Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

«Розробка захищеного додатку обміну конфіденційними даними в корпоративній мережі на основі технології сокетів»

Виконав: студент 4 курсу, гр.2КІТС-20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Ільчук Р.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. МБІС
Карпінець В.В.
(прізвище та ініціали)
«6» червня 2024 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Городецька О.С.
(прізвище та ініціали)
«6» червня 2024 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
"6" червня 2024р.

Вінниця ВНТУ – 2024

Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 22 " березня 2024 р.

З А В Д А Н Н Я
на бакалаврську дипломну роботу студенту
Ільчуку Роману Валерійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи «Розробка захищеного додатку обміну конфіденційними даними в корпоративній мережі на основі технології сокетів»

Керівник роботи Карпінець Василь Васильович к.т.н., доцент каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від " 11 " березня 2024року № 80

2. Термін подання студентом роботи 10.06.2024 р.

3. Вихідні дані до роботи Електронні джерела та наукові статті по темі. Існуюче програмне забезпечення, яке стосується теми бакалаврської дипломної роботи.

4. Зміст текстової частини:

1. Огляд сучасного стану теорії і практичної реалізації проблеми безпечного обміну конфіденційними даними в корпоративній мережі на основі технологій сокетів
2. Проектування програмного додатку для обміну конфіденційними даними на технології сокетів
3. Розробка додатку та системи захисту

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)

Презентація у форматі PowerPoint з схемою алгоритму роботи додатку і скріншотами інтерфейсу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ І	Карпінець В.В. к.т.н., доцент каф. МБІС		
Розділ ІІ	Карпінець В.В. к.т.н., доцент каф. МБІС		
Розділ ІІІ	Карпінець В.В. к.т.н., доцент каф. МБІС.		

7. Дата видачі завдання 22 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Дослідження актуальності обраної теми	22.03.2024	24.03.2024	
2	Пошук та вивчення літератури за темою	25.03.2024	31.03.2024	
3	Аналіз методів захисту інформації	01.04.2024	07.04.2024	
4	Дослідження інформаційних джерел	08.04.2024	14.04.2024	
5	Вибір мови програмування та середовища розробки	15.04.2024	21.04.2024	
6	Розробка алгоритму роботи додатку	22.04.2024	28.04.2024	
7	Розроблення інтерфейсу додатку	29.04.2024	05.05.2024	
8	Тестування програмного модуля	06.05.2024	12.05.2024	
9	Оформлення матеріалів до захисту БДР	13.05.2024	22.05.2024	
10	Переддипломний захист	24.05.2024	25.05.2024	
11	Захист бакалаврської дипломної роботи	12.06.2024	17.06.2024	

Студент _____ Ільчук Р. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карпінець В. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана бакалаврська дипломна робота присвячена розробці захищеного додатку для обміну конфіденційними даними в корпоративній мережі з використанням технології сокетів. Метою дослідження є створення ефективного механізму захисту даних під час їх передачі, забезпечення конфіденційності, цілісності та автентичності інформації. Дипломна робота включає аналіз сучасних методів шифрування та захисту даних, а також використання криптографічних алгоритмів AES та SHA-3 для шифрування та хешування інформації. Для забезпечення безпеки передачі даних додаток реалізує механізми аутентифікації та авторизації користувачів за допомогою фреймворку Spring Security. Додаток забезпечує надійний захист корпоративної інформації та запобігає несанкціонованому доступу до конфіденційних даних під час їх обміну в мережі.

Ключові слова: захищений додаток, обмін даними, корпоративна мережа, сокети, AES, SHA-3, Spring Security, аутентифікація, авторизація, конфіденційність, цілісність, захист даних.

ABSTRACT

This bachelor thesis is devoted to the development of a secure application for exchanging confidential data in a corporate network using socket technology. The purpose of the research is to create an effective mechanism for protecting data during its transmission, ensuring confidentiality, integrity and authenticity of information. The thesis includes an analysis of modern encryption and data protection methods, as well as the use of AES and SHA-3 cryptographic algorithms for encryption and hashing of information. To ensure data transfer security, the application implements user authentication and authorization mechanisms using the Spring Security framework. The application ensures reliable protection of corporate information and prevents unauthorized access to confidential data during their exchange on the network.

Keywords: secure application, data exchange, corporate network, sockets, AES, SHA-3, Spring Security, authentication, authorization, privacy, integrity, data protection.

ЗМІСТ

ВСТУП 4

1. ОГЛЯД СУЧАСНОГО СТАНУ ТЕОРІЇ І ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ ПРОБЛЕМИ БЕЗПЕЧНОГО ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ В КОРПОРАТИВНІЙ МЕРЕЖІ НА ОСНОВІ ТЕХНОЛОГІЇ СОКЕТІВ 6

1.1 Теоретичні основи безпеки даних в корпоративних мережах 6

1.2 Методи обміну конфіденційними даними в корпоративних мережах .. 11

1.3 Практичні аспекти застосування технологій сокетів у безпечному обміні даними 13

1.4 Практичні приклади реалізації безпечного обміну даними за допомогою технології сокетів та алгоритмів 16

1.5 Висновки та постановка задач 20

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ НА ТЕХНОЛОГІЯХ СОКЕТІВ 22

2.1 Вибір підходу до захисту 22

2.2 Розробка алгоритму роботи додатку 25

2.3 Аргументація вибору середовища розробки та мови програмування ... 29

2.4. Висновок до розділу 32

3.РОЗРОБКА ДОДАТКУ ТА СИСТЕМИ ЗАХИСТУ 33

3.1 Програмна реалізація додатку 33

3.2. Інструкція користувача 44

3.3. Тестування програмного додатку 47

3.4. Висновок до розділу 49

ВИСНОВКИ 51

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52

ДОДАТКИ 55

Додаток А. Технічне завдання.....**Error! Bookmark not defined.**

Додаток Б. Лістинг програми 59

Додаток В. Ілюстративний матеріал 74

Додаток Г. Протокол перевірки на атиплагіат**Error! Bookmark not defined.**

ВСТУП

Актуальність задачі. З розвитком інформаційних технологій та збільшенням обсягів корпоративних даних, забезпечення безпеки при обміні конфіденційною інформацією в корпоративних мережах стало критично важливим. В умовах зростання кіберзагроз та підвищення вимог до конфіденційності даних, з'являється необхідність у створенні надійних та безпечних засобів комунікації. Використання технології сокетів для розробки додатків обміну даними є перспективним напрямком, що дозволяє забезпечити високу швидкість та безпеку передачі інформації.

Метою даної роботи є розробка захищеного додатку обміну конфіденційними даними в корпоративній мережі на основі технології сокетів. Основною метою є створення безпечного та конфіденційного середовища для обміну повідомленнями, яке забезпечує захист від зловмисників та зберігає приватність користувачів. Це дозволить забезпечити високу надійність та безпеку для комунікацій, враховуючи сучасні загрози та вимоги до конфіденційності.

Об'єктом дослідження є процес розробки захищеного додатку для обміну конфіденційними даними на основі технології сокетів. Дослідження включає аналіз існуючих рішень у сфері корпоративних комунікацій та протоколів передачі даних, розробку архітектури додатку, його реалізацію та перевірку ефективності і безпеки запропонованого рішення.

Предметом дослідження є розробка додатку для обміну конфіденційними даними в корпоративній мережі на основі технології сокетів. Дослідження включає проектування архітектури додатку, реалізацію комунікаційного протоколу з додатковими заходами безпеки, та інтеграцію цього рішення у функціональний додаток для корпоративного використання.

Новизна роботи. Основною новизною даної роботи є поєднання технології сокетів з сучасними методами криптографічного захисту даних для створення безпечного середовища комунікації. Використання додаткових заходів безпеки, таких як шифрування даних та автентифікація користувачів, ускладнює процеси перехоплення та розшифрування інформації для зловмисників. Цей інноваційний

підхід може забезпечити вищий рівень безпеки та конфіденційності у корпоративних мережах.

Практична цінність. Результати даної роботи можуть мати велику практичну цінність для розробників корпоративних додатків та користувачів. Розроблений додаток на основі технології сокетів може стати альтернативою існуючим рішенням з вищим рівнем безпеки та конфіденційності. Використання сучасних методів криптографічного захисту додатково ускладнює процеси перехоплення та розшифрування даних для зловмисників. Це може допомогти у забезпеченні приватності та безпеки обміну конфіденційною інформацією в корпоративних мережах, зменшенні ризику витоку даних та підвищенні загального рівня кібербезпеки.

1. ОГЛЯД СУЧАСНОГО СТАНУ ТЕОРІЇ І ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ ПРОБЛЕМИ БЕЗПЕЧНОГО ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ В КОРПОРАТИВНІЙ МЕРЕЖІ НА ОСНОВІ ТЕХНОЛОГІЇ СОКЕТІВ

В сучасному світі, де цифрові технології проникають у всі сфери бізнесу, захист конфіденційної інформації в корпоративних мережах стає надзвичайно важливою задачею. Зловмисники намагаються отримати доступ до конфіденційних даних з метою використання їх у шкідливих цілях, що може призвести до серйозних фінансових втрат, порушення репутації та навіть правових наслідків для компаній.

У зв'язку з цим, існує постійна необхідність вдосконалення технологій та методів захисту конфіденційних даних. Сучасні корпоративні мережі використовують різноманітні технології, такі як шифрування даних, мережеві файерволи, ідентифікація користувачів та інші методи для забезпечення безпеки інформації.

Незважаючи на існуючі технології захисту, виникають нові виклики, такі як зростаюча складність атак, соціальний інженерінг та внутрішні загрози з боку співробітників компанії. Тому важливо постійно аналізувати та вдосконалювати методи захисту конфіденційних даних для забезпечення ефективного функціонування корпоративних мереж у сучасних умовах.

1.1 Теоретичні основи безпеки даних в корпоративних мережах

У розділі "Теоретичні основи безпеки даних в корпоративних мережах" ми звертаємо увагу на основні концепції та принципи, які становлять основу безпеки інформації в мережевих середовищах організацій. Розуміння цих фундаментальних аспектів дозволяє краще усвідомити загрози, з якими стикаються корпоративні мережі, та розробляти відповідні стратегії захисту.

Однією з ключових складових безпеки є виявлення загроз. У сучасному світі, де відбувається стрімкий розвиток технологій, мережеві атаки набувають все більшої складності. Зловмисники постійно вдосконалюють свої методи, щоб

здійснювати атаки на інфраструктуру компаній, використовуючи для цього різноманітні види програмного та апаратного забезпечення. Отже, важливо аналізувати та розуміти потенційні загрози, з якими може стикнутися корпоративна мережа, щоб вчасно приймати відповідні заходи для їх запобігання [1].

Далі, необхідно розглянути основні принципи безпеки даних. Конфіденційність, цілісність та доступність (іноді відомі як CIA-тріада) є керованими концепціями в сфері безпеки інформації. Конфіденційність визначає, що дані доступні лише тим, хто має на це право. Цілісність означає, що дані залишаються незмінними і не піддаються модифікації несанкціонованими користувачами. А доступність гарантує, що дані будуть доступні в потрібний момент для авторизованих користувачів. Розуміння цих принципів є важливим для розробки ефективних стратегій захисту даних у корпоративній мережі [2].

Додатково, розглянемо значення застосування стандартів і протоколів безпеки. У сфері інформаційної безпеки велике значення має використання стандартизованих методів та протоколів, які забезпечують безпеку обміну даними. Наприклад, шифрування даних за допомогою алгоритмів AES або RSA дозволяє захистити інформацію від несанкціонованого доступу. На риснку (1.1) наведено схему роботи алгоритму RSA.

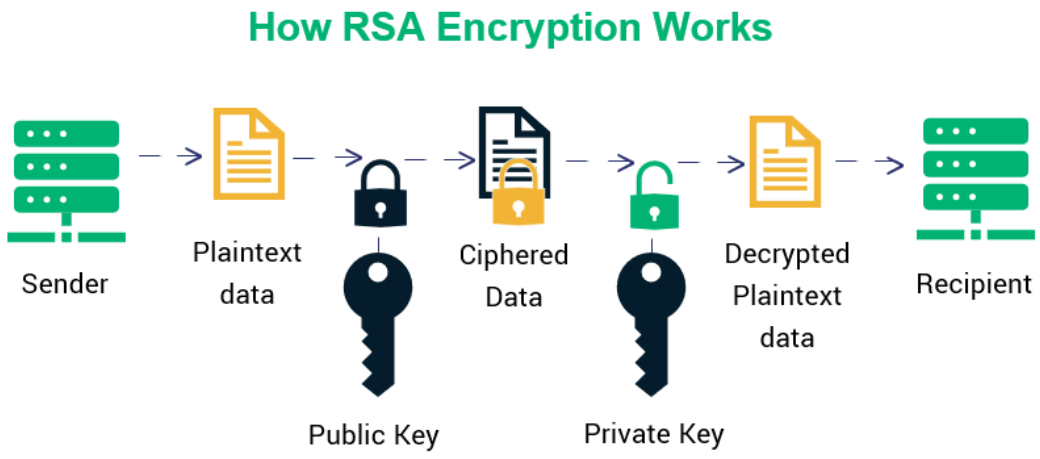


Рисунок 1.1 – Схема роботи алгоритму RSA

Протоколи безпеки, такі як SSL/TLS, забезпечують безпечне з'єднання між клієнтом та сервером, що є критично важливим для забезпечення

конфіденційності та цілісності даних.

Загалом, розуміння теоретичних аспектів безпеки даних в корпоративних мережах є ключовим для ефективного захисту інформації та запобігання можливим загрозам. Врахування цих аспектів допомагає розробникам та адміністраторам мереж розробляти та впроваджувати ефективні стратегії захисту, які відповідають сучасним викликам у сфері інформаційної безпеки [3].

Важливим аспектом забезпечення безпеки даних є також розуміння різних типів загроз, з якими можуть стикнутися організації. Загрози можуть включати технічні атаки, такі як віруси, черви та хакерські атаки, а також соціально-інженерні методи, такі як фішинг та соціальний інженерінг. Розуміння цих різноманітних загроз є критично важливим для ефективного захисту інформації. Важливо також враховувати роль людей у забезпеченні безпеки даних. Навчання персоналу та підвищення їхньої свідомості щодо питань безпеки є важливою складовою ефективної стратегії захисту. Забезпечення того, щоб персонал розумів потенційні загрози та знаходився на висоті з вимогами безпеки, може допомогти у запобіганні витоків даних та інших інцидентів [4].

Месенджери займають важливе місце в комунікації в корпоративних мережах. Забезпечення мережевої безпеки для таких платформ є важливим завданням [5]. Огляд безпекових функцій деяких популярних месенджерів

– Trello забезпечує безпеку даних користувачів за допомогою протоколу HTTPS для захищеного з'єднання та автентифікації користувачів через OAuth. Дані зберігаються в зашифрованому вигляді на серверах, а доступ до них контролюється через ролі та дозволи.

– Asana використовує HTTPS для захисту даних під час передачі та OAuth для безпечної автентифікації. Дані шифруються як під час передачі, так і під час зберігання. Asana пропонує розширені можливості контролю доступу з ролями та дозволами для забезпечення конфіденційності інформації.

