.params.id, 'string', false, 'room')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    let [messages, messagesError] = await handle(Message.find(obj).lean().exec());

    if (messagesError) return res.status(500).send(messagesError);

    let formattedMessages = messages.map(el => {
        return messageObj(el, user)
    })

    return res.status(200).send(formattedMessages);
}

async function createMessage(req, res) {
    let user = req.user.id;
    let io = req.app.io;

    let params = [
        new ValidationField('room', req.body.room, 'string', false, 'room'),
        new  ValidationField('messageObj',  req.body.messageObj,  'messageObj',  false,
'messageObj')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    let [room, roomError] = await handle(Room.findById(req.body.room).lean().exec());

    if (roomError) return res.status(500).send(roomError);

    if (!room) return res.status(404).end();

    obj['sender'] = user;
    obj['readBy'] = [user];

    let [createdMessage, createdMessageError] = await handle(Message.create(obj));

    if (createdMessageError) return res.status(500).send(createdMessageError);

    let   [updatedRoom,   updatedRoomError]   =   await   handle(Room.updateOne({_id:
req.body.room}, {
        lastMessage: createdMessage
    }));

    if (updatedRoomError) return res.status(500).send(updatedRoomError);

    for (let el of room.users) {
```

```
        io.to(el.id).emit('new-message',      JSON.stringify(messageObj(createdMessage,
el.id)));
        io.to(`${room._id}${el.id}`).emit('new-room-message',
JSON.stringify(messageObj(createdMessage, el.id)));
    }

    return res.status(200).send(messageObj(createdMessage, user));
}

async function changeMessage(req, res) {

}

async function deleteMessage(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false)
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    let             [message,             messageError]             =             await
handle(Message.findById(req.params.id).lean().exec());

    if (messageError) return res.status(500).send(messageError);

    if (!message) return res.status(404).end();

    let  [deleteMessage,  deleteMessageError]  =  await  handle(Message.deleteOne({_id:
req.params.id}));

    if (deleteMessageError) return res.status(500).send(deleteMessageError);

    return res.status(200).end();
}

async function messageDetail(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false)
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    let [message, messageError] = await handle(Message.findById(req.params.id));

    if (messageError) return res.status(500).send(messageError);

    if (!message) return res.status(404).end();

    return res.status(200).send(messageObj(message, req.user.id));
}

module.exports = {
    getMessagesList,   createMessage,   changeMessage,   deleteMessage,   messageDetail,
getMessagesFromRoom
}
const
    {searchParams} = require('../Utils/utils'),
    mongoose = require('../Config/database'),
```

```
    sendMail = require('../Config/nodemailer'),
    config = require('../Config/config.json'),

    User = mongoose.model('User'),
    Room = mongoose.model('Room'),
    Message = mongoose.model('Message'),

    moment = require('moment'),
    fsx = require('fs-extra'),
    path = require('path'),
    md5 = require('md5'),

    ValidationField = require('../Models/validationField'),

    FieldsValidator = require('../Utils/fieldsValidator'),
    {roomObj} = require('../Utils/modelObjects'),
    {handle, generateCode} = require('../Utils/utils'),
    {generateRoomKey} = require('../Utils/room');

async function getRoomsList(req, res) {
    let {limit, offset, sortField, sortType} = searchParams(req);

    let params = [];

    let {errors, obj} = FieldsValidator(params);

    obj['users.id'] = req.user.id;

    if (errors.length > 0) return res.status(400).send(errors);

    let [foundRooms, foundRoomsError] = await
handle(Room.find(obj).sort([['lastMessage.createdAt', -1]]).lean().exec());

    if (foundRoomsError) return res.status(500).send(foundRoomsError);

    let [totalCount, totalCountError] = await handle(Room.countDocuments(obj));

    if (totalCountError) return res.status(500).send(totalCountError);

    let result = [];

    for await (let el of foundRooms) {
        result.push(await roomObj(el, req.user.id));
    }

    return res.status(200).send({
        data: result,
        totalCount
    });
}

async function createRoom(req, res) {
    let params = [];

    if (req.body.users) {
        for (let user of req.body.users) {
            params.push(
                new ValidationField('id', user.id, 'string', false),
                new ValidationField('publicKey', user.publicKey, 'string', false)
            )
        }
    } else {
```

```
        return res.status(400).send({
            errors: ["FIELD USERS IS REQUIRED"]
        })
    }

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    obj['users'] = req.body.users;

    let [createdRoom, createdRoomError] = await handle(Room.create(obj));

    if (createdRoomError) return res.status(500).send(createdRoomError);

    return res.status(200).send(roomObj(createdRoom, req.user.id));
}

async function roomDetail(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false)
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(500).send(errors);

    let [room, roomError] = await handle(Room.findById(req.params.id).lean().exec());

    if (roomError) return res.status(500).send(roomError);

    if (!room) return res.status(404).end();

    return res.status(200).send(await roomObj(room, req.user.id));
}

async function deleteRoom(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false)
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    let [room, roomError] = await handle(Room.findById(req.params.id));

    if (roomError) return res.status(500).send(roomError);

    if (!room) return res.status(404).end();

    let [deleteRoom, deleteRoomError] = await
handle(Room.findByIdAndDelete(req.params.id));

    if (deleteRoomError) return res.status(500).send(deleteRoomError);

    return res.status(200).end();
}

async function updateUsersList(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false)
```

```
    ];

    if (req.body.users) {
        for (let user of req.body.users) {
            params.push(
                new ValidationField('id', user.id, 'string', false),
                new ValidationField('publicKey', user.publicKey, 'string', false)
            )
        }
    } else {
        return res.status(400).send({
            errors: ["FIELD USERS IS REQUIRED"]
        })
    }

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    obj['users'] = req.body.users;

    let [room, roomError] = await handle(Room.findById(req.params.id));

    if (roomError) return res.status(500).send(roomError);

    if (!room) return res.status(404).end();

    let [updateRoom, updateRoomError] = await handle(Room.findByIdAndUpdate(req.params.id, obj));

    if (updateRoomError) return res.status(500).send(updateRoomError);

    return res.status(200).end();
}

async function getRoomWithUser(req, res) {
    let user = req.user.id;

    let params = [
        new ValidationField('userId', req.params.id, 'string', false)
    ];

    let {obj, errors} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    // find info about both users

    let [users, usersError] = await handle(User.find({$or: [{_id: user}, {_id: req.params.id}]}).lean().exec());

    if (usersError) return res.status(500).send(usersError);

    let
        searchArr = [{
            id: users[0]._id + "",
            publicKey: users[0].publicKey
        }, {
            id: users[1]._id + "",
            publicKey: users[1].publicKey
        }],
        reversedSearchArr = [{
```

```
            id: users[1]._id + "",
            publicKey: users[1].publicKey
        }, {
            id: users[0]._id + "",
            publicKey: users[0].publicKey
        }];

    if (users.length !== 2) return res.status(500).end();

    let [room, roomError] = await handle(Room.findOne({
        $or: [
            {
                users: searchArr
            }, {
                users: reversedSearchArr
            }
        ]
    }).lean().exec());

    if (roomError) return res.status(500).send(roomError);

    if (!room) {
        let [createdRoom, createdRoomError] = await handle(Room.create({users:
searchArr}));

        if (createdRoomError) return res.status(500).send(createdRoomError);

        room = createdRoom;

        req.app.io.to(req.params.id).emit('new-room', await roomObj(room,
req.params.id));
    }

    return res.status(200).send(await roomObj(room, user));
}

module.exports = {
    getRoomsList,
    createRoom,
    roomDetail,
    deleteRoom,
    updateUsersList,
    getRoomWithUser
}
const
    UserRights = require('../Rights/MainRights'),

    mongoose = require('../Config/database'),
    sendMail = require('../Config/nodemailer'),
    config = require('../Config/config.json'),

    User = mongoose.model('User'),

    moment = require('moment'),
    fsx = require('fs-extra'),
    path = require('path'),
    md5 = require('md5'),

    ValidationField = require('../Models/validationField'),

    FieldsValidator = require('../Utils/fieldsValidator'),
    {userObj} = require('../Utils/modelObjects'),
```

```
    {handle, generateCode, searchParams} = require('../Utils/utils');

async function getUsersList(req, res) {

    let {limit, offset, sortField, sortType} = searchParams(req);

    let params = [
        new ValidationField('email', req.query.email, 'searchString', true, 'email'),
        new ValidationField('name', req.query.name, 'searchString', true, 'name'),
        new   ValidationField('lastName',   req.query.lastName,   'searchString',   true,
'lastName'),
        new   ValidationField('nickname',   req.query.nickname,   'searchString',   true,
'nickname')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) return res.status(400).send(errors);

    obj._id = {$ne: req.user.id};

    let userRights = new UserRights(req.user.id, null, null);
    await userRights.initializeUser();

    // let ids = obj['_id'];
    //
    // obj['_id'] = await userRights.whatCanRead(ids);

    let              [users,              usersError]              =              await
handle(User.find(obj).limit(limit).skip(offset).sort({[sortField]:
sortType}).lean().exec());

    if (usersError) {
        return res.status(403).send({
            error: usersError
        })
    }

    let            [countUsers,            countUsersError]            =            await
handle(User.countDocuments(obj).lean().exec());

    if (countUsersError) {
        return res.status(403).send({
            error: countUsersError
        })
    }

    return res.status(200).send({
        data: users.map(el => userObj(el)),
        count: countUsers
    });
}

async function createUser(req, res) {
    /* fields
        * email
        * name
        * lastName
        * nickname
        * publicKey
    */
```