– Slack забезпечує захист даних користувачів через протокол HTTPS та автентифікацію через OAuth. Дані зберігаються в зашифрованому вигляді, а всі файли та повідомлення шифруються під час передачі. Slack підтримує

двофакторну автентифікацію (2FA) та пропонує різноманітні налаштування для контролю доступу та управління безпекою на рівні організації.

– Signal використовує наскрізне шифрування для всіх повідомлень, дзвінків та файлів. Це забезпечує конфіденційність вмісту, який може бути прочитаний тільки відправником і отримувачем.

– Podio забезпечує безпеку даних через шифрування під час передачі та зберігання (HTTPS), двофакторну автентифікацію (2FA) і контроль доступу з налаштуванням ролей і дозволів. Інтеграції здійснюються через безпечну автентифікацію OAuth.

Вони забезпечують зв'язок між користувачами через Інтернет, використовуючи смартфони, комп'ютери або інші пристрої [6]. Далі буде зображено процес відправки та отримання повідомлення в меседжерах рисунок (1.2).

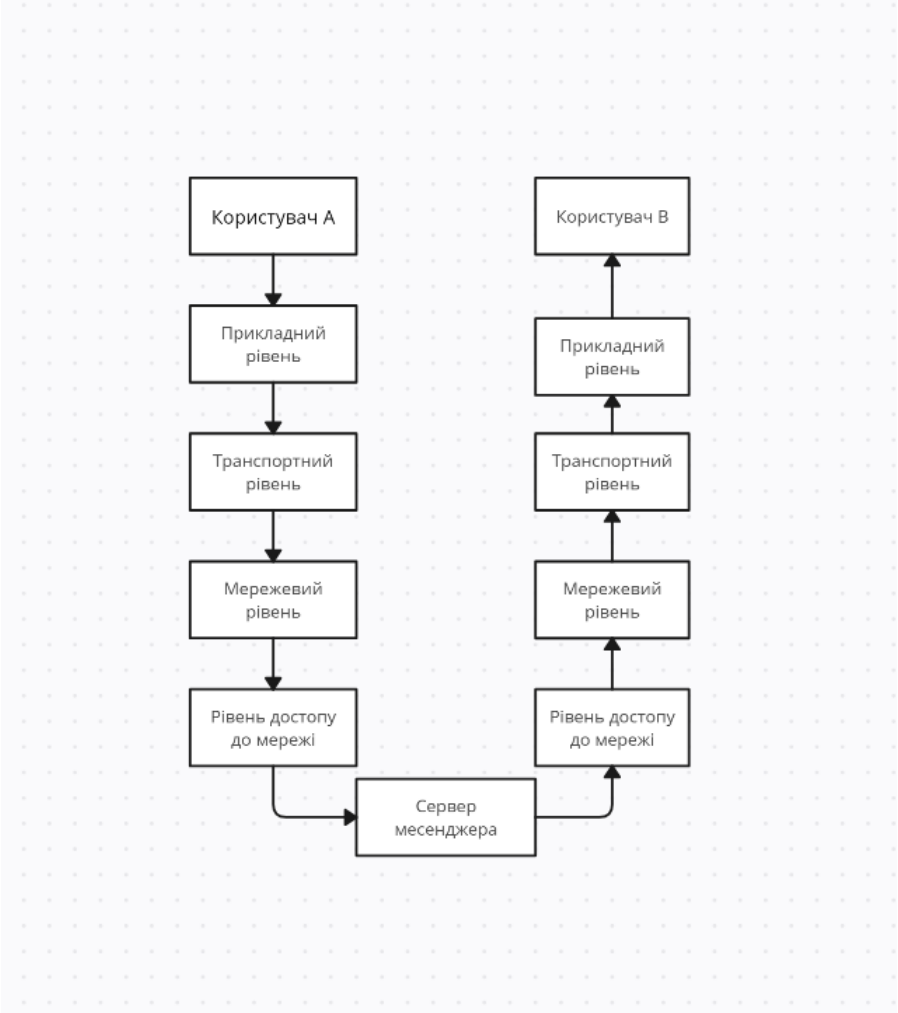


Рисунок 1.2 – Схема відправки та отримування повідомлення

Схема роботи більшості месенджерів включає кілька основних етапів, які охоплюють відправку, передачу та отримання повідомлень. Послідовна схема роботи месенджера:

1. Користувач відправляє повідомлення:

– Користувач вводить текстове повідомлення або інший тип повідомлення (наприклад, зображення, відео) в інтерфейсі месенджера і натискає кнопку "Відправити".

– Месенджер формує повідомлення і додає необхідні метадані (наприклад, час відправлення, ідентифікатор користувача).

2. Прикладний рівень (Application Layer):

– Повідомлення передається на сервер месенджера через інтернет за допомогою протоколів прикладного рівня (наприклад, XMPP, HTTP/HTTPS, WebSocket).

– Сервер може виконувати функції автентифікації та авторизації користувача, зберігання історії повідомлень, керування станом користувачів (онлайн/офлайн) та інших функцій.

3. Транспортний рівень (Transport Layer):

– Повідомлення інкапсулюється в сегменти TCP або датаграми UDP для надійної передачі даних.

– Для забезпечення надійної доставки (TCP) використовується підтвердження отримання сегментів та повторна передача у випадку втрат.

4. Мережевий рівень (Internet Layer):

– Сегменти TCP або датаграми UDP інкапсулюються в IP-пакети, які містять адресу джерела і призначення.

– IP-пакети маршрутизуються через інтернет до сервера месенджера.

5. Рівень доступу до мережі (Network Access Layer):

– IP-пакети передаються через фізичну мережу (наприклад, Ethernet, Wi-Fi) у вигляді кадрів (frames).

– Кадри досягають мережевого обладнання (наприклад, маршрутизаторів, комутаторів) і перенаправляються до місця призначення.

6. Обробка на сервері:

– Сервер месенджера отримує повідомлення, виконує необхідну обробку (наприклад, перевірку вмісту, збереження у базі даних, пересилання отримувачеві).

– Сервер визначає місцезнаходження отримувача (онлайн або офлайн) і передає повідомлення далі.

7. Передача до отримувача:

– Повідомлення передається до кінцевого пристрою отримувача за аналогічною схемою (мережевий рівень → транспортний рівень → прикладний рівень).

8. Користувач отримує повідомлення:

– Прикладний рівень отримує повідомлення, і воно відображається у інтерфейсі месенджера отримувача.

– Користувач бачить повідомлення і може відповідати на нього.

1.2 Методи обміну конфіденційними даними в корпоративних мережах

Одним з ключових аспектів обміну даними в корпоративних мережах є передача даних через мережу. Для забезпечення безпеки такого обміну використовуються різні підходи. Наприклад, синхронний обмін даними передбачає передачу даних між клієнтом і сервером у режимі реального часу, коли кожен крок обміну даними чекає на завершення попереднього кроку, що гарантує послідовність та надійність обміну.

З іншого боку, асинхронний обмін даними дозволяє клієнту і серверу взаємодіяти без очікування один одного. Це може бути корисно в умовах, коли виникають затримки в передачі даних або коли клієнт і сервер мають різні темпи обробки даних. Обидва ці підходи можуть використовуватися для забезпечення безпечного обміну конфіденційними даними, але вони мають свої переваги та обмеження, які слід враховувати при розробці стратегій захисту [7].

Окрім методів передачі даних, важливо також розглянути використання сокетів для забезпечення безпечного з'єднання між клієнтом і сервером. Сокети є

інтерфейсом додатків (API), який дозволяє програмам взаємодіяти через мережу. Вони можуть бути використані для створення з'єднань на рівні TCP або UDP, в залежності від потреб конкретного застосування [8].

TCP (Transmission Control Protocol) забезпечує надійне та послідовне з'єднання між клієнтом і сервером, що гарантує доставку даних у порядку, в якому вони були відправлені. У той же час, UDP (User Datagram Protocol) пропонує більш швидку, але менш надійну передачу даних, оскільки він не гарантує доставку даних або їх послідовність [9].

Обидва ці протоколи можуть бути застосовані для забезпечення безпечного обміну конфіденційними даними, але їх вибір залежить від конкретних вимог щодо швидкості, надійності та інших факторів.

Розглянувши ці аспекти методів обміну конфіденційними даними та використання сокетів, можна краще зрозуміти різні стратегії та підходи до забезпечення безпеки даних у корпоративних мережах.

Електронна пошта з шифруванням

Електронна пошта є одним із найпоширеніших способів обміну інформацією. Для забезпечення конфіденційності використовуються методи шифрування, такі як S/MIME (Secure/Multipurpose Internet Mail Extensions), які шифрують повідомлення та вкладення, захищаючи їх від несанкціонованого доступу [10].

– Віртуальні приватні мережі (VPN)

VPN створює захищений тунель для передачі даних між двома кінцевими точками через Інтернет. Це забезпечує конфіденційність і цілісність даних, оскільки весь трафік шифрується, що робить його недоступним для перехоплення [9].

– Безпечні протоколи передачі даних

Застосування захищених протоколів, таких як HTTPS (Hypertext Transfer Protocol Secure), FTPS (FTP Secure), SFTP (SSH File Transfer Protocol), дозволяє забезпечити захист даних під час їх передачі через мережу. Ці протоколи використовують шифрування для забезпечення конфіденційності та цілісності переданих даних [11].

– Системи управління документами (DMS)

Системи управління документами забезпечують централізоване зберігання та обмін документами з можливістю контролю доступу і аудиту. Це знижує ризик несанкціонованого доступу до конфіденційної інформації шляхом впровадження політик доступу та реєстрації дій користувачів [12].

– Платформи для обміну файлами

Платформи, такі як Microsoft SharePoint, Google Drive for Business, Dropbox Business, надають можливість безпечного обміну файлами з підтримкою шифрування та контролю доступу. Ці сервіси забезпечують зручність доступу до файлів та одночасно високий рівень безпеки [13].

– Чати та месенджери з шифруванням

Використання месенджерів, які підтримують наскрізне шифрування, таких як Signal і WhatsApp, забезпечує захищений обмін повідомленнями та файлами в реальному часі. Наскрізне шифрування гарантує, що повідомлення можуть читати лише відправник та отримувач [14].

1.3 Практичні аспекти застосування технологій сокетів у безпечному обміні даними

Технології сокетів забезпечують ефективний і безпечний обмін даними в корпоративних мережах, надаючи можливість прямого підключення між клієнтом і сервером. Сокети використовують різні протоколи, такі як TCP (Transmission Control Protocol) і UDP (User Datagram Protocol). TCP-сокети забезпечують надійну передачу даних, де важлива цілісність, тоді як UDP-сокети використовуються для швидкої передачі, де затримка має критичне значення [15].

Для забезпечення безпеки даних, що передаються через сокети Document Management System, застосовуються методи шифрування. Найпоширенішим протоколом є SSL/TLS (Secure Sockets Layer / Transport Layer Security), який забезпечує конфіденційність і цілісність даних за допомогою асиметричного шифрування для обміну ключами та симетричного для передачі даних. Інший популярний метод – SSH (Secure Shell), який використовується для захищених

з'єднань, забезпечуючи шифрування даних і аутентифікацію користувачів [16].

Сокети представляють собою кінцеві точки двостороннього зв'язку між двома програмами, що працюють в мережі. Вони використовують різні протоколи, такі як TCP (Transmission Control Protocol) та UDP (User Datagram Protocol), для передачі даних.

Основи роботи сокетів

– TCP-сокети: Забезпечують надійний і послідовний потік даних між клієнтом і сервером. Вони використовуються для застосувань, де важлива цілісність даних, наприклад, у фінансових транзакціях.

– UDP-сокети: Забезпечують швидку передачу даних без гарантії їхньої доставки. Вони використовуються для застосувань, де важлива швидкість передачі, наприклад, у потоковому відео [17].

Створення захищеного з'єднання через сокети включає кілька кроків: налаштування сокета (вибір протоколу, визначення адреси та порту), ініціалізацію SSL/TLS (завантаження сертифікатів і ключів, налаштування параметрів шифрування), встановлення з'єднання (виконання SSL/TLS рукостискання), обмін даними (шифрування і передача через захищений канал, розшифрування на приймаючій стороні) і завершення з'єднання (коректне завершення сеансу та закриття сокета).

Приклади використання сокетів включають веб-сервери, що використовують їх для обробки HTTP-запитів, перетворюючи їх у захищені HTTPS за допомогою SSL/TLS, фінансові системи, що використовують захищені сокети для передачі конфіденційних фінансових даних, та інструменти віддаленого управління, такі як SSH, для захищеного доступу до серверів [18].

Виклики при використанні сокетів включають необхідність забезпечення надійної аутентифікації користувачів і серверів, управління криптографічними ключами і вплив шифрування на продуктивність системи. Для аутентифікації використовуються сертифікати і ключі, для управління ключами – апаратні засоби безпеки (HSM), а для оптимізації продуктивності – апаратне прискорення шифрування.

Використання технологій сокетів для безпечного обміну даними в корпоративних мережах забезпечує високий рівень безпеки та ефективність передачі даних за умови правильної конфігурації та управління шифруванням, аутентифікацією і ключами.

Технології сокетів також використовуються в інших додатках, таких як системи миттєвих повідомлень, IoT (Internet of Things) пристрої, та мультимедійні програми для потокової передачі аудіо і відео. В таких додатках безпека і продуктивність залишаються ключовими пріоритетами. Наприклад, в системах миттєвих повідомлень часто використовуються наскрізне шифрування та складні методи аутентифікації, щоб захистити дані від несанкціонованого доступу [19].

Інша важлива сфера використання технологій сокетів – це забезпечення безпеки в хмарних обчисленнях. Використання захищених сокетів дозволяє гарантувати, що дані, передані між хмарними сервісами та кінцевими користувачами, залишаються конфіденційними та непідробленими. Хмарні провайдери часто використовують SSL/TLS для захисту трафіку і забезпечення безпечного доступу до ресурсів [20].

Надійність і безпека технологій сокетів також мають важливе значення для критично важливих систем, таких як медичні інформаційні системи та інфраструктура національної безпеки. В таких випадках важливо забезпечити не лише конфіденційність і цілісність даних, але й їх доступність і своєчасну доставку.

Для впровадження та підтримки безпечних з'єднань на основі сокетів важливо також враховувати вимоги до моніторингу і журналювання. Постійний моніторинг мережевої активності допомагає виявляти потенційні загрози і порушення безпеки, а журналювання забезпечує запис всіх подій для подальшого аналізу та аудиту.

Загалом, використання технологій сокетів для безпечного обміну даними в корпоративних мережах забезпечує високий рівень безпеки та ефективність передачі даних за умови правильної конфігурації та управління шифруванням, аутентифікацією і ключами. Ретельне планування, налаштування і регулярний

моніторинг є ключовими факторами успішного впровадження цих технологій у корпоративних середовищах [21].