```
    let {email, name, lastName, nickname, publicKey} = req.body;

    let params = [
        new ValidationField('email', email, 'email', false, 'email'),
        new ValidationField('name', name, 'string', false, 'name'),
        new ValidationField('lastName', lastName, 'string', false, 'lastName'),
        new ValidationField('nickname', nickname, 'nickname', false, 'nickname'),
        new ValidationField('publicKey', publicKey, 'string', false, 'publicKey')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [findNickUser, findNickUserError] = await handle(User.findOne({nickname:
nickname}).lean().exec());

    if (findNickUserError) {
        return res.status(403).send({
            error: findNickUserError
        })
    }

    if (findNickUser) {
        return res.status(409).send({
            field: 'nickname'
        });
    }

    let [findUser, findUserError] = await handle(User.findOne({email:
email}).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (findUser) {
        return res.status(409).send({
            field: 'email'
        });
    }

    obj.code = generateCode();
    obj.codeExpiresIn = moment(moment().valueOf() + 600000).format('YYYY-MM-DD
HH:mm:ss');
    obj.onlineStatus = 'offline';
    obj.hidden = false;

    let [createdUser, createdUserError] = await handle(User.create(obj));

    if (createdUserError) {
        return res.status(403).send({
            error: createdUserError
        })
    }
```

```
    let [sendEmail, sendEmailError] = await handle(sendMail(config.email.sender,
createdUser.email, 'Secure chat authorization code', 'Your chat auth code is -> ' +
createdUser.code));

    if (sendEmailError) {
        return res.status(403).send({
            error: sendEmailError
        })
    }

    return res.status(200).send(userObj(createdUser));
}

async function getUserInfo(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false)
    ]

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [user, userError] = await handle(User.findById(req.params.id).lean().exec());

    if (userError) {
        return res.status(403).send({
            error: userError
        })
    }

    if (!user) {
        return res.status(404).end();
    }

    let userRights = new UserRights(req.user.id, null, user);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('read');

    if (!rights.can) return res.status(rights.status).send(rights.message);

    return res.status(200).send(userObj(user));
}

async function updateUser(req, res) {
    /* fields
        id
        name
        lastName
        email
        profileDescription
        nickname
    */

    let {name, lastName, profileDescription} = req.body;

    let params = [
        new ValidationField('id', req.params.id, 'string', false),
```

```
        new ValidationField('name', name, 'string', true, 'name'),
        new ValidationField('lastName', lastName, 'string', true, 'lastName'),
        new ValidationField('profileDescription', profileDescription, 'string', true,
'profileDescription')
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [findUser, findUserError] = await
handle(User.findById(req.params.id).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (!findUser) {
        return res.status(404).end();
    }

    let userRights = new UserRights(req.user.id, null, findUser);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('change');

    if (rights.can) return res.status(rights.status).send(rights.message);

    let [updatedUser, updatedUserError] = await handle(User.updateOne({_id:
req.params.id}, obj).lean().exec());

    if (updatedUserError) {
        return res.status(403).send({
            error: updatedUserError
        })
    }

    return res.status(200).end();
}

async function deleteUser(req, res) {
    let {errors, obj} = FieldsValidator([new ValidationField('id', req.params.id,
'string', false)]);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [findUser, findUserError] = await
handle(User.findById(req.params.id).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
```

```
        })
    }

    if (!findUser) {
        return res.status(404).end();
    }

    let userRights = new UserRights(req.user.id, null, findUser);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('delete');

    if (rights.can) return res.status(rights.status).send(rights.message);

    // TODO delete user traces

    let [deleteUser, deleteUserError] = await handle(User.deleteOne({_id:
req.params.id}));

    if (deleteUserError) {
        return res.status(403).send({
            error: deleteUserError
        })
    }

    return res.status(200).end();
}

async function blockUnblockUser(req, res) {
    let {errors, obj} = FieldsValidator([
        new ValidationField('id', req.params.id, 'string', false),
        new ValidationField('status', req.query.status, 'string', false)
    ]);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [findUser, findUserError] = await handle(User.findOne({_id:
req.params.id}).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (!findUser) return res.status(404).end();

    let userRights = new UserRights(req.user.id, null, findUser);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('block');

    if (rights.can) return res.status(rights.status).send(rights.message);

    let [updateUser, updateUserError] = await
handle(User.findByIdAndUpdate(req.params.id, {status: req.query.status}));

    if (updateUserError) {
```

```
        return res.status(403).send({
            error: updateUserError
        });
    }

    return res.status(200).end();
}

async function loadImage(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'string', false),
        new ValidationField('file', req.files?.avatar, 'file', false)
    ];

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({errors: errors})
    }

    let [user, userError] = await handle(User.findById(req.params.id).lean().exec());

    if (userError) {
        return res.status(403).send({
            error: userError
        })
    }

    if (!user) {
        return res.status(404).end();
    }

    let userRights = new UserRights(req.user.id, null, user);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('change');

    if (rights.can) return res.status(rights.status).send(rights.message);

    let file = req.files.avatar,
        extension = path.extname(file.name),
        fileName = md5('verylongfilename' + Date.now()),
        avatar = 'uploads/avatar/' + fileName + extension

    await file.mv('./' + avatar);

    let [update, updateError] = await handle(User.updateOne({_id: req.params.id},
{avatar}));

    if (updateError) return res.status(500).send(updateError);

    let [updatedUser, updatedUserError] = await
handle(User.findById(req.params.id).lean().exec());

    if (updatedUserError) {
        return res.status(500).send({
            error: updatedUserError
        })
    }

    res.status(200).send(userObj(updatedUser));
}
```

```
async function deleteImage(req, res) {
    let {errors, obj} = FieldsValidator([
        new ValidationField('id', req.params.id, 'string', false)
    ]);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [findUser, findUserError] = await handle(User.findById(req.params.id).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (!findUser) return res.status(404).end();

    let userRights = new UserRights(req.user.id, null, findUser);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('change');

    if (rights.can) return res.status(rights.status).send(rights.message);

    if (!findUser.avatar) return res.status(200).end();

    let [deleteFile, deleteFileError] = await handle(fsx.remove('./' + findUser.avatar));

    if (deleteFileError) {
        return res.status(403).send({
            error: deleteFileError
        })
    }

    let [updateUser, updateUserError] = await handle(User.updateOne({_id: req.params.id}, {
        avatar: null
    }))

    if (updateUserError) {
        return res.status(403).send({
            error: updateUserError
        })
    }

    return res.status(200).end();
}

async function changeEmail(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'number', false),
        new ValidationField('email', req.body.email, 'email', false)
    ]

    let {errors, obj} = FieldsValidator(params);
```

```
    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }

    let [findUser, findUserError] = await handle(User.findById(req.params.id).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (!findUser) return res.status(404).end();

    let userRights = new UserRights(req.user.id, null, findUser);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('change');

    if (rights.can) return res.status(rights.status).send(rights.message);

    let [findUserWithSameEmail, findUserWithSameEmailError] = await handle(User.findOne({email: req.body.email}).lean().exec());

    if (findUserWithSameEmailError) {
        return res.status(403).send({
            error: findUserWithSameEmailError
        })
    }

    if (findUserWithSameEmail) return res.status(409).end();

    let [updateEmail, updateEmailError] = await handle(User.updateOne({_id: req.params.id}, {
        email: req.body.email
    }));

    if (updateEmailError) {
        return res.status(403).send({
            error: updateEmailError
        })
    }

    return res.status(200).end();
}

async function changeNickname(req, res) {
    let params = [
        new ValidationField('id', req.params.id, 'number', false),
        new ValidationField('nickname', req.body.nickname, 'string', false)
    ]

    let {errors, obj} = FieldsValidator(params);

    if (errors.length > 0) {
        return res.status(400).send({
            errors: errors
        })
    }
```

```
    let             [findUser,             findUserError]             =             await
handle(User.findById(req.params.id).lean().exec());

    if (findUserError) {
        return res.status(403).send({
            error: findUserError
        })
    }

    if (!findUser) return res.status(404).end();

    let userRights = new UserRights(req.user.id, null, findUser);
    await userRights.initializeUser();

    let rights = await userRights.checkRights('change');

    if (rights.can) return res.status(rights.status).send(rights.message);

    let     [findUserWithSameNickname,     findUserWithSameNicknameError]     =     await
handle(User.findOne({nickname: req.body.nickname}).lean().exec());

    if (findUserWithSameNicknameError) {
        return res.status(403).send({
            error: findUserWithSameNicknameError
        })
    }

    if (findUserWithSameNickname) return res.status(409).end();

    let  [updateNickname,  updateNicknameError]  =  await  handle(User.updateOne({_id:
req.params.id}, {
        nickname: req.body.nickname
    }));

    if (updateNicknameError) {
        return res.status(403).send({
            error: updateNicknameError
        })
    }

    return res.status(200).end();
}

module.exports = {
    getUsersList,
    createUser,
    getUserInfo,
    updateUser,
    deleteUser,
    blockUnblockUser,
    loadImage,
    deleteImage,
    changeNickname,
    changeEmail
}
{
  "usage": "TO USE REPLACE DATA WITH YOUR VALUES AND DELETE THIS LINE",
  "port": 1111,
  "jwt-token": "TestTestTestTest",
  "secret-key": "AAAAAAAAAAAAA",
  "secret-key-resolved": "AAAAAAAAAAAAAAAAA",
```

```
  "db": {
    "name": "1111",
    "host": "localhost",
    "user": "chat",
    "password": "1111",
    "port": 27017
  },
  "email": {
    "sender": "sender",
    "service": "gmail",
    "auth": {
      "user": "1111",
      "password": "1111"
    }
  }
}
const
    mongoose = require('mongoose'),
    config = require('./config.json');

mongoose.model('Room',      require('../Models/room'));
mongoose.model('User',      require('../Models/user'));
mongoose.model('Message',   require('../Models/message'));
mongoose.model('Logs',      require('../Models/logs'));

mongoose.Promise = global.Promise;

mongoose.connect(`mongodb://${config.db.user}:${config.db.password}@${config.db.host}:$
{config.db.port}/${config.db.name}`, {
    useFindAndModify: false,
    useNewUrlParser: true,
    useUnifiedTopology: true,
    connectTimeoutMS: 30000,
    promiseLibrary: global.Promise,
    // ssl: true // for remote server only
})
    .then(() => {
        console.log('connected successfully');
    }).catch(error => {
        throw Error("connect to db is impossible because: " + error);
    })

module.exports = mongoose;
let expressJwt = require('express-jwt'),
    config = require('./config.json');

module.exports = expressJwt({
    secret: config["jwt-token"],
    algorithms: ['HS256'],
    getToken: function (req) {
        if (req.header("token")) {
            return req.header("token");
        }
        return null;
    }
});
const
    nodemailer = require('nodemailer'),
    config = require('../Config/config.json');

const transporter = nodemailer.createTransport({
    service: config.email.service,
```