1.4 Практичні приклади реалізації безпечного обміну даними за допомогою технології сокетів та алгоритмів

Одним із простих і водночас ефективних прикладів використання сокетів для безпечного обміну даними є створення захищеного чат-сервера. У цьому випадку сервер створює сокет і слухає підключення від клієнтів на певному порту. Клієнти підключаються до сервера через створені сокети. Для забезпечення конфіденційності використовується SSL/TLS шифрування, яке гарантує захист переданих даних. Бібліотека OpenSSL може бути використана для налаштування SSL/TLS. Клієнти шифрують повідомлення перед відправкою на сервер, який приймає зашифровані повідомлення, розшифровує їх і пересилає іншим клієнтам у зашифрованому вигляді [22].

Захищена передача файлів є критично важливою для корпоративних мереж, особливо коли йдеться про конфіденційні дані. Сервер створює сокет і слухає підключення на визначеному порту, тоді як клієнт підключається до сервера через сокет для передачі файлів. Використовується SSL/TLS для шифрування всіх даних, що передаються між клієнтом і сервером, забезпечуючи таким чином конфіденційність і цілісність інформації. Файли розбиваються на менші блоки, які шифруються та передаються окремо. Після отримання кожного блоку клієнт відправляє серверу підтвердження, що допомагає забезпечити цілісність переданих даних.

Віддалене керування серверами за допомогою захищених сокетів дозволяє адміністраторам безпечно управляти системами з будь-якої точки світу. Серверний додаток слухає підключення на визначеному порту, а клієнтський додаток підключається до сервера через захищене з'єднання. Всі команди і відповіді передаються через захищений канал, використовуючи SSL/TLS шифрування. Для підтвердження особи клієнта і сервера використовуються цифрові сертифікати. Адміністратор відправляє зашифровані команди серверу,

який їх виконує і повертає результат назад клієнту [23].

Розробка внутрішньокорпоративного месенджера дозволяє співробітникам безпечно обмінюватися повідомленнями та файлами всередині корпоративної мережі. Сервер слухає підключення від клієнтів на визначеному порту, тоді як клієнти підключаються до сервера, створюючи сокети для зв'язку. Всі повідомлення і файли шифруються за допомогою SSL/TLS. OpenSSL забезпечує налаштування SSL/TLS для обох сторін з'єднання. Повідомлення шифруються і передаються між клієнтами через сервер, файли розбиваються на блоки, шифруються і передаються через захищене з'єднання. Сервер також може мультиплексувати повідомлення між декількома клієнтами для реалізації групових чатів.

Ці приклади демонструють, як технологія сокетів може бути використана для створення безпечних додатків для обміну конфіденційними даними. Використання SSL/TLS забезпечує високий рівень безпеки, захищаючи дані від несанкціонованого доступу та перехоплення [24].

SHA-3 (Secure Hash Algorithm 3) використовується для хешування даних у системах обміну конфіденційною інформацією.

SHA-3 є найновішим членом родини алгоритмів Secure Hash Algorithm, розроблених Національним інститутом стандартів і технологій (NIST). Він був обраний у 2012 році після всеохоплюючого конкурсу, який тривав кілька років і залучив найкращі криптографічні уми з усього світу. Цей конкурс забезпечив ретельну перевірку та аналіз усіх кандидатів, що гарантує високу надійність і безпеку обраного алгоритму. SHA-3 відомий своєю стійкістю до багатьох видів атак, включаючи атаки на колізії та атаки на передобрази, що робить його надійним вибором для захисту даних [25].

Однією з ключових переваг SHA-3 є його принципово нова структура, яка відрізняється від попередніх алгоритмів SHA-1 і SHA-2. Ця нова структура базується на криптографічній губці (sponge construction), що надає додаткову гнучкість і безпеку. Вона дозволяє SHA-3 бути стійким до нових видів атак, які можуть бути ефективними проти алгоритмів з традиційною структурою. Крім

того, криптографічна губка забезпечує високу ступінь паралелізму, що дозволяє ефективно використовувати SHA-3 на сучасному апаратному забезпеченні, включаючи багатоядерні процесори та апаратні прискорювачі.

SHA-3 також надає різні варіанти довжини хеш-значення (224, 256, 384 та 512 біт), що дозволяє гнучко налаштовувати алгоритм під конкретні потреби проекту. Це забезпечує додатковий рівень безпеки, оскільки можна вибрати оптимальний розмір хешу для різних застосунків, балансуючи між безпекою та продуктивністю.

Інтеграція SHA-3 в існуючі системи є досить простою завдяки його стандартам і сумісності з широким спектром програмного забезпечення та криптографічних бібліотек. Це дозволяє легко оновлювати або замінювати існуючі алгоритми хешування, такі як SHA-1 або SHA-2, на SHA-3 без значних змін в інфраструктурі.

SHA-3 також має перевагу у використанні у ресурсомістких середовищах завдяки своїй ефективності та оптимізації для апаратного прискорення. Це робить його придатним для використання в різних пристроях, включаючи мобільні пристрої та IoT (Internet of Things), де ресурси обмежені, але вимоги до безпеки залишаються високими [26].

Завдяки своїй сучасній структурі, гнучкості, високій стійкості до криптографічних атак та ефективності, SHA-3 є відмінним вибором для забезпечення цілісності даних та автентифікації у системах обміну конфіденційною інформацією. Він забезпечує найвищий рівень безпеки, підтверджений численними незалежними оцінками та аналізами, що робить його надійним і перспективним рішенням для сучасних та майбутніх систем.

AES (Advanced Encryption Standard) є одним з найбільш популярних та широко використовуваних симетричних алгоритмів шифрування у світі, і його вибір для захисту конфіденційних даних у системах обміну інформацією має ряд важливих переваг.

AES є надзвичайно надійним та стійким до криптографічних атак. Він був розроблений для забезпечення високого рівня безпеки і був затверджений

Національним інститутом стандартів та технологій (NIST) у 2001 році як офіційний стандарт шифрування для урядових установ США. Відтоді AES став галузевим стандартом для шифрування даних у багатьох комерційних та урядових застосунках по всьому світу.

AES підтримує різні довжини ключів — 128, 192 та 256 біт, що дозволяє балансувати між продуктивністю та рівнем безпеки. Наприклад, для більшості комерційних застосунків використання 128-бітного ключа забезпечує достатню безпеку та високу швидкість шифрування, тоді як для критично важливих або високоризикованих систем можна використовувати 256-бітний ключ, що значно підвищує стійкість до зламів.

AES є надзвичайно ефективним з точки зору продуктивності. Він оптимізований для швидкого виконання як на програмному, так і на апаратному рівнях. Багато сучасних процесорів мають апаратну підтримку AES, що дозволяє значно прискорити операції шифрування та дешифрування. Це робить AES ідеальним вибором для додатків, де потрібна висока швидкість обробки даних, таких як реального часу системи обміну повідомленнями або потокове шифрування даних [27].

Четверта причина вибору AES полягає в його широкій підтримці та стандартизації. Завдяки своїй популярності та широкому використанню, AES підтримується багатьма програмними бібліотеками та криптографічними інструментами. Це забезпечує легку інтеграцію алгоритму в різні програмні платформи та середовища розробки, а також полегшує роботу розробників з реалізації та обслуговування шифрувальних систем.

Крім того, AES має просту та зрозумілу структуру, що полегшує його впровадження та перевірку на правильність. Відкритість та детальна документація алгоритму дозволяють розробникам легко зрозуміти його роботу та забезпечити правильне впровадження у своїх додатках. Це знижує ризик помилок у реалізації, які можуть призвести до вразливостей у системі.

Зрештою, AES має відмінну репутацію серед криптографів і був ретельно досліджений та проаналізований з моменту його впровадження. Багато років

незалежних оцінок і перевірок підтвердили його безпеку і надійність, що робить AES надійним вибором для захисту конфіденційних даних у сучасних системах.

Зважаючи на ці причини, AES є ідеальним вибором для реалізації захисту даних у системах обміну конфіденційною інформацією. Він забезпечує високий рівень безпеки, ефективність, продуктивність і сумісність, що робить його одним з найбільш надійних та широко визнаних алгоритмів шифрування на сьогоднішній день [28].

1.5 Висновки та постановка задач

У даному розділі було проведено аналіз методів захисту даних під час їх передачі в корпоративній мережі, розглянуто існуючі методи забезпечення конфіденційності та цілісності інформації, а також досліджено технології шифрування та аутентифікації. Зокрема, було вивчено криптографічні алгоритми AES для шифрування даних та SHA-3 для хешування, що забезпечує цілісність та автентичність інформації. Було також розглянуто можливості фреймворку Spring Security для реалізації аутентифікації та авторизації користувачів.

Проведене дослідження дозволило визначити основні вимоги та виклики у розробці захищеного додатку для обміну конфіденційними даними в корпоративній мережі. Таким чином, у процесі проведення загального огляду теоретичного матеріалу, були поставлені наступні задачі:

– Розробити алгоритм роботи програмного додатку: Визначити основні компоненти та функції додатку, які забезпечуватимуть захищений обмін даними. Алгоритм має включати механізми шифрування, хешування, аутентифікації та авторизації користувачів.

– Здійснити програмну реалізацію додатку: Використовуючи технологію сокетів, реалізувати передачу даних між клієнтами та сервером з використанням алгоритмів AES та SHA-3, а також інтегрувати фреймворк Spring Security для управління доступом та аутентифікацією користувачів.

– Дослідити коректність та ефективність роботи розробленої програми: Провести тестування додатку для перевірки його працездатності та відповідності

вимогам безпеки. Тестування має включати перевірку правильності шифрування та розшифрування даних, функцій аутентифікації та авторизації, а також захисту від несанкціонованого доступу.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ НА ТЕХНОЛОГІЯХ СОКЕТІВ

У цьому розділі детально розглядається процес проектування та реалізації алгоритму обміну конфіденційними даними в корпоративній мережі. Особлива увага приділяється використанню двох важливих алгоритмів шифрування та хешування: Advanced Encryption Standard (AES) та Secure Hash Algorithm-3 (SHA-3), також відомого як Keccak.

Детально описується, як ці алгоритми можуть бути інтегровані в систему захисту для забезпечення конфіденційності, цілісності та аутентифікації даних у корпоративній мережі. Розглядаються їх основні принципи функціонування та технічні характеристики, що роблять їх ефективними засобами забезпечення безпеки.

2.1 Вибір підходу до захисту

Обрано ServerSocketChannel, SocketChannel та DatagramSocket з таких причин:

ServerSocketChannel та SocketChannel:

– Переваги: Вони базуються на Java NIO (Non-blocking I/O), що дозволяє ефективно обробляти багато з'єднань одночасно. ServerSocketChannel використовується для створення серверних сокетів, тоді як SocketChannel використовується для створення з'єднань з клієнтами. Це дозволяє ефективно управляти з'єднаннями та забезпечити швидку передачу даних.

– Недоліки: Вимагають додаткового програмування для ефективної обробки асинхронних подій.

DatagramSocket:

– Переваги: Цей тип сокетів базується на протоколі UDP, що забезпечує швидку передачу даних без побудови надійного з'єднання. Він особливо корисний для додатків, де важлива швидкодія передачі та коли деяка втрата даних допустима.

– Недоліки: Можлива втрата даних через характер UDP.

Обрано ці три типи сокетів через їхні специфічні характеристики, які відповідають вимогам системи обміну конфіденційними даними. ServerSocketChannel та SocketChannel дозволяють ефективно управляти з'єднаннями з клієнтами та забезпечити швидку передачу даних. DatagramSocket забезпечує швидку передачу даних без побудови надійного з'єднання, що може бути корисним для систем, де важлива швидкість передачі. Такий підхід дозволить забезпечити ефективний та надійний обмін конфіденційною інформацією у системі.

Обрано алгоритми AES і SHA-3 (Keccak) з таких причин:

AES (Advanced Encryption Standard):

– Переваги: AES є одним з найбільш поширених симетричних алгоритмів шифрування, який відомий своєю ефективністю та безпекою. Він використовується для шифрування конфіденційних даних у багатьох сучасних системах.

– Недоліки: Хоча AES є дуже надійним алгоритмом, він може бути вразливим до атак, таких як злам методом перебору ключа.

SHA-3 (Keccak):

– Переваги: SHA-3 (Keccak) є одним з найбільш стійких алгоритмів хешування, який відомий своєю стійкістю до криптографічних атак. Він використовується для забезпечення цілісності даних та автентифікації.

– Недоліки: Швидкість обчислення SHA-3 (Keccak) може бути трохи повільнішою, ніж у деяких інших алгоритмів хешування.

Обрано ці два алгоритми через їхню відомість, надійність та широке застосування у сучасних системах шифрування та хешування даних. Вони забезпечують ефективний та надійний захист конфіденційної інформації у системі обміну даними.

Обрані алгоритми є широко використовуваними у сучасних криптографічних застосунках та мають високий рівень довіри у галузі інформаційної безпеки. Їхні стандартизовані та визнані організаціями, такими як

Національний інститут стандартів та технологій (NIST), що свідчить про їхню надійність та ефективність.

AES, як симетричний алгоритм, є особливо ефективним у шифруванні великих обсягів даних та має широку підтримку в різних мережевих та пристрійних середовищах. Використання AES дозволяє забезпечити конфіденційність даних під час їхньої передачі по мережі, що є важливим аспектом в забезпеченні безпеки у корпоративних та комерційних середовищах [29]. Далі буде зображено роботу алгоритм AES на рисунку (2.1)

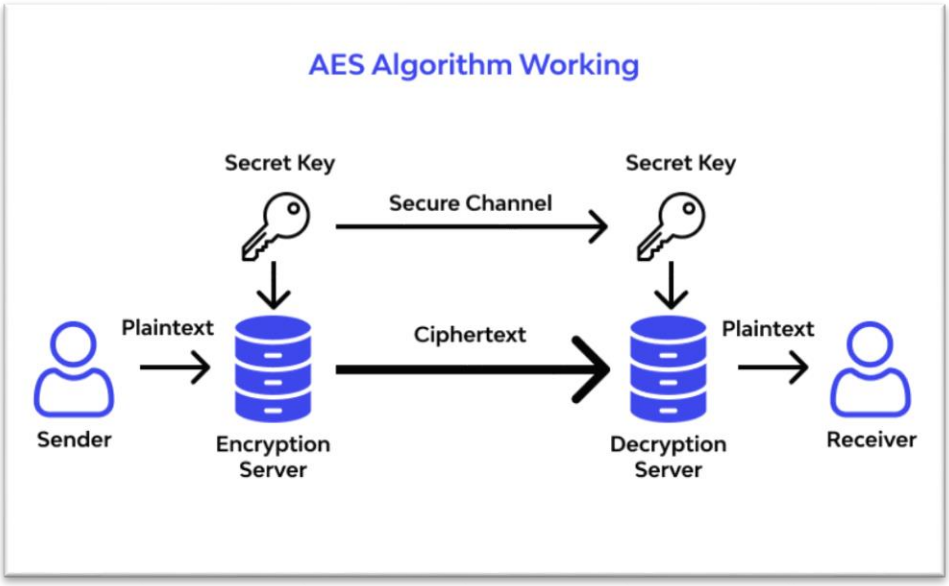


Рисунок 2.1 – Схема роботи алгоритму AES

SHA-3 (Keccak) відомий своєю стійкістю до криптографічних атак і використовується для забезпечення цілісності даних та автентифікації. Використання SHA-3 дозволяє перевірити, чи були дані під час передачі по мережі змінені або підроблені, що допомагає у запобіганні атак на цілісність даних та забезпечує їхню автентичність [30]. Далі буде зображено алгоритм SHA-3 на рисунку (2.2)

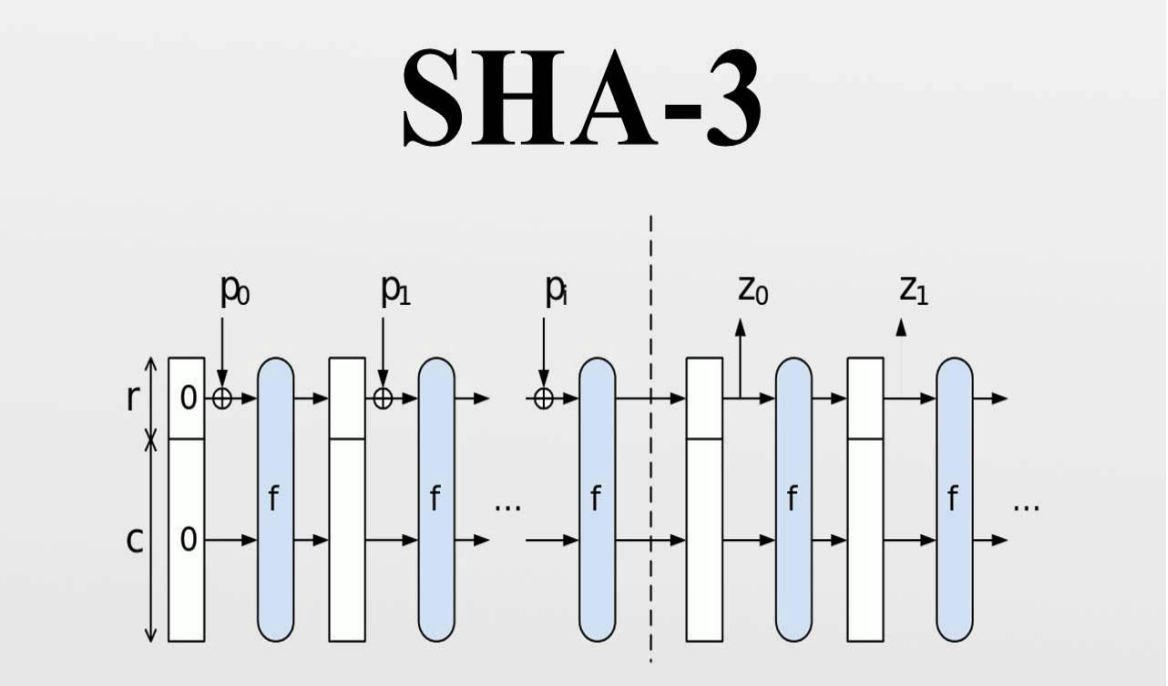


Рисунок 2.2 – Схема алгоритму SHA-3 (Keccak)

Загальна комбінація AES і SHA-3 (Keccak) дозволяє забезпечити високий рівень безпеки та захисту конфіденційної інформації у системі обміну даними. Їхнє використання допомагає уникнути потенційних загроз безпеці, таких як перехоплення та зміна даних, а також забезпечує довіру та конфіденційність в обміні інформацією між сторонами.

2.2 Розробка алгоритму роботи додатку

Було створено схему алгоритму роботи додатку на основі технологій сокетів для корпоративної мережі рисунок (2.3) та на рисунку (2.4).

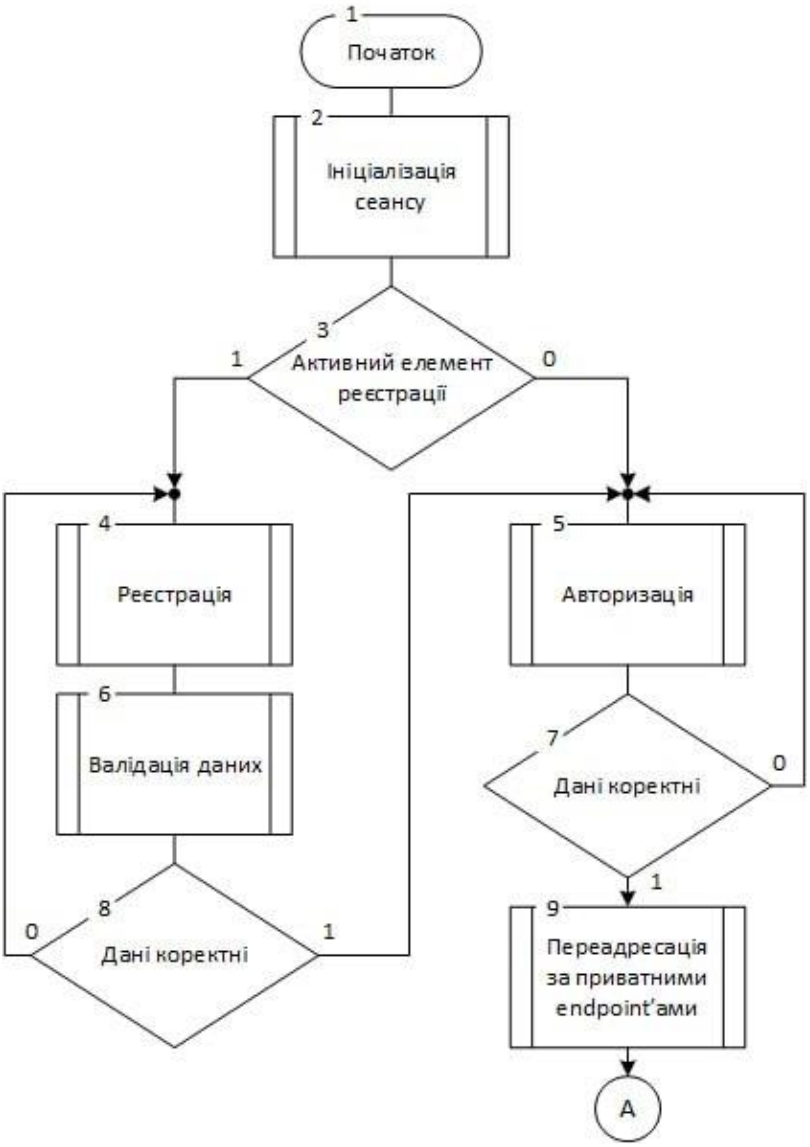


Рисунок 2.3 – Схема алгоритму роботи додатку на основі технології сокетів

Рисунок 2.4 – Схема алгоритму роботи додатку на основі технології сокетів

Створимо опис кожного кроку наведеній схемі. Кожен крок виконує певні дії, що допомагають користувачеві успішно передавати дані через сокети

Крок 1. Початок:

Початок роботи алгоритму.

Крок 2. Ініціалізація зміних:

Ініціалізація необхідних змінних для подальшої роботи додатку.

Крок 3. Активний елемент реєстрації:

Визначається чи активований елемент реєстрації якщо елемент активний то переходить до кроку 4, якщо не активний переходить до кроку 5.

Крок 4. Реєстрація:

Виконується процес реєстрації нового користувача.

Крок 5. Валідація даних:

Відбувається перевірка введених даних на коректність, якщо дані не правильні то повертається до кроку 4 для повторної реєстрації, якщо дані правильні то переходить до авторизації крок 5.

Крок 6. Дані коректні:

Перевірка чи дані коректні якщо дані не коректні то повертається до кроку 4 якщо правильні переходить до авторизації.

Крок 7. Авторизація:

Виконується процес авторизації користувача.

Крок 8. Дані коректні:

Перевірка коректності введених даних для авторизації, якщо дані не правильні то відбувається повторна авторизація крок 5 при правильних даних переходить до кроку 9.

Крок 9. Переадресація за приватними endpoint'ами:

Після успішної авторизації користувач переадресовується на активний режим сервера.

Крок 10. Активний режим сервера:

В залежності від стану сервера вибирається один із двох шляхів, якщо сервер в активному режимі то переходить до кроку 11 якщо ні до кроку 12.

Крок 11. Додавання адреси сервера до пулу:

Адреса сервера додається до пулу доступних серверів..

Крок 12. Список доступних серверів:

Показує список доступних серверів для підключення.

Крок 13. Перехід в режим прослуховування запитів:

Система переходить у режим прослуховування вхідних запитів.

Крок 14. Обрано сервер для підключення:

Перевірка, чи обрано сервер для підключення: якщо так, переходимо до кроку 16, якщо ні повертаємося до списку доступних серверів (крок 12).

Крок 15. Підключення вхідних запитів:

Підключаються вхідні запити для обробки.

Крок 16. Ініціалізація підключення:

Підготовка до підключення до обраного сервера.

Крок 17. Робота з повідомленнями:

Обробка повідомлень, що надходять.

Крок 18. Робота з повідомленнями:

Обробка повідомлень, що надходять.

Крок 19. Сеанс примусово зупинено:

Перевірка, чи сеанс був примусово зупинений: якщо так, переходимо до кінця крок 21, якщо ні, повертаємося до обробки повідомлень крок 17.

Крок 20. Сеанс примусово зупинено:

Перевірка чи сеанс був примусово зупинений, якщо так переходимо до кінця роботи крок 21 якщо ні повертаємося до обробки повідомлень крок 18.

Крок 21. Кінець:

Кінцева точка процесу, де завершуються всі дії.

2.3 Аргументація вибору середовища розробки та мови програмування

Для розробки месенджера в корпоративній мережі на основі технології сокетів було використано середовище IntelliJ IDEA, бібліотека JavaFX, мову програмування JAVA.

IntelliJ IDEA є одним із найкращих виборів для розробки системи обміну конфіденційними даними завдяки своїм розширеним можливостям та функціям. Ця інтегрована середовище розробки (IDE) підтримує широкий спектр мов програмування і фреймворків, включаючи Java, Kotlin, Scala та багато інших, що дозволяє створювати багатофункціональні додатки з використанням найсучасніших технологій. Вбудована підтримка систем контролю версій, таких як Git, SVN та Mercurial, полегшує керування версіями коду та роботу в команді, забезпечуючи ефективне співробітництво та відстеження змін у проекті.

Інструменти для роботи з базами даних, які надає IntelliJ IDEA, дозволяють

легко підключатися до різних СУБД, переглядати структуру бази даних, виконувати SQL-запити та аналізувати результати, що є критично важливим для додатків, які обробляють великі обсяги даних. Крім того, розширені можливості редагування коду, такі як автодоповнення, рефакторинг, перевірка синтаксису та аналіз коду, значно підвищують продуктивність розробників і допомагають знизити ймовірність помилок у коді.

Інтерфейс IntelliJ IDEA є інтуїтивно зрозумілим і налаштовуваним, що забезпечує комфортну роботу та дозволяє розробникам швидко адаптуватися до середовища. Наявність великої кількості плагінів та розширень дає змогу додатково налаштувати IDE під конкретні потреби проекту, що робить його універсальним інструментом для розробки будь-яких додатків. Завдяки цим можливостям, IntelliJ IDEA є потужним та ефективним інструментом для створення безпечних та надійних систем обміну конфіденційними даними.

JavaFX було використано для розробки графічного інтерфейсу. JavaFX надає сучасний та багатий набір інструментів для створення привабливих і функціональних графічних інтерфейсів користувача (GUI). Завдяки своїй модульній архітектурі та підтримці FXML, розробники можуть створювати складні інтерфейси з чітким розділенням логіки та представлення, що полегшує розробку та обслуговування коду.

JavaFX підтримує високоякісну графіку, анімацію та мультимедіа, що дозволяє створювати інтерактивні та динамічні інтерфейси, які покращують користувацький досвід. Можливість використовувати CSS для стилізації компонентів інтерфейсу дозволяє створювати привабливі та консистентні дизайни, які легко налаштовуються та оновлюються. Також важливо відзначити, що JavaFX є кросплатформним рішенням, що забезпечує однакову роботу додатків на різних операційних системах, таких як Windows, macOS та Linux.

JavaFX забезпечує високу продуктивність завдяки використанню апаратного прискорення графіки через API DirectX та OpenGL, що робить його придатним для розробки ресурсоємних додатків. Його архітектура дозволяє створювати масштабовані додатки з високим рівнем продуктивності, що є

важливим для систем обміну конфіденційними даними, які потребують швидкої обробки та відображення великої кількості інформації. Завдяки цим перевагам, JavaFX є оптимальним вибором для створення надійних, безпечних та функціональних графічних інтерфейсів у сучасних додатках.

Вибір мови програмування Java для розробки системи обміну конфіденційними даними є обґрунтованим з кількох ключових причин. По-перше, Java відома своєю високою продуктивністю та стабільністю, що є критично важливим для розробки безпечних та надійних систем. Завдяки компіляції у байт-код, який виконується на віртуальній машині Java (JVM), Java-додатки забезпечують високий рівень продуктивності та можуть працювати на різних платформах без необхідності переробки коду.

По-друге, Java має вбудовані засоби безпеки, що робить її ідеальною для розробки систем, де конфіденційність і захист даних є пріоритетними. Вбудовані механізми, такі як управління пам'яттю, контроль за типами даних і система обробки виключень, забезпечують додатковий рівень безпеки, зменшуючи ризик помилок, які можуть бути використані для атак на систему.

По-третє, Java має велику екосистему бібліотек і фреймворків, які полегшують розробку складних додатків. Наприклад, існують численні бібліотеки для роботи з мережами, шифрування даних, управління базами даних, а також фреймворки для побудови веб-додатків та мікросервісів. Це дозволяє значно скоротити час розробки та забезпечити високий рівень функціональності системи.

Java також підтримується великим співтовариством розробників та має добре документовані ресурси, що полегшує навчання та вирішення проблем, які можуть виникнути під час розробки. Регулярні оновлення та виправлення з боку Oracle та спільноти відкритого програмного забезпечення забезпечують, що мова залишається сучасною та здатною задовольнити нові вимоги безпеки та функціональності.

Крім того, Java є об'єктно-орієнтованою мовою програмування, що сприяє створенню чистого, модульного та легко підтримуваного коду. Це важливо для розробки великомасштабних додатків, де структура та якість коду відіграють

ключову роль у довготривалій підтримці та розвитку системи.

На додаток до цього, Java має чудову підтримку інструментів для розробки та тестування, таких як IntelliJ IDEA, Eclipse, NetBeans, а також безліч інструментів для безперервної інтеграції та розгортання. Це забезпечує ефективний робочий процес для розробників та дозволяє швидко виявляти та виправляти помилки.

Завдяки своїй кросплатформенності, безпеці, продуктивності, широкій екосистемі бібліотек та підтримці розробницького співтовариства, Java є відмінним вибором для розробки системи обміну конфіденційними даними. Вона забезпечує всі необхідні інструменти та функції для створення надійного, масштабованого та безпечного програмного забезпечення, яке відповідає сучасним стандартам та вимогам.

2.4. Висновок до розділу

У цьому розділі розглянуто проектування та реалізацію алгоритму обміну конфіденційними даними в корпоративній мережі з використанням AES та SHA-3 (Keccak). Також було обґрунтовано вибір мови програмування та середовища.