```
    auth: {
        user: config.email.auth.user,
        pass: config.email.auth.password
    },
    secure: true
});

async function sendMail(from, to, subject, text, html) {
    return new Promise((resolve, reject) => {
        transporter.sendMail({
            from: from,
            to: to,
            subject: subject,
            text: text,
            html: html
        }, (error, info) => {
            if (error) {
                console.log("Email send error \n" + error);
                reject(error);
            } else {
                console.log('Email sent: ' + info.response);
                resolve(true);
            }
        })
    })
}

module.exports = sendMail;
```

Додаток Г. Лістинг коду клієнтської частини

```
const {ipcRenderer, contextBridge} = require('electron');

let validIPCChannels = [
    'error',
    'keySave'
]

contextBridge.exposeInMainWorld('Electron', {
    errors: {
        showError: (channel, message, title) => {
            if (validIPCChannels.includes(channel)) {
                ipcRenderer.send(channel, message, title);
            }
        }
    },
    files: {
        keySave: (channel, key, type) => {
            if (validIPCChannels.includes(channel)) {
                ipcRenderer.send(channel, key, type);
            }
        }
    }
})
const
    {app, BrowserWindow, ipcMain} = require('electron'),
    path = require('path');

app.setName("CryptoGram");

// require('electron-reload')(__dirname, {
//     electron: path.join(__dirname, 'node_modules', '.bin', 'electron')
// });

let mainWindow;

async function createWindow () {
    const win = new BrowserWindow({
        width: 1200,
        height: 800,
        resizable: false,
        autoHideMenuBar: true,
        webPreferences: {
            nodeIntegration: false,
            preload: path.resolve(app.getAppPath(), 'preload.js')
        }
    })

    await win.loadFile("index.html")
}

app.whenReady().then(createWindow);

// default handlers for macOS
app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
        app.quit();
    }
})
```

```
app.on('activate', async () => {
    if (BrowserWindow.getAllWindows().length === 0) {
        mainWindow = await createWindow();
    }
})

require('./electron/ipcMain')(ipcMain, mainWindow);
require('./electron/fileSave')(ipcMain, mainWindow);
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="Content-Security-Policy" content="script-src 'self'" />
    <title>CryptoGram</title>
</head>
<body>
  <div id="cryptogram"></div>
</body>
<script src="./build/js/app.js"></script>
</html>
let socket = require('socket.io-client')(process.env.BACKEND_URL, {
    query: {
        'token': localStorage.getItem('token')
    }
});

export default socket;
import React from "react";
import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(<App />, document.getElementById('cryptogram'))
* {
    margin: 0;
    padding: 0;
    outline: 0;
    font-family: Arial, sans-serif;
    color: #FFFFFF;
}

body {
    background-color: #262626;
    width: 100%;
    height: 100%;
}

.clear {
    clear: both;
}

.normal {
    padding: 10px;
}

::-webkit-scrollbar {
    width: 5px;
}

::-webkit-scrollbar-track {
    background: #FFFFFF;
```

```
}

::-webkit-scrollbar-thumb {
    background: #888;
}

::-webkit-scrollbar-thumb:hover {
    background: #555;
}

input {
    color: white;
    background-color: #555555;
}

::placeholder {
    color: #9c9c9c;
}

.block-with-inputs {
    font-size: 16px;
    position: absolute;
    top: 40%;
    left: 50%;
    transform: translate(-50%, -50%);
    display: grid;
    column-gap: 20px;
    row-gap: 20px;
    grid-template-columns: repeat(2, 1fr);
}

.block-with-inputs input {
    outline: none;
    padding: 10px;
    border-radius: 10px;
    border: 1px solid #262626;
}

.block-with-inputs p {
    grid-column: auto / span 2;
    text-align: center;
    line-height: 20px;
}

button {
    cursor: pointer;
}

.link {
    color: #0052ff;
}

.btn-reg-login {
    grid-column: auto / span 2;
    margin: auto;
    padding: 10px 0;
    width: 50%;
    background-color: #0052ff;
    color: #FFFFFF;
    border: none;
    border-radius: 10px;
}
```

```
.over {
    position: absolute;
    z-index: 1000;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
}

.popup {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: #555555;
    border: 1px solid #888888;
    width: 350px;
    box-sizing: border-box;
    padding: 10px;
    opacity: 1;
    z-index: 3;
    border-radius: 10px;
}

.popup button {
    background-color: #0052ff;
    color: #FFFFFF;
    border: none;
    border-radius: 10px;
    margin: 10px auto;
    float: right;
    padding: 10px 35px;
}

.over-cover {
    position: absolute;
    z-index: 2;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: black;
    opacity: 0.7;
}

.warning {
    border: 1px solid red !important;
}
import React from "react";

import {HashRouter as Router, Switch, Route} from "react-router-dom";

import './app.scss';
import Settings     from "./components/Settings";
import LoadPage     from "./components/LoadPage";
import SendCode     from "./components/SendCode";
import Register     from "./components/Register";
import PageNotFound from "./components/PageNotFound";
import Login        from "./components/Login";
import MainView     from "./components/MainView";
import DialogPage   from "./components/DialogPage";
```

```
import ErrorPage    from "./components/ErrorPage";

import MainStore    from "./store/MainStore";

export default function App () {
    return(
        <Router>
            <Switch>
                <Route path="/settings">
                    <Settings />
                </Route>
                <Route path="/send-code">
                    <SendCode store={MainStore.sendCodeStore}/>
                </Route>
                <Route path="/login">
                    <Login store={MainStore}/>
                </Route>
                <Route path="/register">
                    <Register store={MainStore.registerStore}/>
                </Route>
                <Route path="/main-view">
                    <MainView store={MainStore}/>
                </Route>
                <Route path="/not-found">
                    <PageNotFound />
                </Route>
                <Route path="/error">
                    <ErrorPage />
                </Route>
                <Route path="/room/:id">
                    <DialogPage store={MainStore.dialogPageStore}/>
                </Route>
                <Route path="/">
                    <LoadPage />
                </Route>
            </Switch>
        </Router>
    )
}
export function apiError (error) {
    return {
        status: error.response.status,
        data: null,
        errors: error.response.data
    }
}

export function apiSuccess (response) {
    return {
        status: response.status,
        data: response.data,
        errors: null
    }
}
import Axios from "axios";

import {apiError, apiSuccess} from "./apiReturn";

export default async function axios(data, url, method, headers) {
    try {
        let request = await Axios({
            url: url,
```

```
            data: data,
            method: method,
            headers: headers
        })

        return [apiSuccess(request), null];
    } catch (e) {
        return [null, apiError(e)];
    }
}
import JsEncrypt from "jsencrypt";
import keypair from "keypair";
import {toJS} from "mobx";

/*
    ↓↓↓↓ key pair generation example ↓↓↓↓

    (async () => {
        let pair = keypair({bits: 512});

        console.log(pair);
    })()
*/

function messageShortener(message) {
    if (message.length >= 70) {
        return message.substr(0, 70) + "...";
    } else return message;
}

export function cryptMessage(message, publicKey) {
    let crypt = new JsEncrypt();

    crypt.setPublicKey(publicKey);

    let Parts = message.match(/.{1,50}/g),
        encryptedArr = [];

    for (let el of Parts) {
        encryptedArr.push(crypt.encrypt(el));
    }

    return encryptedArr;
}

export function decryptMessagesFromRoomObj(roomArr, privateKey) {
    let crypt = new JsEncrypt();

    crypt.setPrivateKey(privateKey);

    for (let i = 0; i < roomArr.length; i++) {
        if (!roomArr[i].lastMessage) continue;
        let decryptedArr = [];

        for (let elMes of roomArr[i].lastMessage.messageObj.message) {
            decryptedArr.push(crypt.decrypt(elMes));
        }

        roomArr[i].lastMessage.messageObj.message                                    =
messageShortener(decryptedArr.join(''));
    }
```

```
    return roomArr;
}

export function decryptMessages(messageArr, privateKey) {
    let crypt = new JsEncrypt();

    crypt.setPrivateKey(privateKey);

    for (let i = 0; i < messageArr.length; i++) {
        let decryptedArr = [];

        for (let elMes of messageArr[i].messageObj.message) {
            decryptedArr.push(crypt.decrypt(elMes));
        }

        messageArr[i].messageObj.message = decryptedArr.join('');
    }

    return messageArr;
}
.dialog-page {
    height: 100%;
    display: block;
    width: 100%;
    position: absolute;
    left: 0;
    right: 0;
    top: 0;
    bottom: 0;
}

.dialog-header {
    box-sizing: border-box;
    display: block;
    height: 10vh;
    width: 100vw;
}

.dialog-bottom {
    display: flex;
    height: 10vh;
    width: 100vw;
    box-sizing: border-box;
    justify-content: center;
    padding-top: 15px;
}

.message-box {
    height: 80vh;
    box-sizing: border-box;
    width: 100vw;
    display: block;
    overflow-y: scroll;
}

.message-box p {
    word-wrap: break-word;
}

.dialog-header-left {
    float: left;
    width: 5%;
```

```
    box-sizing: border-box;
    height: 100%;
    position: relative;
}

.dialog-header-left img {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 30px;
    height: 30px;
    cursor: pointer;
}

.dialog-header-right {
    float: left;
    box-sizing: border-box;
    width: 95%;
    height: 100%;
    padding: 5px;
    position: relative;
    display: flex;
}

.dialog-header-avatar {
    width: 65px;
    height: 65px;
    border-radius: 100%;
    margin: 0 10px 0 0;
}

.message-input {
    outline: 0;
    box-sizing: border-box;
    height: 40px;
    width: 80%;
    display: block;
    border: none;
    border-radius: 15px;
    padding: 10px;
}