3.РОЗРОБКА ДОДАТКУ ТА СИСТЕМИ ЗАХИСТУ

3.1 Програмна реалізація додатку

Цей код дозволяє шифрувати та дешифрувати текстові дані, використовуючи AES з режимом CBC та PKCS5Padding для заповнення. Він також забезпечує безпеку шляхом використання вектора ініціалізації (IV), що допомагає уникнути повторного використання ключів для однакових текстових даних.

```
public class AES {
   private String content;
   protected static final String CIPHER_TYPE ="AES/CBC/PKCS5Padding";
   public static String encrypt(String content, SecretKeySpec secretKeySpec, IvParameterSpec ivSpec) throws
NoSuchPaddingException, NoSuchAlgorithmException, IllegalBlockSizeException, BadPaddingException,
InvalidAlgorithmParameterException, InvalidKeyException, IOException {
      Cipher encryptCipher = Cipher.getInstance(CIPHER_TYPE);
      encryptCipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, ivSpec);
      byte[] encrypted = encryptCipher.doFinal(content.getBytes());
      return Base64.getEncoder().encodeToString(encrypted);
   }
   public static String decrypt(String content, SecretKeySpec secretKeySpec, IvParameterSpec ivSpec) throws
NoSuchPaddingException, NoSuchAlgorithmException, BadPaddingException, IllegalBlockSizeException,
InvalidAlgorithmParameterException, InvalidKeyException {
      Cipher cipher = Cipher.getInstance(CIPHER_TYPE);
      cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivSpec);
      byte[] plainText = cipher.doFinal(Base64.getDecoder().decode(content));
      return new String(plainText);
   }
}
```

Спочатку імпортуються необхідні криптографічні бібліотеки, після чого визначається клас AES, який містить константу CIPHER_TYPE для вказівки типу шифру. Метод encrypt приймає текст, секретний ключ (SecretKeySpec) та вектор ініціалізації (IvParameterSpec), ініціалізує шифрувальний об'єкт Cipher та шифрує вхідний текст, повертаючи його у вигляді рядка, закодованого в Base64. Метод decrypt приймає зашифрований текст, секретний ключ та вектор ініціалізації, ініціалізує дешифрувальний об'єкт Cipher, декодує вхідний текст з формату Base64 і дешифрує його, повертаючи оригінальний текст. Цей підхід забезпечує безпеку даних, використовуючи AES.

```
private void authorization(ActionEvent event) throws IOException {
   String login = this.login.getText();
   String password = this.password.getText();
   token = HttpServerRequest.loginClient(login, password);
```

```
        if(token.equals("")){
            errorLabel.setText("Невдала авторизація");
            errorLabel.setStyle("-fx-text-fill: red;");
        }else{
            Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
            openModeWindow();
            currentStage.close();
    private void openModeWindow() throws IOException {
        FXMLLoader loader = new FXMLLoader(ChatApp.class.getResource("modeForm.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        Stage newFormStage = new Stage();
        newFormStage.setTitle("Вибір режиму");
        newFormStage.setScene(scene);
        newFormStage.show();
```

Ця частина коду визначає контролер AuthController для керування процесом авторизації та реєстрації користувачів у додатку. Основні функції цього класу включають ініціалізацію форми, обробку зміни ролей, підтвердження авторизації або реєстрації, та відкриття нового вікна після успішної авторизації.

```
public static ClientSocketHandler getClientSocketHandler(SocketChannel socketChannel) {
 for (ClientSocketHandler handler : clients) {
 if (handler.clientChannel == socketChannel) {
 return handler;
 return null;
 public static ClientSocketHandler getClientSocketHandlerByName(String name) {
 for (ClientSocketHandler handler : clients) {
  if (handler.name.equals(name)) {
  return handler;
  return null;
```

Ця частина коду містить два методи для отримання об'єкта ClientSocketHandler з відповідного списку клієнтів. Метод getClientSocketHandler шукає об'єкт за заданим каналом SocketChannel, а метод getClientSocketHandlerByName – за заданим іменем клієнта. Обидва методи перебирають список клієнтів і повертають відповідний об'єкт, якщо такий знаходиться.

```
public static void sendMessageByServer(SocketChannel clientChannel, String message) throws IOException {
   ByteBuffer buffer = StandardCharsets.UTF_8.encode(message);
   clientChannel.write(buffer);
   buffer.clear();
}
```

Ця частина коду містить метод, який використовується для надсилання повідомлення сервером до клієнта. Метод кодує повідомлення у формат ByteBuffer і відправляє його через канал зв'язку clientChannel.

```
public class HttpServerRequest {
   private String inetAddress;
   private final int PORT = 5000;
   protected final static String ADD_ENDPOINT = "http://localhost:8080/pool/addServer";
   protected final static String DELETE_ENDPOINT = "http://localhost:8080/pool/deleteServer";
   protected final static String GET_ENDPOINT = "http://localhost:8080/pool/getServers";
   protected final static String REGISTER_ENDPOINT = "http://localhost:8080/auth/register";
   protected final static String LOGIN_ENDPOINT = "http://localhost:8080/auth/login";
   public HttpServerRequest(String inetAddress) {
      this.inetAddress = inetAddress;
   }
```

Ця частина коду оголошує клас HttpServerRequest і визначає його змінні та константи. Зокрема, змінні включають: inetAddress для зберігання IP-адреси сервера, PORT як константу для порту, що використовується для зв'язку, та кілька констант URL-адрес для різних HTTP-запитів, таких як додавання сервера, видалення сервера, отримання списку серверів, реєстрація та авторизація.

```
public static String encrypt(String message) throws IOException, InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {
   byte[] keyIv = KeccakHash.getHash512(InetAddress.getLocalHost().getHostAddress());
   byte[] key = new byte[24];
   byte[] iv = new byte[16];
   System.arraycopy(keyIv,0,key,0,24);
   System.arraycopy(keyIv,0,iv,0,16);
   SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
   IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
   return KeccakHash.setHash(AES.encrypt(message, secretKeySpec, ivParameterSpec));
```

Метод encrypt використовується для шифрування повідомлення. Він

генерує ключ і вектор ініціалізації (IV) за допомогою хешування IP-адреси. Потім створює відповідні специфікації ключа та IV. Метод шифрує повідомлення за допомогою AES та повертає зашифроване повідомлення разом з його хешем для перевірки цілісності.

```
public void hostServer() {
    HttpClient httpClient = HttpClients.createDefault();
    // Set the request parameters
    String requestBody = "{\"inetAddress\": \"" + inetAddress + "\"}";
    System.out.println(requestBody);
    // Create HttpPost instance
    HttpPost httpPost = new HttpPost(ADD_ENDPOINT);
    httpPost.setHeader("Authorization", "Bearer " + AuthController.token);
    httpPost.setHeader("Content-Type", "application/json");
    try {
        StringEntity requestEntity = new StringEntity(requestBody);
        httpPost.setEntity(requestEntity);
        HttpResponse response = httpClient.execute(httpPost);
```

Метод hostServer використовує HTTP-клієнт для надсилання POST-запиту на сервер, щоб додати новий сервер до пулу серверів. Запит містить IP-адресу сервера у форматі JSON, а також включає заголовки авторизації та типу контенту.

```
public void unhostServer() {
    HttpClient httpClient = HttpClients.createDefault();
    // Set the request parameters
    String requestBody = "{\"inetAddress\": \"" + inetAddress + "\"}";
    // Create HttpPost instance
    HttpPatch httpPatch = new HttpPatch(DELETE_ENDPOINT);
    httpPatch.setHeader("Content-Type", "application/json");
        StringEntity requestEntity = new StringEntity(requestBody);
        httpPatch.setEntity(requestEntity);
        HttpResponse response = httpClient.execute(httpPatch);
        if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
            System.out.println("Request successful");
            System.out.println("Request failed with status code: " + response.getStatusLine().getStatusCode());
        catch (IOException e)
        e.printStackTrace();
```

Метод unhostServer використовує HTTP-клієнт для надсилання PATCH-запиту на сервер, щоб видалити сервер з пулу серверів. Запит містить IP-адресу

сервера у форматі JSON, а також включає заголовок типу контенту. Метод виконує запит і виводить результат запиту на консоль.

```
public static HttpResponse registerClient(String login, String password) throws IOException {
    HttpClient httpClient = HttpClients.createDefault();
    HttpPost httpPost = new HttpPost(REGISTER_ENDPOINT);
    httpPost.setHeader("Content-Type", "application/json");
    String requestBody = "{\"username\": \"" + login + "\", \"password\": \"" + password + "\"}";
    System.out.println(requestBody);
    StringEntity requestEntity = new StringEntity(requestBody);
    httpPost.setEntity(requestEntity);
    return httpClient.execute(httpPost);
```

Метод registerClient використовує HTTP-клієнт для надсилання POST-запиту на сервер, щоб зареєструвати нового користувача. Запит містить ім'я користувача та пароль у форматі JSON, а також включає заголовок типу контенту. Метод виконує запит і повертає відповідь сервера.

```
public static ArrayList<String> getServers(){
    HttpClient httpClient = HttpClients.createDefault();
    HttpGet httpGet = new HttpGet(GET_ENDPOINT);
    System.out.println(AuthController.token);
    httpGet.setHeader("Authorization", "Bearer " + AuthController.token);
    httpGet.setHeader("Content-Type", "application/json");
     try { HttpResponse response = httpClient.execute(httpGet);
      if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
         System.out.println("Request successful");
         HttpEntity responseEntity = response.getEntity();
         String responseBody = EntityUtils.toString(responseEntity);
         JSONArray jsonArray = new JSONArray(responseBody);
         System.out.println(jsonArray);
         ArrayList<String> list = new ArrayList<>();
         for(int i = 0; i < jsonArray.length(); i++) {
            list.add(jsonArray.getJSONObject(i).getString("inetAddress"));
            System.out.println(jsonArray.getJSONObject(i).getString("inetAddress"));
```

Метод getServers використовує HTTP-клієнт для надсилання GET-запиту на сервер, щоб отримати список серверів. Запит включає заголовки авторизації та типу контенту. Метод зчитує відповідь у форматі JSON і повертає список IP-адрес серверів.

```
public static String loginClient(String login, String password) throws IOException {
    HttpClient httpClient = HttpClients.createDefault();
    HttpPost httpPost = new HttpPost(LOGIN_ENDPOINT);
    httpPost.setHeader("Content-Type", "application/json");
    String requestBody = "{\"username\": \"" + login + "\", \"password\": \"" + password + "\"}";
    StringEntity requestEntity = new StringEntity(requestBody);
    httpPost.setEntity(requestEntity);
    HttpResponse response = httpClient.execute(httpPost);
    if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
        String responseBody = EntityUtils.toString(response.getEntity());
        JSONObject jsonObject = new JSONObject(responseBody);
        String token = jsonObject.getString("jwt");
        System.out.println("Login request successful");
        System.out.println(responseBody);
        return token;
    } else {
        System.out.println("Register request failed with status code: " + response.getStatusLine().getStatusCode());
        return null;
```

Метод loginClient використовує HTTP-клієнт для надсилання POST-запиту на сервер, щоб авторизувати користувача. Запит містить ім'я користувача та пароль у форматі JSON, а також включає заголовок типу контенту. Метод зчитує відповідь у форматі JSON, витягує токен авторизації та повертає його.

```
public static String getTime() {
    LocalTime currentTime = LocalTime.now();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");
    return currentTime.format(formatter);
```

Ця частина коду містить метод getTime, який повертає поточний час у форматі HH:mm. Метод отримує поточний час за допомогою класу LocalTime і форматуванням перетворює його у потрібний формат.

```
public static String decrypt(String bufferContent) throws UnknownHostException,
InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException,
NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {
    byte[] keyIv = KeccakHash.getHash512(keyAddress);
    byte[] key = new byte[24];
    byte[] iv = new byte[16];
    System.arraycopy(keyIv,0,key,0,24);
    System.arraycopy(keyIv,0,iv,0,16);
    SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
```

```
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
        String checkedBuffer = KeccakHash.checkHash(bufferContent);
        if(checkedBuffer!=null){
            return AES.decrypt(checkedBuffer, secretKeySpec, ivParameterSpec);
        }else{
            return "Помилка передачі";
```

Метод decrypt використовується для розшифровки отриманого повідомлення. Спочатку він генерує ключ і вектор ініціалізації (IV) за допомогою хешування IP-адреси. Потім створює відповідні специфікації ключа та IV. Метод перевіряє хеш отриманого повідомлення і, якщо перевірка успішна, розшифровує його за допомогою AES. Якщо перевірка не вдається, повертається повідомлення про помилку.

```
    private void sendMessageByClient() throws IOException, InvalidAlgorithmParameterException,
NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException,
InvalidKeyException {
        String message = messageInput.getText();
        String last = encrypt(message);
        sendMessage(serverSocketChannel, last);
        messageInput.setText("");
    }
```

Метод sendMessageByClient використовується для відправки повідомлення клієнтом. Спочатку метод отримує текст повідомлення з поля введення, потім шифрує його за допомогою методу encrypt. Після цього він відправляє зашифроване повідомлення на сервер через канал сокета serverSocketChannel. Після відправки поле введення очищується.

```
    public class KeccakHash {
        public static String getHash256(String content){
            Keccak.Digest256 digest256 = new Keccak.Digest256();
            byte[] hashbytes = digest256.digest(content.getBytes());
            String sha3Hex = new String(Hex.encode(hashbytes));
            return sha3Hex;
            public static byte[] getHash512(String content){
            Keccak.Digest512 digest512 = new Keccak.Digest512();
            return digest512.digest(content.getBytes(StandardCharsets.UTF_8));
            public static String setHash(String content){
            return content+getHash256(content);
```

```
public static String checkHash(String content){
String main = content.substring(0, content.length()-64);
String hash = content.substring(content.length()-64);
if(getHash256(main).equals(hash)){
   return main;
}else{
   return null;
```

Метод encrypt, який використовує алгоритм AES для шифрування повідомлення з використанням SHA-3 (Keccak) для хешування.

```
private void sendMessageByHost() throws IOException, InvalidAlgorithmParameterException,
NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException,
InvalidKeyException {
   String message = messageInput.getText();
   Command.SERVERDEFAULTMESSAGE(message);
   messageList.getItems().add(getTime()+" | Server: " + message);
   messageInput.setText("");
```

Метод sendMessageByHost використовується для відправки повідомлення від сервера. Він отримує текст повідомлення з поля введення, відправляє це повідомлення за допомогою команди Command.SERVERDEFAULTMESSAGE, а потім додає повідомлення до списку повідомлень разом з поточним часом. Після цього поле введення очищується.

```
public void unhostServer() {
 HttpClient httpClient = HttpClients.createDefault();
 String requestBody = "{\"inetAddress\": \"" + inetAddress + "\"}";
 HttpPatch httpPatch = new HttpPatch(DELETE_ENDPOINT);
 httpPatch.setHeader("Content-Type", "application/json");
 try {
    StringEntity requestEntity = new StringEntity(requestBody);
    httpPatch.setEntity(requestEntity);
    HttpResponse response = httpClient.execute(httpPatch);
    if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
       System.out.println("Request successful");
    } else {
       System.out.println("Request failed with status code: " + response.getStatusLine().getStatusCode());
 } catch (IOException e) {
    e.printStackTrace();
```

Ця частина коду містить метод unhostServer, який використовує HTTP-

клієнт для надсилання PATCH-запиту на сервер, щоб видалити сервер з пулу серверів. Запит містить IP-адресу сервера у форматі JSON. Метод встановлює тип контенту, виконує запит і виводить результат запиту на консоль.

Також було розроблено інтерфейс програми, на рисунку (3.1) зображено вікно авторизації та реєстрації користувача.

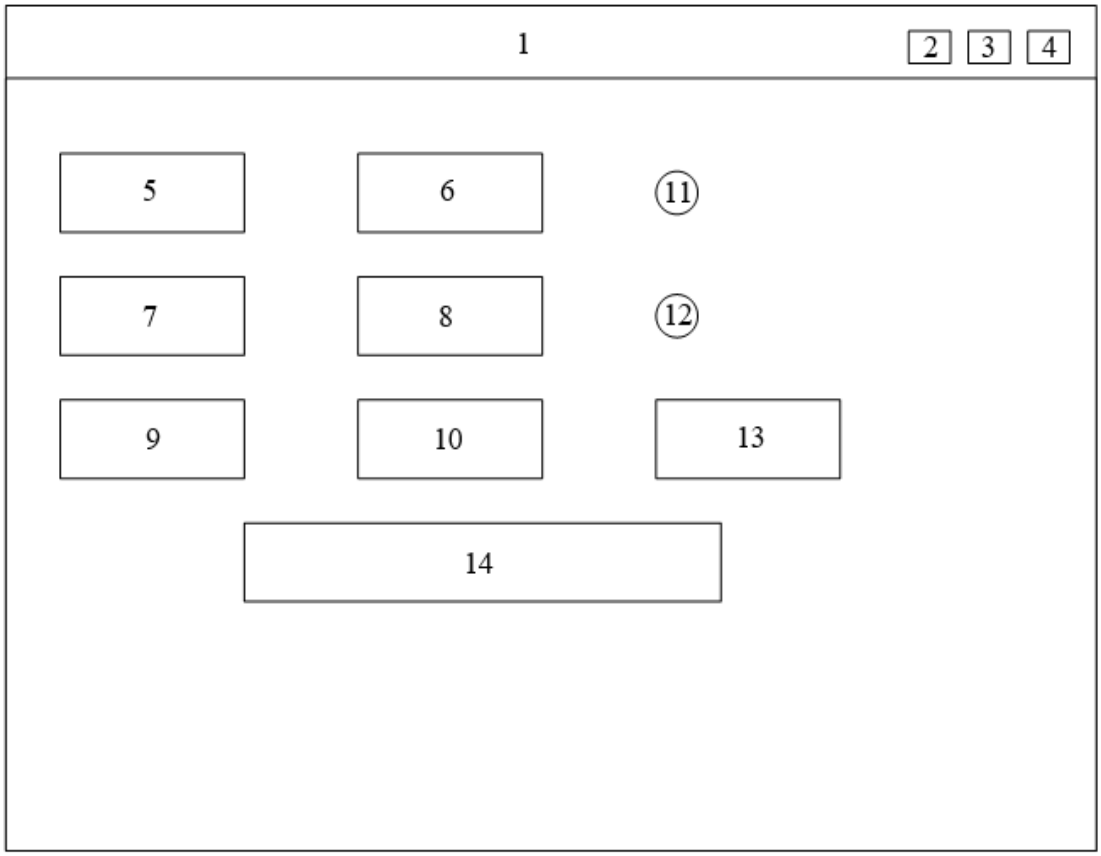


Рисунок 3.1 – Графічна схема інтерфейсу головного вікна авторизації та реєстрації

Елементи вікна авторизації та реєстрації:

1. Заголовок вікна.
2. Згорнути вікно – кнопка системного керування.
3. Розгорнути вікно – кнопка системного керування.
4. Закрити вікно – кнопка системного керування.
5. Текстове поле з назвою “Логін”.
6. Робоче поле для запису логіну.
7. Текстове поле з назвою “Пароль”.

8. Робоче поле для запису паролю.

9. Текстове поле з назвою “Повторити пароль”.

10. Робоче поле для запису повторного паролю.

11. Радіо кнопка – Авторизація.

12. Радіо кнопка – Реєстрації.

13. Кнопка підтвердженя.

14. Текстове поле повідомлення не правильно введених даних.

Далі зображено схему інтерфейсу для передавання даних між користувачами і сервером рисунок (3.2).

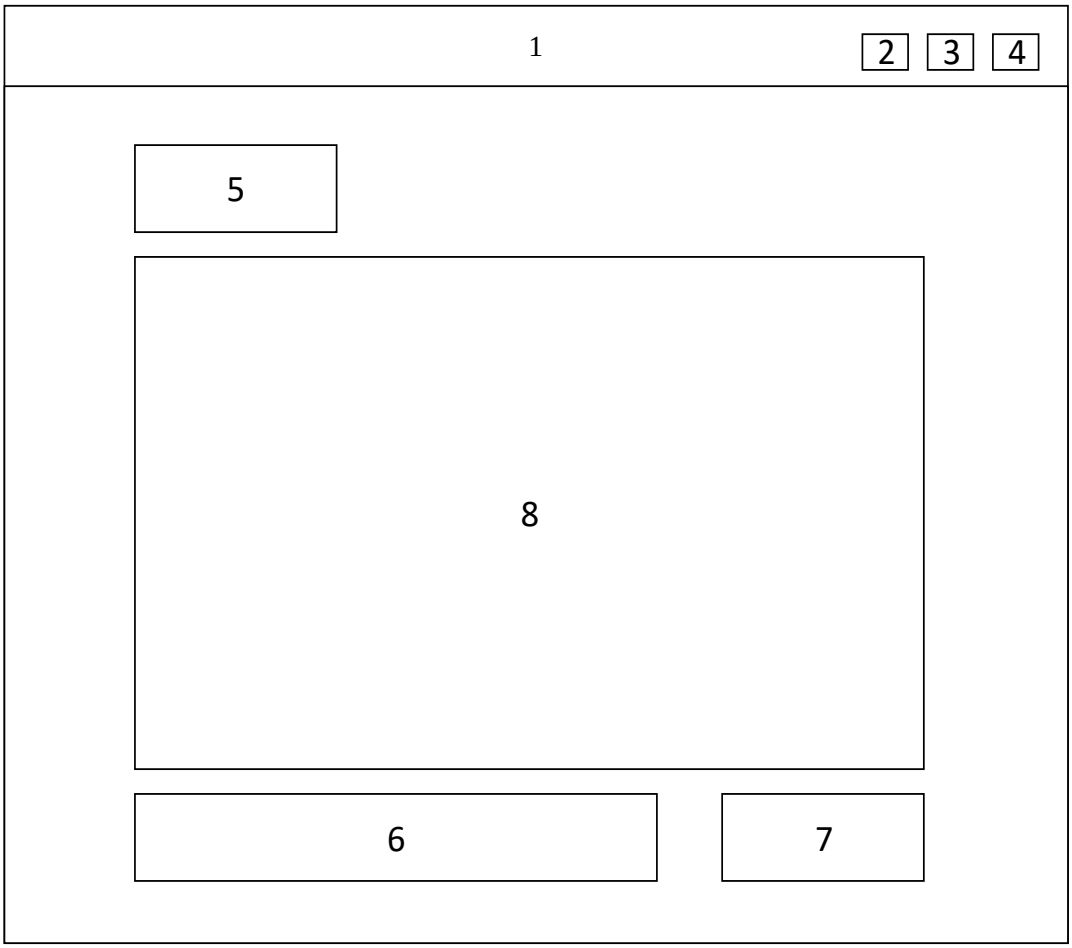


Рисунок 3.2 – Графічна схема інтерфейсу головного вікна передавання даних

Елементи вікна передавання даних:

1. Заголовок вікна.

2. Згорнути вікно – кнопка системного керування.

3. Розгорнути вікно – кнопка системного керування.

4. Закрити вікно – кнопка системного керування.

5. Текстове поле статусу.

6. Робоче поле для введення тексту.

7. Кнопка – Надіслати.

8. Робоче поле виведення тексту.

Зображено схему інтерфейсу у якому вибирається режим роботи сервера або клієнта рисунок (3.3).

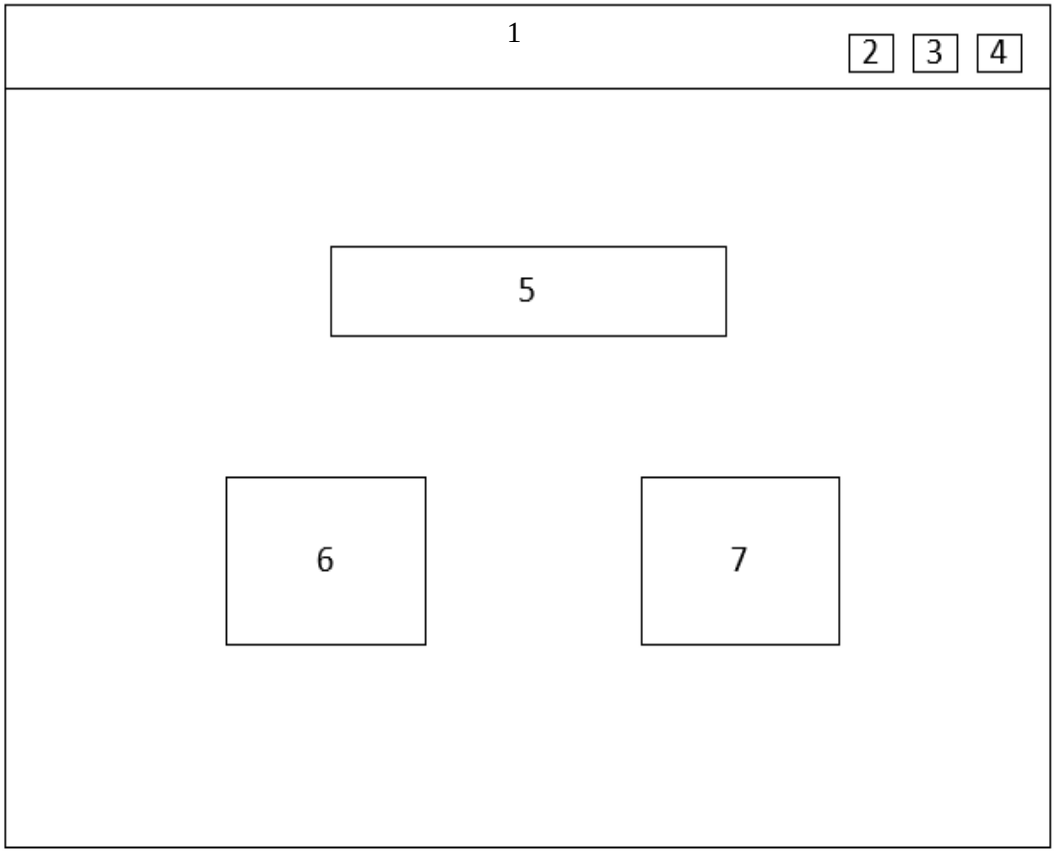


Рисунок 3.3 – Графічна схема інтерфейсу вікна вибору статусу

Елементи вікна передавання даних:

1. Заголовок вікна.

2. Згорнути вікно – кнопка системного керування.

3. Розгорнути вікно – кнопка системного керування.

4. Закрити вікно – кнопка системного керування.

5. Текстове поле вибору режиму роботи.

6. Кнопка вибору режиму сервера.

7. Кнопка вибору режиму клієнта.

3.2. Інструкція користувача

На основі розробленого додатку було створено інструкцію для роботи з месенджером створеного на технології сокетів. Щоб працювати з додатком потрібно його завантажити далі на завантажену програму рисунок (3.4) потрібно натиснути два рази.

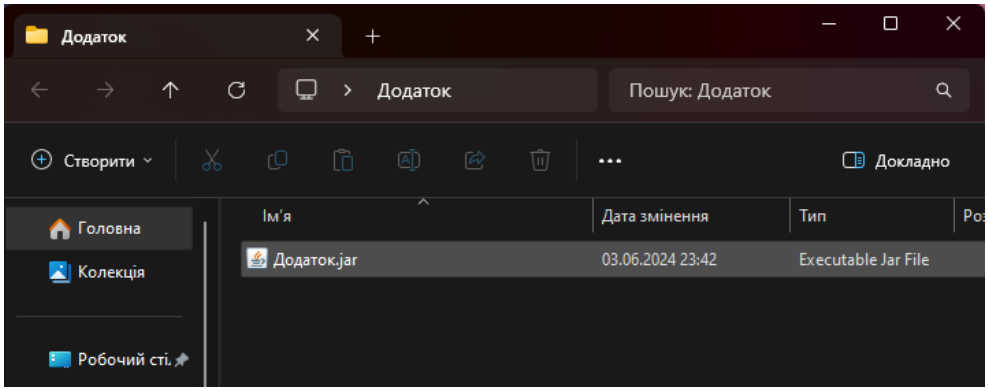


Рисунок 3.4 – Завантажений додаток

Після відкриття додатку відкривається вікно авторизації та реєстрації на рисунку (3.5) вибрано радіо кнопку авторизації тоді пишете логін і пароль і заходите в свій акаунт.

Рисунок 3.5 – Головне вікно авторизації та реєстрації

Після натискання радіо кнопки реєстрації, можна створити акаунт і буде активне поле повторного паролю для його підтвердження рисунок (3.6).

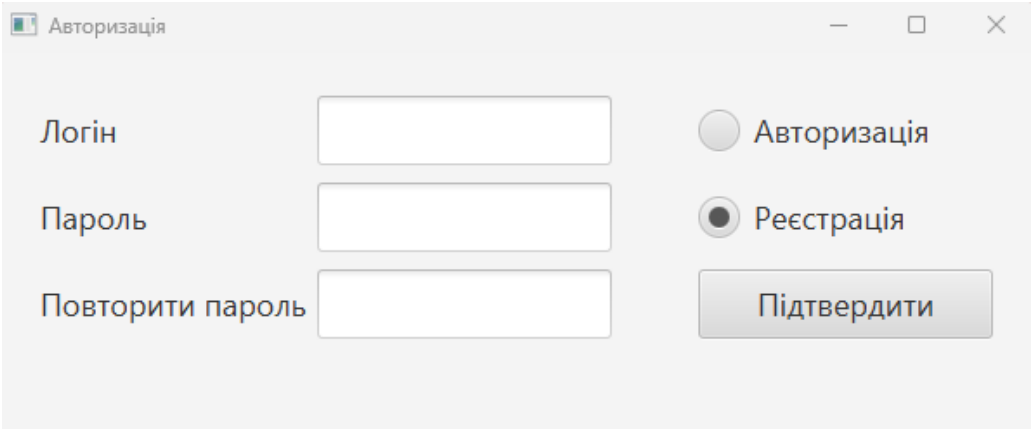


Рисунок 3.6 – Головне вікно авторизації та реєстрації

Після успішної реєстрації та авторизації акаунта відкриється нове вікно, в якому ви можете вибрати в ролі кого вам бути клієнта або сервера рисунок (3.7).