.send-message-btn {
    border-radius: 100%;
    display: block;
    width: 40px;
    height: 40px;
    background-color: #0052ff;
    cursor: pointer;
    position: relative;
    margin-left: 20px;
}

.send-message-btn img {
    position: absolute;
    width: 25px;
    height: 25px;
    top: 50%;
    left: 55%;
    transform: translate(-50%, -50%);
}
```

```
.dialog-header-user-info-block {
    padding: 10px;
    box-sizing: border-box;
    display: flex;
    flex-direction: column;
    justify-content: space-around;
}

.dhuibf {
    font-size: 18px;
}

.dhuibs {
    font-size: 14px;
}
.upload-btn {
    grid-column: auto / span 2;
    width: 100%;
    box-sizing: border-box;
    background-color: #888888;
}

.email-login, .code-login {
    grid-column: auto / span 2;
    width: 100%;
    box-sizing: border-box;
    margin: auto;
}

.login-btn {
    width: 100%;
}
.main-view {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
}

.main-view-profile {
    float: left;
    height: 100%;
    width: 30%;
    box-sizing: border-box;
    position: relative;
}

.main-view-rooms {
    float: left;
    height: 100%;
    width: 70%;
    box-sizing: border-box;
}

.avatar {
    width: 125px;
    height: 125px;
    display: block;
    margin: 50px auto;
    background-repeat: no-repeat;
```

```
    background-position: center center;
    border-radius: 100%;
    background-size: cover;
    cursor: pointer;
}

.main-view-user-name {
    text-align: center;
    font-size: 20px;
    margin-bottom: 40px;
}

.exit-btn {
    position: absolute;
    top: 2%;
    right: 2%;
    height: 30px;
    width: 30px;
    cursor: pointer;
    background-size: cover;
}

.main-view-profile-options {
    cursor: pointer;
    text-align: center;
    font-size: 18px;
    margin-bottom: 10px;
}

.main-view-profile-options:hover {
    color: #0052ff;
}

.user-search-input {
    width: 95%;
    margin: 10px auto;
    box-sizing: border-box;
    display: block;
    outline: none;
    border-radius: 15px;
    padding: 13px;
    border: none;
}

.user-tile {
    display: flex;
    width: 95%;
    box-sizing: border-box;
    padding: 10px;
    height: 70px;
    margin: auto;
    flex-direction: row;
    cursor: pointer;
}

.user-tile p {
    display: block;
    margin: auto 0 auto 10px;
    font-size: 18px;
}

.user-tile-avatar {
```

```
    height: 50px;
    width: 50px;
    background-size: cover;
    background-position: center;
    border-radius: 100%;
    float: left;
}

.roomClass {
    height: 90px;
    display: flex;
    width: 95%;
    box-sizing: border-box;
    margin: auto;
    padding: 10px;
    cursor: pointer;
}

.roomClass-avatar {
    height: 70px;
    width: 70px;
    background-size: cover;
    background-position: center;
    border-radius: 100%;
    float: left;
}

.roomClass-info {
    position: relative;
    flex: 1;
    margin: auto 0 auto 10px;
}

.room-user-name {
    font-size: 18px;
    margin-bottom: 10px;
}

.room-last-message-time {
    position: absolute;
    right: 0;
    top: 0;
}
.my-message, .inner-message {
    max-width: 400px;
    width: fit-content;
    min-width: 100px;
    box-sizing: border-box;
    padding: 10px 37px 15px 10px;
    border-radius: 15px;
    margin-bottom: 10px;
    border: 1px solid rgba(81, 81, 81, 0.2);
    position: relative;
}

.my-message .message, .inner-message .message {
    /* padding-bottom: 20px; */
    display: block;
    max-width: 240px;
    word-break: break-all;
    word-break: break-word;
}
```

```
.my-message {
    margin-right: 1%;
    margin-left: auto;
    background-color: #0052ff;
}

.inner-message {
    margin-left: 1%;
    margin-right: auto;
    background-color: #555555;
}

.message-time {
    position: absolute;
    bottom: 5px;
    right: 10px;
    color: #fff;
    font-size: 10px;
}

.message-obj {
    position: relative;
    display: block;
    margin: auto;
}
.register-image {
    background-position: center;
    background-size: cover;
    background-repeat: no-repeat;
    width: 200px;
    height: 200px;
    margin: 0 auto 20px auto;
    grid-column: auto / span 2;
}

.upload-btn {
    margin: auto;
    padding: 10px;
    border-radius: 10px;
    border: none;
    background-color: #0052ff;
    color: #FFFFFF;
}
.send-code-input {
    grid-column: auto / span 2;
    width: 50%;
    margin: auto;
    box-sizing: border-box;
}
import {makeAutoObservable} from "mobx";
import {decryptMessages} from "../utils/cryptDecrypt";

class DialogPageStore {
    roomId;
    messages;
    roomObj;
    privateKey;
    publicKey;
    inputValue;

    constructor() {
```

```
        this.roomId = null;
        this.messages = [];
        this.roomObj = null;
        this.inputValue = '';

        makeAutoObservable(this);
    }

    setRoomObj(roomObj) {
        this.roomObj = roomObj;
    }

    setRoomId (id) {
        this.roomId = id;
    }

    pushMessage (message) {
        this.messages.push(...decryptMessages([message], this.privateKey));
    }

    setMessages (messages) {
        // this.messages = messages.map(el => el.messageObj.message.join(''));
        this.messages = decryptMessages(messages, this.privateKey);
    }

    clearDialogPage() {
        this.roomId = null;
        this.messages = [];
        this.roomObj = null;
    }

    setInputValue (value) {
        this.inputValue = value;
    }
}

export default DialogPageStore;
import {makeAutoObservable} from "mobx";

class LoginStore {
    loginInputValue;
    loginInputStyle;

    btnText;
    privateKey;

    popupText;
    popupShow;
    behaviour;

    constructor() {
        this.loginInputValue = '';
        this.loginInputStyle = 'normal';

        this.btnText = 'Upload private key';
        this.privateKey = null;

        this.popupText = '';
        this.popupShow = 'none';
        this.behaviour = () => {this.popupShow = 'none'}

        makeAutoObservable(this);
```

```
    }

    changeInputValue(value) {
        this.loginInputValue = value;
    }

    changeInputStyle(value) {
        this.loginInputStyle = value;
    }
}

export default LoginStore;
import {makeAutoObservable} from "mobx";

import SendCodeStore from "./SendCodeStore";
import RegisterStore from "./RegisterStore";
import LoginStore from "./LoginStore";
import DialogPageStore from "./DialogPageStore";

import {getUserDataApi} from "../api/user";
import {decryptMessagesFromRoomObj} from "../utils/cryptDecrypt";

import {findUsersByNickname} from '../api/mainView';

class Store {
    sendCodeStore;
    registerStore;
    loginStore;
    mainViewStore;
    dialogPageStore;

    privateKey;
    publicKey;

    userObj;

    searchPeopleInput;
    foundedUsers;

    rooms;

    constructor() {
        this.sendCodeStore      = new SendCodeStore();
        this.registerStore      = new RegisterStore();
        this.loginStore         = new LoginStore();
        this.dialogPageStore    = new DialogPageStore();
        this.privateKey         = null;
        this.publicKey          = null;
        this.userObj            = null;
        this.searchPeopleInput  = '';
        this.foundedUsers       = [];
        this.rooms = [];

        makeAutoObservable(this);
    }

    setFoundedUsers(array) {
        this.foundedUsers = array;
    }

    setRoomAndUserAndKeys(user, privateKey, publicKey, rooms) {
        this.userObj = user;
```

```
        this.privateKey = privateKey;
        this.dialogPageStore.privateKey = privateKey;
        this.publicKey = publicKey;
        this.dialogPageStore.publicKey = publicKey;
        this.setRooms(rooms);
    }

    setUser (user) {
        this.userObj = user;
    }

    updateAvatar(newAvatar) {
        this.userObj.avatar = newAvatar;
    }

    setPrivateKey(value) {
        this.privateKey = value;
        this.dialogPageStore.privateKey = value;
    }

    setPublicKey(value) {
        this.publicKey = value;
        this.dialogPageStore.publicKey = value;
    }

    changeSearchPeopleValue (value) {
        this.searchPeopleInput = value;
    }

    getRooms () {
        return this.rooms;
    }

    setRooms (rooms) {
        this.rooms = decryptMessagesFromRoomObj(rooms, this.privateKey);
    }

    setRoomsWithoutDecryption(rooms) {
        this.rooms = rooms;
    }

    decryptOneRoom(roomObj) {
        return decryptMessagesFromRoomObj([roomObj], this.privateKey)[0];
    }

    updateRooms (room) {
        this.rooms = [room, ...this.rooms];
    }
}

let MainStore = new Store();

export default MainStore;
import {makeAutoObservable} from "mobx";

import {registerUserApi} from "../api/register";

class RegisterStore {
    emailInputValue;
    nameInputValue;
    lastNameInputValue;
    nicknameInputValue;
```

```
emailInputStyle;
nameInputStyle;
lastNameInputStyle;
nicknameInputStyle;

privateKey;
publicKey;

showPopup;

constructor() {
    this.emailInputValue = '';
    this.nameInputValue = '';
    this.lastNameInputValue = '';
    this.nicknameInputValue = '';

    this.emailInputStyle = 'normal';
    this.nameInputStyle = 'normal';
    this.lastNameInputStyle = 'normal';
    this.nicknameInputStyle = 'normal';

    this.publicKey = null;
    this.privateKey = null;

    this.showPopup = 'none';

    makeAutoObservable(this);
}

changeInputValue(value, field) {
    switch (field) {
        case 'emailInputValue':
            this.changeInputStyle('normal', 'emailInputStyle');
            break;
        case 'nameInputValue':
            this.changeInputStyle('normal', 'nameInputStyle');
            break;
        case 'lastNameInputValue':
            this.changeInputStyle('normal', 'lastNameInputStyle');
            break;
        case 'nicknameInputValue':
            this.changeInputStyle('normal', 'nicknameInputStyle');
            break;
    }
    this[field] = value;
}

changeInputStyle(value, field) {
    this[field] = value;
}

setPublicKey (value) {
    this.publicKey = value;
}

setPrivateKey(value) {
    this.privateKey = value;
}

clearAllInputs() {
    this.emailInputValue = '';
```

```
        this.nameInputValue = '';
        this.lastNameInputValue = '';
        this.nicknameInputValue = '';

        this.emailInputStyle = 'normal';
        this.nameInputStyle = 'normal';
        this.lastNameInputStyle = 'normal';
        this.nicknameInputStyle = 'normal';

        this.publicKey = null;
        this.privateKey = null;
    }

    async registerUser() {
        return    await    registerUserApi(this.emailInputValue,    this.nameInputValue,
this.lastNameInputValue, this.nicknameInputValue, this.publicKey);
    }
}

export default RegisterStore;
import {makeAutoObservable} from "mobx";
import {sendCodeApi} from "../api/login";

class SendCodeStore {
    classList;
    placeholder;
    sendCodeInputValue;

    popupText;
    popupShow;
    behaviour;

    constructor() {
        this.classList = 'normal';
        this.placeholder = 'Enter your email to receive code';
        this.sendCodeInputValue = '';

        this.popupText = '';
        this.popupShow = 'none';
        this.behaviour = () => {this.popupShow = 'none'}

        makeAutoObservable(this);
    }

    setErrorStyle() {
        this.classList = 'warning';
        this.placeholder = 'Your email is incorrect, please enter another';
    }

    setNormalStyle() {
        this.classList = 'normal';
        this.placeholder = 'Enter your email to receive code';
    }

    async sendCode(email) {
        return await sendCodeApi(email);
    };

    setSendCodeInput(value) {
        this.sendCodeInputValue = value;
    };
}
```

```
export default SendCodeStore;
import React, {useEffect} from "react";
import {observer} from "mobx-react";
import {cryptMessage} from "../utils/cryptDecrypt";
import {createMessageApi} from "../api/message";
import {toJS} from "mobx";

async function sendMessage(store) {
    if (store.inputValue.length > 0) {
        let messageCreateObj = {
            room: store.roomId,
            messageObj: {}
        };

        for (let user of store.roomObj.users) {
            messageCreateObj.messageObj[user.id]                =                {message:
cryptMessage(store.inputValue, user.publicKey)};
        }

        let         [createdMessage,         createdMessageError]         =         await
createMessageApi(messageCreateObj);

        if (createdMessageError) history.push('/error');

        // store.pushMessage(createdMessage.data);
        store.setInputValue("");
    }
}

const DialogBottom = observer(({store}) => {
    return <div className='dialog-bottom'>
        <input
            className="message-input"
            type="text"
            onChange={(e) => {
                store.setInputValue(e.target.value);
            }}
            value={store.inputValue}
            placeholder="Input your message"
            onKeyDown={
                async event => {
                    if (event.key === 'Enter') {
                        await sendMessage(store);
                    }
                }
            }
        />

        <div
            className="send-message-btn"
            onClick={
                async () => {
                    await sendMessage(store);
                }
            }
        >
            <img src="./build/assets/arrowforward.png" alt=""/>
        </div>
    </div>
});
```

```
export default DialogBottom;
import React from 'react';

import {useHistory, useParams} from "react-router-dom";

import {observer} from "mobx-react";
import {toJS} from "mobx";
import socket from "../socket";


const DialogHeader = observer(({store}) => {
    let history = useHistory();
    return <div className="dialog-header">
        <div className="dialog-header-left">
            <img src="./build/assets/arrowback.png" alt="HELP" onClick={() => {
                socket.emit('leaveRoom',
`${store.roomObj?.id}${localStorage.getItem('id')}`);
                store.clearDialogPage();
                history.goBack();
            }}/>
        </div>
        <div className="dialog-header-right">
            <img
                src={store.roomObj?.otherUser?.avatar ? process.env.BACKEND_URL + "/" +
store.roomObj.otherUser.avatar : "./build/assets/defaultAvatar.png"}
                alt="" className="dialog-header-avatar"/>
            <div className="dialog-header-user-info-block">
                <p                    className="dhuibf">{store.roomObj?.otherUser?.name}
{store.roomObj?.otherUser?.lastName}</p>
                <p className="dhuibs">{store.roomObj?.otherUser?.onlineStatus}</p>
            </div>
        </div>
        <div className="clear"></div>
    </div>
})

export default DialogHeader;
import React, {useEffect} from "react";
import {observer} from "mobx-react";
import MessageComp from "./MessageComp";

const DialogMessageBox = observer(({store}) => {
    let messagesEnd = React.useRef < HTMLInputElement > null;

    useEffect(() => {
        messagesEnd.scrollIntoView({behavior: "auto"})
    });

    return <div className="message-box">
        {
            store.messages.map(el => {
                return <MessageComp messageObj={el} key={el.id}/>
            })
        }

        <div style={{float: "left", clear: "both"}}
             ref={(el) => {
                 messagesEnd = el;
             }}>
        </div>
    </div>
});
```

```
export default DialogMessageBox;
import React, {useEffect} from "react";
import {useHistory, useParams} from "react-router-dom";
import {observer} from "mobx-react";
import {getRoomDetailApi} from '../api/room';
import {getMessagesApi} from '../api/message';

import '../styles/DialogPage.css';
import '../styles/Message.css';

import DialogHeader from './DialogHeader';
import DialogBottom from "./DialogBottom";
import DialogMessageBox from "./DialogMessageBox";
import {toJS} from "mobx";

import socket from '../socket';

import {decryptMessages} from "../utils/cryptDecrypt";


const DialogPage = observer(({store}) => {
    let history = useHistory();
    let {id} = useParams();

    useEffect(async () => {
        socket.on('new-room-message', (data) => {
            data = JSON.parse(data);
            store.pushMessage(data);
        });

        let [apiDetail, apiDetailError] = await getRoomDetailApi(id);

        if (apiDetailError) {
            switch (apiDetailError.status) {
                case 404:
                    history.push('/not-found');
                    break;
                default:
                    history.push('/error');
                    break;
            }
        }

        store.setRoomId(id);

        store.setRoomObj(apiDetail.data);

        let [messages, messagesError] = await getMessagesApi(id);

        if (messagesError) history.push('/error');

        store.setMessages(messages.data);

        function listener(event) {
            if (event.keyCode === 27) {
                socket.emit('leaveRoom',
`${store.roomObj?.id}${localStorage.getItem('id')}`);
                store.clearDialogPage();
                history.goBack();
                document.removeEventListener('keydown', listener);
            }
```

```
        }

        document.addEventListener('keydown', listener);
    }, []);

    return <div className="dialog-page">
        <DialogHeader store={store}/>

        <DialogMessageBox store={store} />

        <DialogBottom store={store} />
    </div>
});

export default DialogPage;
import React from "react";
import {Link} from "react-router-dom";

export default function ErrorPage () {
    return(
        <h1>SERVER SIDE ERROR</h1>
    )
}
import React from "react";
import MainView from "./MainView";
import SendCode from "./SendCode";

import MainStore from "../store/MainStore";

export default function LoadPage () {
    return localStorage.getItem('token') !== null ? <MainView store={MainStore}/> :
<SendCode store={MainStore.sendCodeStore} />
}
import React, {useRef} from "react";
import {observer} from "mobx-react";

import {getTokenApi} from "../api/login";
import MainStore from "../store/MainStore";

import {useHistory} from 'react-router-dom';
import '../styles/Login.css';
import Popup from "./Popup";
import {toJS} from "mobx";

const Login = observer(({store}) => {
    let      mailInputValue      =      store.sendCodeStore.sendCodeInputValue      ||
store.registerStore.emailInputValue;
    let history = useHistory();
    let backImageStyleObj = {
        backgroundImage: `url('./build/assets/defaultAvatar.png')`
    }
    let privateRef = useRef(null);
    return (
        <div className={'absolute-block'}>
            <Popup                                  behaviour={store.loginStore.behaviour}
info={store.loginStore.popupText}
                   display={store.loginStore.popupShow}/>
            <div className="block-with-inputs">
                <div className="register-image" style={backImageStyleObj}>
                </div>
                <input   type="text"   className='email-login'   value={mailInputValue}
readOnly/>
```

```
                <input
                    placeholder="Input code from email letter"
                    type="text"
                    className={store.loginStore.loginInputStyle + " code-login"}
                    value={store.loginStore.loginInputValue}
                    onChange={(e) => {
                        store.loginStore.changeInputStyle('normal');
                        store.loginStore.changeInputValue(e.target.value);
                    }}
                />
                <input
                    type="file"
                    ref={privateRef}
                    style={{display: 'none'}}
                    accept=".pem, .txt"
                    onChange={event => {
                        event.preventDefault()
                        const reader = new FileReader()
                        reader.onload = async (e) => {
                            const text = (e.target.result)
                            Electron.files.keySave('keySave', text, 'private');
                            store.loginStore.privateKey = text;
                            store.loginStore.btnText = 'File uploaded!';
                        };
                        reader.readAsText(event.target.files[0])
                    }}
                />
                <button className="upload-btn"
                        onClick={()                                                    =>
privateRef.current.click()}>{store.loginStore.btnText}</button>
                <button
                    className="btn-reg-login login-btn"
                    onClick={async () => {
                        if (!store.loginStore.privateKey) {
                            store.loginStore.popupText = 'Set your private key for normal
work of application';
                            store.loginStore.popupShow = 'block';
                            return;
                        }

                        if (store.loginStore.loginInputValue.length === 0) {
                            store.loginStore.popupText = 'Enter code from your email';
                            store.loginStore.popupShow = 'block';
                            store.loginStore.behaviour = () => {
                                store.loginStore.popupShow = 'none';
                            }
                            return;
                        }

                        if (!mailInputValue) {
                            store.loginStore.popupText = 'Please try login or register
again';
                            store.loginStore.behaviour = () => {
                                history.push('/send-code')
                            };
                            store.loginStore.popupShow = 'block';
                            return;
                        }

                        let   [token,   error]   =   await   getTokenApi(mailInputValue,
store.loginStore.loginInputValue);
                        if (error) {
```