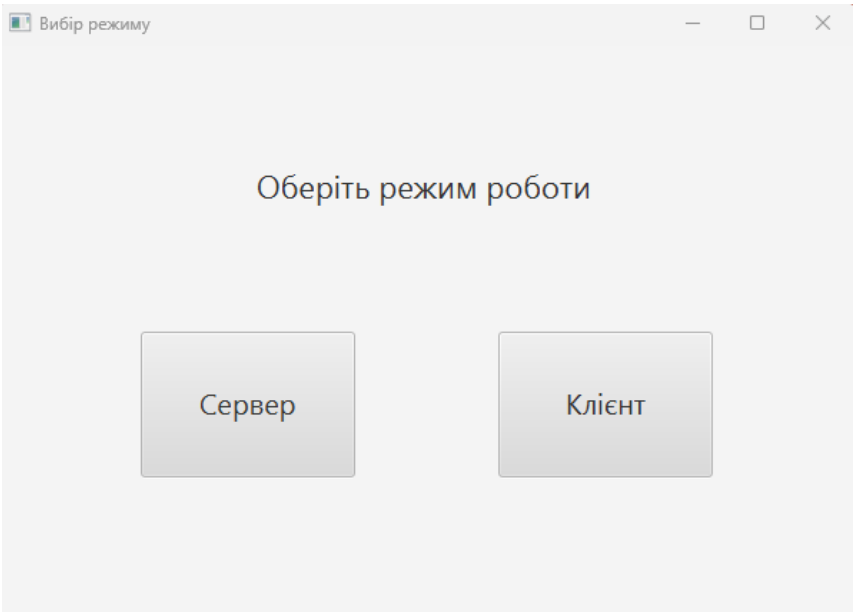


Рисунок 3.7 – Вікно вибору режиму роботи

Після вибору режиму роботи клієнта відкриється вікно з доступними серверами, натиснувши два рази на один із серверів буде установлено підключення рисунок (3.8). Коли з’єднання було успішним ви можете передавати дані іншому клієнту.

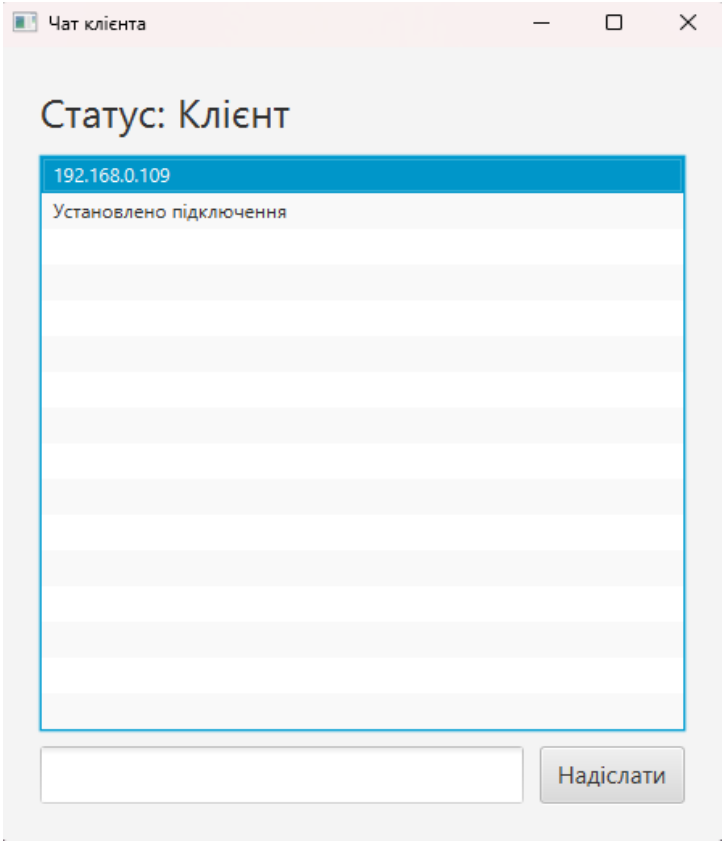


Рисунок 3.8 – Вікно чату клієнта

Після вибору режиму роботи сервера відкриється чат серверу через який ви зможете спілкуватись з клієнтами і клієнти зможуть підключатись до вас (3.9).

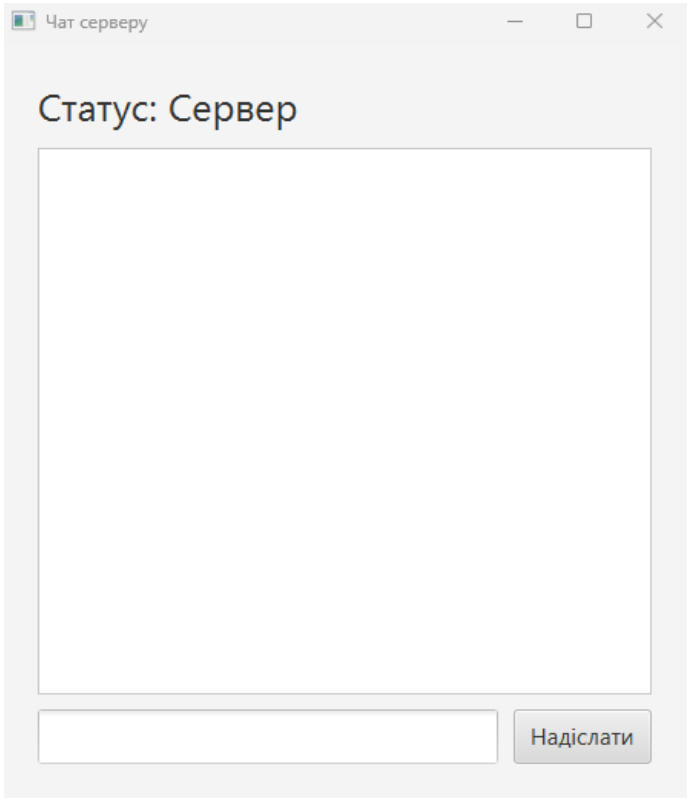


Рисунок 3.9 – Вікно чату сервера

3.3. Тестування програмного додатку

Для того, щоб перевірити правильність роботи додатку було проведено тестування додатку на основі технології сокетів на рисунку (3.10) відображено реєстрацію де реєстрація не відбувається при повторному неправильному написані паролю.

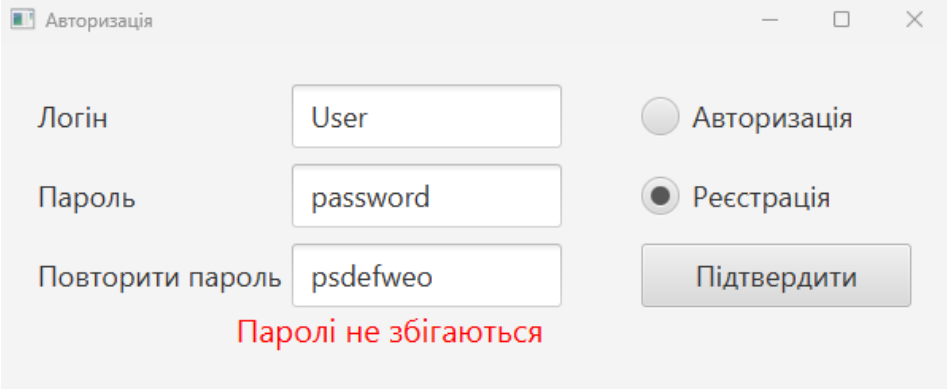


Рисунок 3.10 – Вікно реєстрації

Далі було введено дані не існуючого акаунту і авторизація виявилась не вдалою, тому додаток працює коректно і користувачі які не реєстровані не зможуть його використовувати рисунок (3.11).

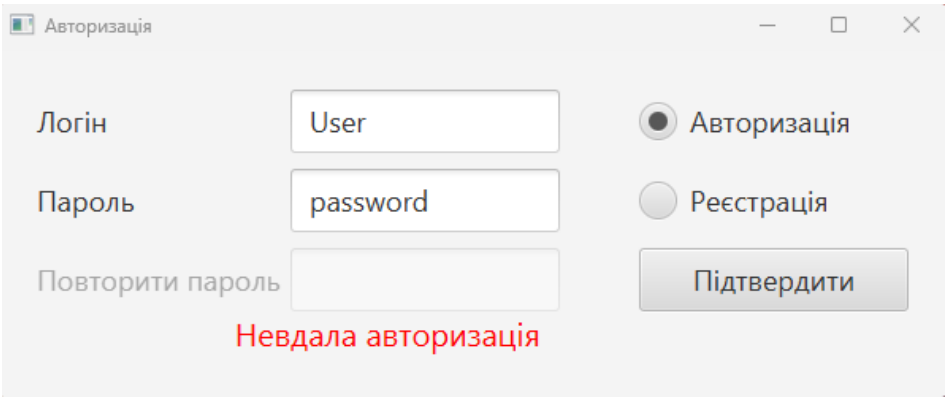


Рисунок 3.11 – Невдала авторизація користувача

Під час реєстрації було ведено логін, пароль і повторний пароль і реєстрація була успішна рисунок (3.12) тому після вдалої реєстрації і авторизації можливе використання додатку.

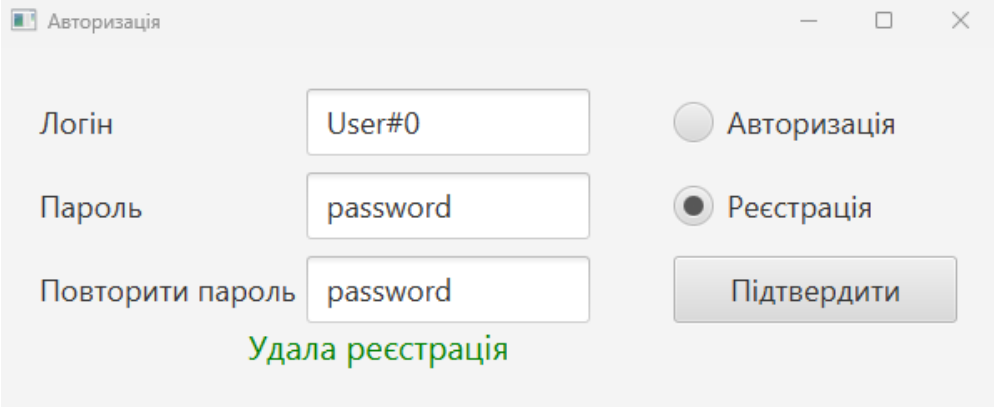


Рисунок 3.12 – Удала реєстрація користувача

Після вдалої реєстрації і вибравши режим роботи клієнта відкриється чат клієнта в якому перевіримо чи можна підключитись до сервера та обмінюватись даними нажавши на сервер два рази з’явиться текст про встановлення підключення рисунок (3.13).

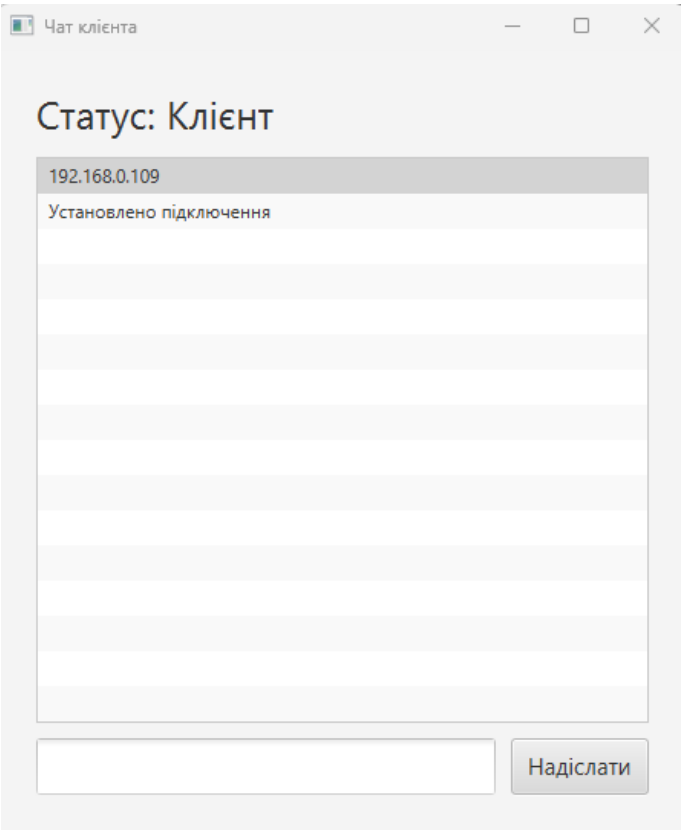


Рисунок 3.13 – Установлено підключення до серверу

Коли було установлено підключення перевірим чи будуть відправлятись повідомлення від сторони клієнта ім’я якого User#0 до серверу рисунок (3.14).

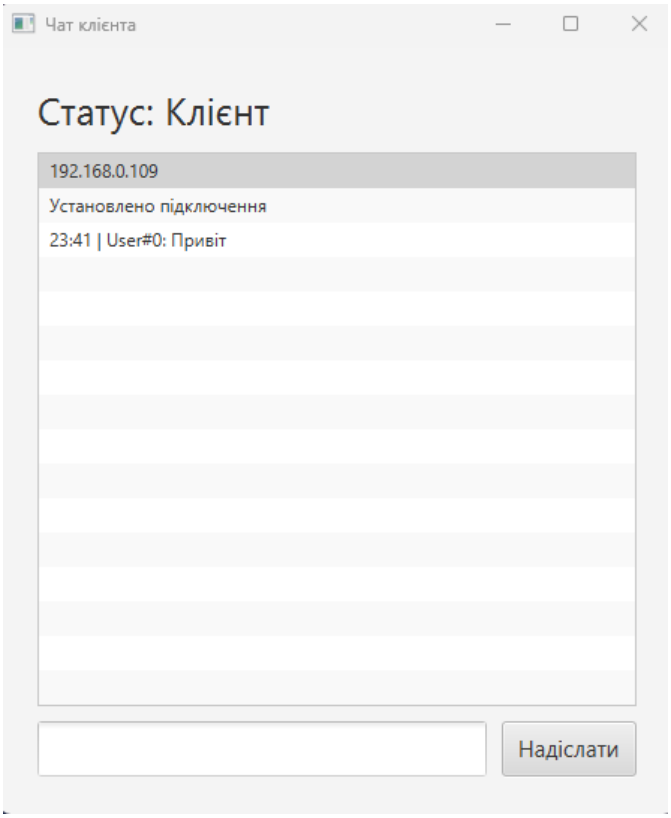


Рисунок 3.14 – Відправлено повідомлення до серверу

Далі проведено перевірку з сторони серверу, на сервер було відправлено коректно повідмлення і з сервера також було відправлено повідомлення рисунок (3.15).