```
                            store.loginStore.popupText = 'Wrong code entered';
                            store.loginStore.popupShow = 'block';
                            return;
                        }
                        if (token.status !== 200) {
                            store.loginStore.changeInputValue("");
                            store.loginStore.popupText = 'User not found';
                            store.loginStore.popupShow = 'block';
                        } else {
                            localStorage.setItem('id', token.data.id);
                            localStorage.setItem('token', token.data.token);
                            history.push('/main-view');
                            store.loginStore.loginInputValue = '';
                            store.registerStore.clearAllInputs();
                        }
                    }}
                >
                    Confirm code
                </button>
            </div>
        </div>
    )

})

export default Login;
import React, {useEffect} from "react";
import {Link, useHistory} from "react-router-dom";
import {observer} from "mobx-react";
import {getRoomsApi} from "../api/mainView";
import Axios from "axios";
import JsEncrypt from 'jsencrypt';
import RoomObj from "./RoomObj";

import '../styles/MainView.css';

import {getUserDataApi} from "../api/user";
import MainViewProfile from "./MainViewProfile";
import MainViewRooms from "./MainViewRooms";

import socket from "../socket";

import {decryptMessagesFromRoomObj} from "../utils/cryptDecrypt";

let crypt = new JsEncrypt();

const MainView = observer(({store}) => {
    let history = useHistory();

    useEffect(async () => {
        console.log('effect');
        let [user, userError] = await getUserDataApi(localStorage.getItem('id'));

        socket.off('new-message');
        socket.off('new-room-message');

        socket.on('new-message', (data) => {
            data = JSON.parse(data);
            let updatedRooms = [];
            let neededRoom = {};
            for (let i = 0; i < store.rooms.length; i++) {
                let obj = JSON.parse(JSON.stringify(store.rooms[i]));
```

```
                if (obj.id === data.room) {
                    obj.lastMessage = data;
                    neededRoom = store.decryptOneRoom(obj);
                } else {
                    updatedRooms.push(obj);
                }
            }

            updatedRooms = [neededRoom, ...updatedRooms];

            store.setRoomsWithoutDecryption(updatedRooms);
        })

        socket.on('new-room', (data) => {
            store.updateRooms(data);
        })

        if (userError) {
            switch (userError.status) {
                case 404:
                    history.push('/not-found');
                    break;
                default:
                    history.push('/error');
                    break;
            }
        }

        let [rooms, error] = await getRoomsApi();

        if (error) {
            switch (error.status) {
                case 404:
                    history.push('/not-found');
                    break;
                default:
                    history.push('/error');
                    break;
            }
        }

        let privateKey = await Axios.get('./private.pem');

        store.setRoomAndUserAndKeys(user.data,   privateKey.data,   user.data.publicKey,
rooms.data.data);

        crypt.setPublicKey(store.publicKey);
        crypt.setPrivateKey(store.privateKey);
    }, []);

    return (
        <div className="main-view">
            <MainViewProfile    store={store} />
            <MainViewRooms      store={store} />
            <div className="clear">
            </div>
        </div>
    )
});

export default MainView;
import React, {useRef} from 'react';
```

```
import {observer} from "mobx-react";
import {toJS} from "mobx";
import axios from "../utils/axios";
import {useHistory} from "react-router-dom";

function getFileExtension (filename) {
    return filename.substring(filename.lastIndexOf('.')+1, filename.length) || filename;
}

const MainViewProfile = observer(({store}) => {
    let history = useHistory();
    let styleObj = {
        backgroundImage: store.userObj?.avatar ? "url(" + process.env.BACKEND_URL + "/"
+ store.userObj.avatar + ")" : "url('./build/assets/defaultAvatar.png')"
    }

    let exitBtnStyleObj = {
        backgroundImage: `url('./build/assets/exitbtn.png')`
    }

    const inputFile = useRef(null);

    return <div className="main-view-profile">
        <div
            className="exit-btn"
            style={exitBtnStyleObj}
            onClick={
                () => {
                    delete localStorage.token;
                    delete localStorage.id;

                    history.push('/send-code');
                }
            }
        >
        </div>
        <input
            type='file'
            id='file'
            ref={inputFile}
            style={{display: 'none'}}
            accept=".png, .jpg"
            onChange={
                async (event) => {
                    event.stopPropagation();
                    event.preventDefault();

                    console.log(event.target.files);

                    if (!event.target.files) return;

                    let file = event.target.files[0];

                    if       (getFileExtension(file.name)       !==       'png'       &&
getFileExtension(file.name) !== 'jpeg' && getFileExtension(file.name) !== 'jpg') {
                        console.log('wrong format');
                        return;
                    }

                    let formData = new FormData();

                    formData.append("avatar", file, file.name);
```

```
                let saveImage;

                let [request, requestError] = await axios(
                    formData,
                    `${process.env.BACKEND_URL}/user/${store.userObj?.id}/image`,
                    "PATCH",
                    {
                        token: localStorage.getItem('token'),
                        "Content-Type": "multipart/form-data"
                    });

                if (requestError) {
                    console.log(requestError);
                    return;
                }

                saveImage = request.data;

                if (saveImage) {
                    store.updateAvatar(saveImage.avatar);
                }
            }
        }
    />

    <div
        className="avatar"
        style={styleObj}
        onClick={
            () => {
                inputFile.current.click();
            }
        }
    >
    </div>
    <p className="main-view-user-name">
        {store.userObj?.name + " " + store.userObj?.lastName}
    </p>

    <p
        className="main-view-profile-options"
    >
        Profile settings
    </p>

    <p
        className="main-view-profile-options"
        onClick={
            async event => {
                let    [deleteImage,    deleteImageError]    =    await    axios({},
`${process.env.BACKEND_URL}/user/${localStorage.getItem('id')}/image`, 'DELETE', {token:
localStorage.getItem('token')});

                if (deleteImageError) return history.push('/error');

                store.userObj.avatar = null;
            }
        }
    >
        Delete profile image
    </p>
```

```
        <p
            className="main-view-profile-options"
        >
            Delete profile
        </p>
    </div>
})
export default MainViewProfile;
import React, {useEffect} from 'react';
import {observer} from "mobx-react";
import RoomObj from "./RoomObj";
import MainViewSearch from './MainViewSearch';
import {useHistory} from "react-router-dom";
import UserTile from "./UserTile";

const MainViewRooms = observer(({store}) => {
    let history = useHistory();

    return <div className="main-view-rooms">
        <MainViewSearch store={store} />
        {
            store.searchPeopleInput.length === 0
                ?
                store.getRooms().map(el => {
                    return <RoomObj room={el} key={el.id}/>
                })
                :
                store.foundedUsers.map(el => {
                    return <UserTile user={el} key={el.id} store={store}/>
                })
        }
    </div>
})

export default MainViewRooms;
import {observer} from "mobx-react";
import React from "react";
import {findUsersByNickname} from "../api/mainView";

const MainViewSearch = observer(({store}) => {
    return <input
        type="text"
        className="user-search-input"
        value={store.searchPeopleInput}
        onChange={
            async (e) => {
                store.changeSearchPeopleValue(e.target.value);
                let [result, resultError] = await findUsersByNickname(e.target.value);

                if (resultError) {
                    history.push('/error');
                }
                store.setFoundedUsers(result.data.data);
            }
        }
        placeholder="Input user nickname to find him"
    />
})

export default MainViewSearch;

import React from "react";
```

```
import {Link} from "react-router-dom";

export default function PageNotFound () {
    return(
        <h1>PAGE NOT FOUND</h1>
    )
}
import React from "react";
import {useHistory} from "react-router-dom";

export default function Popup({info, display, behaviour}) {
    let history = useHistory()
    return <div className={`over`} style={{display}}>
        <div className="over-cover">
        </div>
        <div className="popup">
            <p className={'popup-info'}>{info}</p>
            <button onClick={event => behaviour()}>OK</button>
            <div className="clear">
            </div>
        </div>
    </div>
}
import React, {useRef} from "react";
import {Link, useHistory} from "react-router-dom";
import {observer} from "mobx-react";
import keypair from "keypair";

import {validate} from 'email-validator';
import '../styles/Register.css';
import {JSEncrypt} from "jsencrypt";
import Popup from "./Popup";

const Register = observer(({store}) => {
    let backImageStyleObj = {
        backgroundImage: `url('./build/assets/defaultAvatar.png')`
    }
    let history = useHistory();

    return (
        <div className={'absolute-block'}>
            <Popup behaviour={() => {history.push('/login')}} info={'Your private and
public key were generated successfully, you can find them in root directory of
application'} display={store.showPopup}/>
            <div className="block-with-inputs">
                <div className="register-image" style={backImageStyleObj}>
                </div>
                <input  type="text"  className={`${store.emailInputStyle}  email-reg`}
value={store.emailInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'emailInputValue')
                }}
                       placeholder={"Enter your email"}
                />
                <input   type="text"   className={`${store.nameInputStyle}   name-reg`}
value={store.nameInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'nameInputValue')
                }}
                       placeholder={"Enter your name"}
                />
                <input  type="text"  className={`${store.lastNameInputStyle}  last-name-
reg`} value={store.lastNameInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'lastNameInputValue')
```

```
                }}
                       placeholder={"Enter your lastName"}
                />
                <input  type="text"  className={`${store.nicknameInputStyle}  nickname-
reg`} value={store.nicknameInputValue} onChange={(e) => {
                    store.changeInputValue(e.target.value, 'nicknameInputValue')
                }}
                       placeholder={"Enter your nickname"}
                />

                <button className="btn-reg-login" onClick={async () => {
                    let hasErrors = false;

                    let crypt = new JSEncrypt({default_key_size: 512, log: true});
                    crypt.getKey();
                    store.setPublicKey(crypt.getPublicKey());
                    store.setPrivateKey(crypt.getPrivateKey());

                    if         (store.emailInputValue         ===         ''         ||
!validate(store.emailInputValue)) {
                        store.changeInputStyle('warning', 'emailInputStyle');
                        hasErrors = true;
                    }

                    if (store.nameInputValue === '') {
                        store.changeInputStyle('warning', 'nameInputStyle');
                        hasErrors = true;
                    }

                    if (store.lastNameInputValue === '') {
                        store.changeInputStyle('warning', 'lastNameInputStyle');
                        hasErrors = true;
                    }

                    if (store.nicknameInputValue === '') {
                        store.changeInputStyle('warning', 'nicknameInputStyle');
                        hasErrors = true;
                    }

                    if (hasErrors) return;

                    Electron.files.keySave('keySave', store.publicKey, 'public');
                    Electron.files.keySave('keySave', store.privateKey, 'private');

                    let [response, error] = await store.registerUser();

                    if (error) {
                        console.log(error);
                        switch (error.status) {
                            case 409:
                                switch (error.errors.field) {
                                    case 'nickname':
                                        store.changeInputStyle('warning',
'nicknameInputStyle');
                                        Electron.errors.showError('error',  'User  with
such nickname already exist!', 'Registration error');
                                        break;
                                    case 'email':
                                        store.changeInputStyle('warning',
'emailInputStyle');
                                        Electron.errors.showError('error',  'User  with
such email already exist!', 'Registration error');
```

```
                                    break;
                                }
                                break;
                            case 400:
                                history.push('/error');
                                break;
                        }
                    }

                    // if (response) history.push('/login');
                    if (response) store.showPopup = 'block';
                }}>Register
                </button>

                <p>Or you can authorize by sending code on your email if you already have
account using <Link
                    to="/send-code" className="link">this</Link> link</p>
            </div>
        </div>
    )
})

export default Register;
import {useHistory} from "react-router-dom";
import React from 'react';
import socket from "../socket";

export default function RoomObj({room}) {
    let history = useHistory();
    let imageStyleObj = {
        backgroundImage: room.otherUser.avatar ? "url(" + process.env.BACKEND_URL + "/"
+ room.otherUser.avatar + ")" : "url('./build/assets/defaultAvatar.png')"
    }
    return (
        <div
            className="roomClass"
            onClick={
                () => {
                    socket.emit('joinRoom', `${room.id}${localStorage.getItem('id')}`);
                    history.push(`/room/${room.id}`)
                }
            }
        >
            <div className="roomClass-avatar" style={imageStyleObj}>
            </div>
            <div className="roomClass-info">
                <p    className="room-user-name">{room.otherUser.name    +    "    "    +
room.otherUser.lastName}</p>
                <p                                                    className="room-last-
message">{room.lastMessage?.messageObj.message}</p>
                <p className="room-last-message-time">{room.lastMessage?.createdAt}</p>
            </div>
            <div className="clear">
            </div>
        </div>
    )
}
import React from "react";
import {Link} from "react-router-dom";
import {observer} from "mobx-react";
import {useHistory} from 'react-router-dom';
import Popup from "./Popup";
```

```
import '../styles/SendCode.css';

const SendCode = observer(({store}) => {
    let history = useHistory();
    let backImageStyleObj = {
        backgroundImage: `url('./build/assets/defaultAvatar.png')`
    }
    return (
        <div className={'absolute-block'}>
            <Popup         behaviour={store.behaviour}         display={store.popupShow}
info={store.popupText} />
            <div className="block-with-inputs">
                <div className="register-image" style={backImageStyleObj}>
                </div>
                <input
                    className={store.classList + " send-code-input"}
                    type="text"
                    placeholder={store.placeholder}
                    value={store.sendCodeInputValue}
                    onChange={(e) => {
                        if (store.classList === 'warning') store.setNormalStyle();
                        store.setSendCodeInput(e.target.value)
                    }}
                />
                <button className={'btn-reg-login'} onClick={async () => {
                    if (store.sendCodeInputValue.length === 0) {
                        return store.setErrorStyle();
                    }

                    let [sent, error] = await store.sendCode(store.sendCodeInputValue);

                    if (error) {
                        switch (error.status) {
                            case 400: {
                                store.setErrorStyle();
                                break;
                            }
                            case 404: {
                                store.popupText      =      `User      with      email
${store.sendCodeInputValue} was not found`;
                                store.popupShow = 'block';
                                store.sendCodeInputValue = '';
                                break;
                            }
                        }
                    }

                    if (sent) {
                        history.push('/login');
                    }
                }}>
                    Send code
                </button>
                <p>
                    Or you can register if you doesnt have account already using <Link
to="/register" className={'link'}>this</Link> link
                </p>
            </div>
        </div>
    )
})
```

```
export default SendCode;
import React from "react";
import {Link} from "react-router-dom";

export default function Settings () {
    return(
        <Link to="/">To main</Link>
    )
}
import React from "react";
import {useHistory} from "react-router-dom";
import {getOrCreateRoom} from "../api/mainView";
import socket from "../socket";

export default function UserTile({user, store}) {
    let history = useHistory();
    let imageStyleObj = {
        backgroundImage: user.avatar ? "url(" + process.env.BACKEND_URL + "/" +
user.avatar + ")" : "url('./build/assets/defaultAvatar.png')"
    }

    return <div
        className="user-tile"
        onClick={
            async (e) => {
                let [room, roomError] = await getOrCreateRoom(user.id);
                socket.emit('joinRoom',
`${room.data.id}${localStorage.getItem('id')}`);
                store.changeSearchPeopleValue("");
                history.push(`/room/${room.data.id}`);
            }
        }
    >
        <div className="user-tile-avatar" style={imageStyleObj}>
        </div>
        <p>{user.name + " " + user.lastName}</p>
        <div className="clear">
        </div>
    </div>
}
import axios from '../utils/axios';

export async function sendCodeApi (email) {
    return await axios({email}, `${process.env.BACKEND_URL}/auth/send-code`, "POST",
{});
}

export async function getTokenApi (email, code) {
    return await axios({email, code}, `${process.env.BACKEND_URL}/auth`, "POST", {});
}
import axios from "../utils/axios";

export async function getRoomsApi () {
    return await axios(null, `${process.env.BACKEND_URL}/room`, "GET", {'token':
localStorage.getItem('token')});
}

export async function findUsersByNickname(search) {
    return await axios(null, `${process.env.BACKEND_URL}/user?nickname=${search}`,
'GET', {'token': localStorage.getItem('token')})
}
```

```
export async function getOrCreateRoom(userId) {
    return await axios(null, `${process.env.BACKEND_URL}/room/${userId}/user`, "GET",
{'token': localStorage.getItem('token')})
}
import axios from '../utils/axios';

export async function getMessagesApi(id) {
    return await axios(null, `${process.env.BACKEND_URL}/message/${id}/room`, "GET",
{'token': localStorage.getItem('token')});
}

export async function createMessageApi(data) {
    return await axios(data, `${process.env.BACKEND_URL}/message`, "POST", {
        token: localStorage.getItem('token')
    });
}
import axios from "../utils/axios";

export async function registerUserApi (email, name, lastName, nickname, publicKey) {
    return    await    axios({email,    name,    lastName,    nickname,    publicKey},
`${process.env.BACKEND_URL}/user`, "POST", {});
}
import axios from "../utils/axios";

export async function getRoomDetailApi(id) {
    return await axios(null, `${process.env.BACKEND_URL}/room/${id}`, "GET", {
        token: localStorage.getItem('token')
    });
}
import axios from "../utils/axios";

export async function getDataApi () {
    return await axios({}, `${process.env.BACKEND_URL}/user`, "GET", {
        token: localStorage.getItem('token')
    })
}

export async function getUserDataApi(id) {
    return await axios({}, `${process.env.BACKEND_URL}/user/${id}`, "GET", {
        token: localStorage.getItem('token')
    })
}
const fs = require('fs');

module.exports = (ipcMain, window) => {
    ipcMain.on('keySave', (event, key, type) => {
        switch (type) {
            case 'public': {
                fs.writeFileSync('./public.pem', key);
                break;
            }
            case 'private': {
                fs.writeFileSync('./private.pem', key);
                break;
            }
        }
    })

    // ipcMain.on('')
}
const
```

```
    {dialog} = require('electron');

module.exports = (ipcMain, window) => {
    ipcMain.on('error', (event, message, title) => {
        const options = {
            type: "none",
            buttons: ["Okay"],
            title: title,
            message: message
        }

        dialog.showMessageBoxSync(window, options);
    })
}
```

Додаток Д. Ілюстративні матеріали

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

РОЗРОБКА МЕТОДУ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАХИСТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ-МЕСЕНДЖЕРА З ВИКОРИСТАННЯМ АСИМЕТРИЧНОГО ШИФРУВАННЯ

Розробка методу та програмного забезпечення для захисту інформаційної системи-месенджера з використанням асиметричного шифрування.

Виконав студент групи 1ПІ-20м Пілецький В.Д.

Науковий керівник к.т.н доц. Катєльніков Д.І.

Актуальність теми

Користувачі можуть конфіденційно обмінюватись інформацією, доступ до якої можна отримати лише отримавши фізичний доступ до пристрою користувача з приватним ключем шифрування.

Мета

підвищення безпеки даних користувачів інформаційної системи месенджеру з використанням асиметричного шифрування.

Об’єкт дослідження

процес захисту інформаційної системи месенджера з використанням асиметричного шифрування.

Предмет дослідження

методи та алгоритми захисту інформації.

Методи дослідження

У процесі дослідження застосовувались методи шифрування інформації з асиметричним ключем шифрування

Наукова новизна

- Подальшого розвитку отримав метод захисту інформаційної системи месенджеру з використанням асиметричного шифрування;
- Подальшого розвитку отримав метод покращення швидкодії шифрування інформації асиметричним ключем шифрування, який на відміну від існуючих методів приймає у якості аргументів приймає не один публічний ключ та повідомлення яке потрібно зашифрувати, а 2 ключа шифрування, що дозволяє реалізувати паралельне шифрування обома ключами, а не послідовно, що покращило швидкодію.

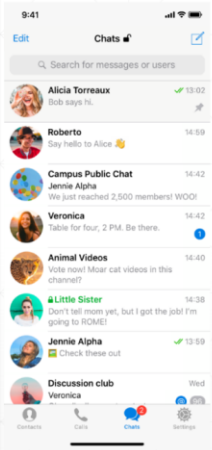
Практична цінність

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби захисту інформаційної системи месенджеру з використанням асиметричного шифрування.

Основні задачі

- – провести аналіз існуючих методів і засобів шифрування інформації для визначення напрямку підвищення криптографічної стійкості;
- – запропонувати новий:
  - – метод захисту інформаційної системи месенджеру;
  - – метод створення та читання зашифрованих повідомлень, таким чином, щоб для комунікації між двома користувачами не потрібно було зберігати ключі шифрування централізовано (на сервері)
- – розробити програмне забезпечення на основі запропонованих методів;
- – провести експериментальні дослідження розроблених засобів захисту інформаційної системи месенджеру.

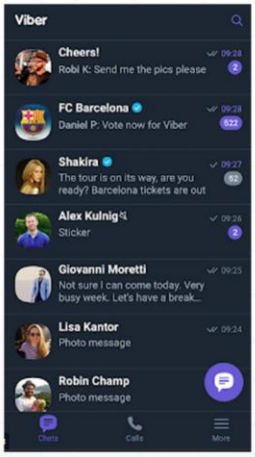
Аналоги



Telegram – кросплатформенна система миттєвого обміну повідомленнями (месенджер) з функціями VoIP, що дозволяє обмінюватися текстовими, голосовими та відеоповідомленнями, стікерами та фотографіями, файлами багатьох форматів. Також можна здійснювати відео- і аудіозвонкі, організовувати конференції, розраховані на багато користувачів групи і канали. Клієнтські програми Telegram доступні для Android, iOS, Windows Phone, Windows, macOS і GNU / Linux. Кількість щомісячних активних користувачів сервісу станом на січень 2021 року становить близько 500 млн осіб.

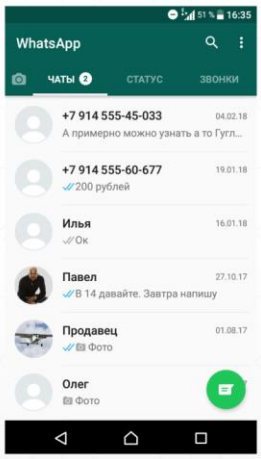
Недоліком даного месенджеру є те, що з максимальним рівнем захищеності повідомлення шифруються лише у спеціальних захищених чатах.

Аналоги



Viber – месенджер, який дозволяє відправляти повідомлення, здійснювати відео та голосові VoIP-дзвінки через інтернет. Голосові дзвінки між користувачами Viber повністю безкоштовні (оплачується лише інтернет трафік по тарифу оператору зв'язку). Viber дає можливість відправляти текстові, голосові, відео повідомлення, документи, зображення, відеозаписи та файли.
Для авторизації користувачів та пошуку контактів додаток використовує номер телефону та передає вміст телефонної адресної книги на сервери компанії Viber Media S.à r.l., Люксембург. Також вони збирають інформацію про здійснені дзвінки та передані повідомлення, довжину дзвінків, учасників дзвінків та чатів.
Недоліком є незручний спосіб синхронізації повідомлень, при втраті телефону є ймовірність втратити всю попередню переписку, а також наявність платних функцій.

Аналоги



WhatsApp - популярний месенджер для мобільних і інших платформ з підтримкою голосового зв'язку і відеозв'язку. Дозволяє пересилати текстові повідомлення, зображення, відео, аудіо, електронні документи та навіть програмні установки через Інтернет[6].
Клієнт працює на платформах Android, iOS, S40, KaiOS, а також Windows, macOS і у вигляді веб-додатки.
Компанія WhatsApp Inc., яка створила мессенджер, була заснована Яном Кумом і Брайаном Ектон 24 лютого 2009 року і розташована в Маунтін-В'ю, США; з жовтня 2014 року належить Facebook Inc. З 2016 додаток офіційно стало безкоштовним і до цього дня є таким, користувач оплачує лише використаний додатком інтернет-трафік. Додатком користується понад мільярд людей.
Недоліком даного додатку є те, що він часто піддається хакерським атакам, після чого дані користувачів сервісу потрапляють до рук зловмисників.

Основні функції додатку

- Відправлення та отримання зашифрованих повідомлень
- Завантаження та видалення фотографії профілю
- Авторизація за допомогою приватного ключа та коду з електронної пошти
- Пошук нових користувачів та створення з ними діалогів

Як влаштовано шифрування/розшифрування даних

Система працює таким чином:

При реєстрацій користувача на клієнті генеруються публічний та приватний ключ. Вони зберігаються у файли public.pem та private.pem відповідно, а публічний ключ користувача зберігається також на сервері.

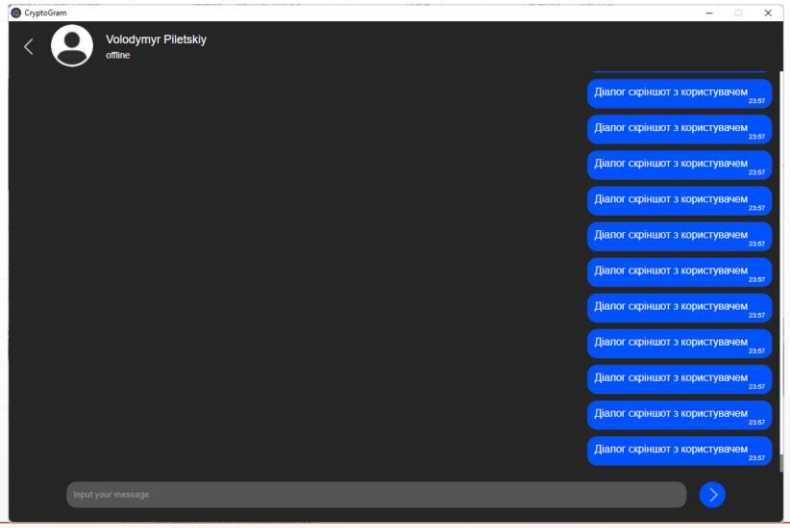
Коли один користувач надсилає повідомлення іншому користувачу, на сервер відправляється 2 версії повідомлення – одна зашифрована публічним ключем першого користувача, інша – публічним ключем другого користувача. Коли на сервер з клієнту відправляється запит на отримання списку повідомлень, сервер повертає лише версію повідомлення зашифровану публічним ключем користувача, який виконує запит.

Використання такої схеми дозволяє не зберігати приватні ключі користувачів на сервері, тобто навіть якщо зловмисник отримає доступ до бази даних або серверу, ніякої інформації він отримати не зможе, а якщо зловмисник отримає доступ до приватного ключа користувача, то він отримає доступ лише до його переписки, а не усієї системи.

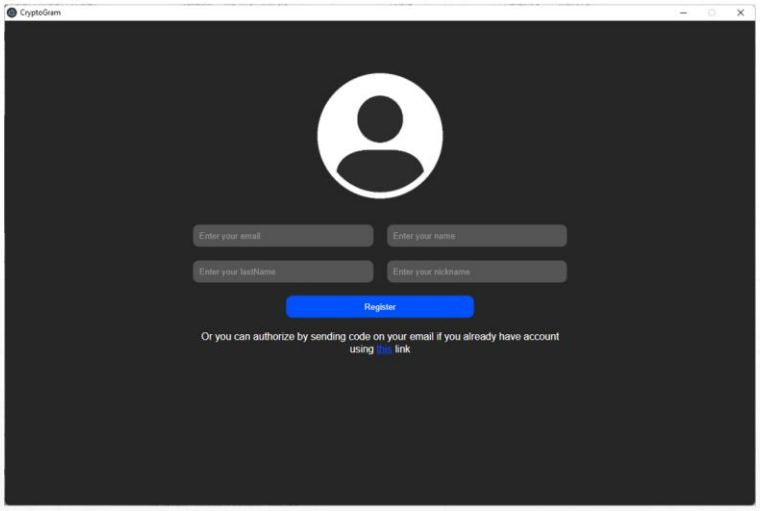
Головними перевагами такої схеми є більша криптографічна стійкість у порівнянні з системами з симетричними ключами шифрування, а також те, що отримати доступ до усіх даних практично неможливо, так як в такому випадку потрібно отримати доступ до приватних ключів усіх користувачів.

Головне вікно програми

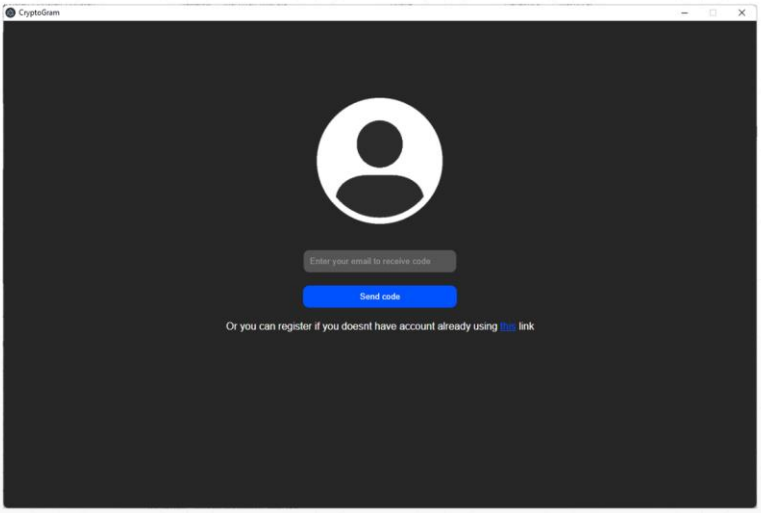
Діалог з користувачем



Реєстрація



Логін



Висновок

В результаті виконання роботи були вирішені наступні задачі:

- розроблено кросплатформений додаток для діалогів з користувачами;
- подальшого розвитку отримав метод захисту інформаційної системи месенджеру з використанням асиметричного шифрування;
- розроблено модуль шифрування даних з використанням асиметричного шифрування;
- розроблено модуль розшифрування даних з використанням асиметричного шифрування;
- розроблено модуль збереження/завантаження фотографії профілю.

ДЯКУЮ ЗА УВАГУ!