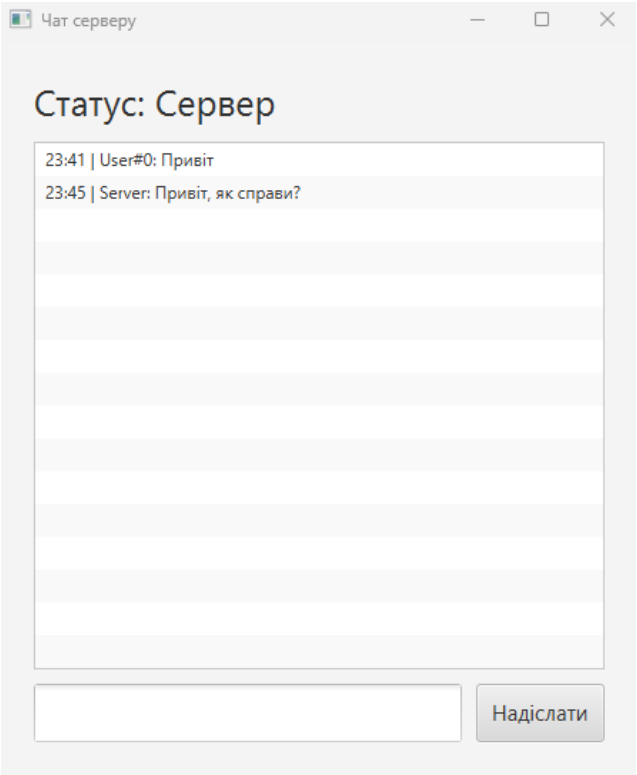


Рисунок 3.15 – Отримане повідомлення і відправка повідомлення до клієнта

Проаналізувавши всі ці дії, що були виконані у процесі тестування, можна зробити висновок що додаток працює правильно, тобто програмний продукт реалізовано задовільнивши усі вимоги поставленого завдання.

3.4. Висновок до розділу

В даному розділі здійснено процес розробки додатку для обміну конфіденційними даними в корпоративній мережі на основі технології сокетів, виконано програмування сервера. Створено детальну інструкцію роботи з месенджером для користувача. Та проведено тестування виготовленого програмного продукту.

ВИСНОВКИ

У даній роботі була проведена розробка захищеного додатку для обміну конфіденційними даними в корпоративній мережі на основі технології сокетів. Для цього були досліджені різні методи шифрування та захисту даних від несанкціонованого доступу під час їх передачі мережею. Були проаналізовані сучасні криптографічні алгоритми, зокрема AES для шифрування даних та SHA-3 для хешування інформації, що забезпечує цілісність та автентичність переданих даних.

Було також вивчено та впроваджено механізми аутентифікації та авторизації користувачів за допомогою фреймворку Spring Security, що дозволяє надійно ідентифікувати користувачів та контролювати доступ до інформації.

Розроблений додаток був протестований для перевірки його працездатності та відповідності вимогам безпеки. Тестування включало перевірку правильності, функцій аутентифікації та авторизації користувачів, а також забезпечення захисту від несанкціонованого доступу.

Було також здійснено огляд інтерфейсу додатку, який дозволяє користувачам зручно та інтуїтивно здійснювати обмін конфіденційними даними в корпоративній мережі.

Отже, можна зробити висновок, що розроблений додаток для обміну конфіденційними даними є ефективним рішенням для забезпечення захисту інформації в корпоративній мережі. Використання технології сокетів у поєднанні з алгоритмами AES та SHA-3, а також фреймворком Spring Security, забезпечує високий рівень безпеки, конфіденційності та цілісності переданих даних, що є критично важливим для захисту корпоративної інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ролик А. И. Моделювання управління потоками даних в корпоративних IP-мережах. Адаптивні системи автоматичного управління. 2011. С. 93–102. URL: https://doi.org/10.20535/1560-8956.18.2011.33509 (дата звернення: 12.05.2024).

2. The CIA triad: Definition, components and examples URL: The CIA triad: Definition, components and examples | CSO Online (дата звернення: 13.05.2024).

3. Ristic I. Bulletproof SSL and TLS URL: https://books.google.com.ua/books?id=fQOLBAAAQBAJ&printsec=frontcover&hl=ru#v=onepage&q&f=false (дата звернення: 13.05.2024).

4. Information security of employees: important rules to consider URL: https://techexpert.ua/employees-cyber-security-at-work/ (дата звернення: 15.05.2023).

5. Encrypted network traffic analysis of secure instant messaging application: a case study of signal messenger app. MDPI. URL: https://www.mdpi.com/2076-3417/11/17/7789 (дата звернення: 13.05.2024).

6. Privacy in instant messaging applications. Masaar. URL: https://masaar.net/en/privacy-in-instant-messaging-applications/ (дата звернення: 15.05.2024)

7. Красношапка Д., Золотько К. простий текстовий формат реляційної бази даних для розробки, опису та обміну даними через мережу. information technology and society. 2023. с. 34–40. url: https://doi.org/10.32689/maup.it.2022.4.5 (дата звернення: 15.05.2024).

8. Craig H. TCP/IP. Network Administration URL: https://www.gob.mx/cms/uploads/attachment/file/84434/02.TCPIP_Network_Administration.pdf (дата звернення: 15.05.2024)

9. TCP/IP - what it is and how it works URL: https://highload.today/uk/tcp-ip-shho-tse-take-i-yak-pratsyuye/ (дата звернення: 15.05.2024)

10. The Essential Guide to S/MIME for Secure Email Communications URL: https://www.ssl.com/article/the-essential-guide-to-s-mime-for-secure-email-communications/ (дата звернення: 15.05.2024)

11. How does a VPN work? URL: https://nordvpn.com/uk/what-is-a-vpn/ (дата звернення: 15.05.2024)

12. Батрак Є.В. РОЗРОБКА ТА РЕАЛІЗАЦІЯ МЕРЕЖНИХ ПРОТОКОЛІВ НАВЧАЛЬНИЙ ПОСІБНИК URL: http://surl.li/ufzfb (дата звернення: 20.05.2024)

13. Document Management System URL:https://evolpe.com.ua/document-management-system/ (дата звернення: 20.05.2024)

14. File sharing platforms URL: https://artismedia.by/blog/10-luchshih-programm-dlia-obmena-failami-na-pk-2024/ (дата звернення: 20.05.2024)

15. ser Datagram Protocol–UDP. Industrial Communication Systems. 2018. с. 839–848. URL: https://doi.org/10.1201/9781315218434-71 (дата звернення: 20.05.2024).

16. Які види шифрування використовують месенджери URL: https://speka.media/naskilki-vasi-cati-v-mesendzerax-v-bezpeci-p0ewlv (дата звернення: 20.05.2024)

17. Basics of sockets URL: https://www.ap-impulse.com/socket-prikladi-step-125/ (дата звернення: 15.05.2024)

18. An overview of HTTP URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (дата звернення: 21.05.2024)

19. What is the IoT? URL: https://www.ibm.com/topics/internet-of-things (дата звернення: 21.05.2024)

20. What is SSL/TLS and why is it important? URL: https://www.ibm.com/topics/internet-of-things (дата звернення: 21.05.2024)

21. Бурячок В. Л. Технології забезпечення безпеки мережевої інфраструктури. 2019. с. 218 http://surl.li/ujmvj (дата звернення: 21.05.2024)

22. OpenSSL URL: https://www.wikiwand.com/ru/OpenSSL (дата звернення: 21.05.2024)

23. Secure File Transfers: Best Practices, Protocols And Tools URL: https://www.advsyscon.com/blog/secure-file-transfers/ (дата звернення: 22.05.2024)

24. Yapri J., Hananto R. Leak in OpenSSL. Journal of Applied Information, Communication and Technology. 2021. с. 1–6. URL: https://doi.org/10.33555/ejaict.v7i1.70 (дата звернення: 22.05.2024).

25. Fox D. Secure Hash Algorithm SHA-3. Datenschutz und Datensicherheit - DuD. 2013. Vol. 37, no. 2. P. 104. URL: https://doi.org/10.1007/s11623-013-0027-z (date of access: 22.06.2024).

26. SHA-3 (Secure Hash Algorithm Version 3) URL: https://bitcoinwiki.org/wiki/SHA-3 (дата звернення: 22.05.2024)

27. Heron S. Advanced Encryption Standard (AES). Network Security. 2009. с. 8–12. URL: https://doi.org/10.1016/s1353-4858(10)70006-4 (дата звернення: 23.05.2024).

28. AES Encryption Explained (Advanced Encryption Standard) URL: https://blog.kraden.com/aes-256-encryption (дата звернення: 23.05.2024)

29. Advanced Encryption Standard (AES) URL: https://ebin.pub/advanced-encryption-standard-aes.html (дата звернення: 23.05.2024)

30. Cryptography and Network Security: Principles and Practice URL: https://www.cs.vsb.cz/ochodkova/courses/kpb/cryptography-and-network-security_-principles-and-practice-7th-global-edition.pdf (дата звернення: 23.05.2024)

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління
інформаційною
безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:

«Розробка захищеного додатку обміну конфіденційними даними в корпоративній мережі на основі технології сокетів»

08-72.БДР.011.00.080.ТЗ

Керівник бакалаврської дипломної роботи
к.т.н., доц., каф., МБІС:
_____ Карпінець В. В.
"_____" _____

Вінниця – 2024 р.

1. Найменування та область застосування

Додаток на основі технологій сокетів з використанням алгоритму AES, SHA-3 і фреймоворку Spring Security. Область застосування: використання як захищеного месенджера для корпоративної мережі.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 80 від 11.03.2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка ефективної системи захисту додатку на технологіях сокетів від несанкціонованого доступу.

3.2 Призначення: розроблений програмний засіб виконує захист несанкціонованого доступу до інформації.

4. Джерела розробки

4.1. Quinn B. Windows Sockets Network Programming. Pearson Education, Limited, 2020. 800 с.

4.2. Harold E. R. Java Network Programming. 2nd ed. O'Reilly, 2000. 760 с.

4.3. Stallings W. Cryptography and Network Security: Principles and Practice. Pearson Education, Limited, 2010. 744 с.

4.4. Ахрамович В. М. Ідентифікація й аутентифікація, керування доступом // Сучасний захист інформації. 2016. с. 47-51.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний модуль повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес виявлення зловмисних посилань.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних

повідомлень;

5.2.2 Програмний засіб повинен виконувати свої функції.

5.2.3 Програмний модуль повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.3

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів..

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	22.03.2024	23.03.2024
2.	Аналіз предметної області обраної теми	24.03.2024	21.04.2024
3.	Розробка алгоритму роботи	10.05.2024	18.05.2024
4.	Написання бакалаврської роботи на основі розробленої теми	18.05.2024	25.05.2024
5.	Передзахист бакалаврської дипломної роботи	26.05.2024	27.05.2024
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2024	30.05.2024
7.	Захист бакалаврської дипломної роботи	12.06.2024	17.06.2024

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

- ПЗ до бакалаврської дипломної роботи;
- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента.

Технічне завдання до виконання прийняв_____Ільчук Р. В.

Додаток Б. Лістинг програми

```
package org.example.chatapplication;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
public class AES {
   private String content;
   protected static final String CIPHER_TYPE ="AES/CBC/PKCS5Padding";
   public static String encrypt(String content, SecretKeySpec secretKeySpec, IvParameterSpec ivSpec) throws
NoSuchPaddingException, NoSuchAlgorithmException, IllegalBlockSizeException, BadPaddingException,
InvalidAlgorithmParameterException, InvalidKeyException, IOException {
      Cipher encryptCipher = Cipher.getInstance(CIPHER_TYPE);
      encryptCipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, ivSpec);
      byte[] encrypted = encryptCipher.doFinal(content.getBytes());
      return Base64.getEncoder().encodeToString(encrypted);
   }
   public static String decrypt(String content, SecretKeySpec secretKeySpec, IvParameterSpec ivSpec) throws
NoSuchPaddingException, NoSuchAlgorithmException, BadPaddingException, IllegalBlockSizeException,
InvalidAlgorithmParameterException, InvalidKeyException {
      Cipher cipher = Cipher.getInstance(CIPHER_TYPE);
      cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivSpec);
      byte[] plainText = cipher.doFinal(Base64.getDecoder().decode(content));
      return new String(plainText);
   }
}
package org.example.chatapplication;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;
import org.apache.http.HttpResponse;
import java.io.IOException;
public class AuthController {
   public static String token;
   @FXML
   private TextField login;
   @FXML
   private TextField password;
   @FXML
   private TextField confirmPassword;

   @FXML
   private RadioButton authRadio;

   @FXML
```

```
private ToggleGroup auth;
@FXML
private Label confirmLabel;
@FXML
private Label errorLabel;
@FXML
public void initialize() {
   if(authRadio.isSelected()) {
      confirmPassword.setDisable(true);
      confirmLabel.setDisable(true);
   }else{
      confirmPassword.setDisable(false);
      confirmLabel.setDisable(false);
   }
   auth.selectedTogglePproperty().addListener((observable, oldValue, newValue) -> {
      handleRoleChange(newValue);
   });
}
private void handleRoleChange(Toggle newValue) {
   if(authRadio.isSelected()) {
      confirmPassword.setDisable(true);
      confirmLabel.setDisable(true);
   }else{
      confirmPassword.setDisable(false);
      confirmLabel.setDisable(false);
   }
}
public void confirmation(ActionEvent event) throws IOException {
   if(authRadio.isSelected()){
      authorization(event);
   }else{
      registration();
   }
}
private void registration() throws IOException {
   String login = this.login.getText();
   String password = this.password.getText();
   String confirmPassword = this.confirmPassword.getText();
   if(password.equals(confirmPassword)) {
      HttpResponse response = HttpServerRequest.registerClient(login, confirmPassword);
      if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
         errorLabel.setText("Удала реєстрація");
         errorLabel.setStyle("-fx-text-fill: green;");
      } else {
         System.out.println("Register request failed with status code: " + response.getStatusLine().getStatusCode());
         errorLabel.setText("Помилка реєстрації");
         errorLabel.setStyle("-fx-text-fill: red;");
      }
   }else{
      errorLabel.setText("Паролі не збігаються");
      errorLabel.setStyle("-fx-text-fill: red;");
   }
}
private void authorization(ActionEvent event) throws IOException {
   String login = this.login.getText();
   String password = this.password.getText();
   token = HttpServerRequest.loginClient(login, password);
   if(token.equals("")){
      errorLabel.setText("Невдала авторизація");
```

```
            errorLabel.setStyle("-fx-text-fill: red;");
        }else{
            Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
            openModeWindow();
            currentStage.close();
        }
    }
    private void openModeWindow() throws IOException {
        FXMLLoader loader = new FXMLLoader(ChatApp.class.getResource("modeForm.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        Stage newFormStage = new Stage();
        newFormStage.setTitle("Вибір режиму");
        newFormStage.setScene(scene);
        newFormStage.show();
    }
}
package org.example.chatapplication;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;
public class ChatApp extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        FXMLLoader loader = new FXMLLoader(ChatApp.class.getResource("authForm.fxml"));
        AnchorPane root = loader.load();
        Scene scene = new Scene(root);
        primaryStage.setTitle("Авторизація");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
package org.example.chatapplication;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.TextArea;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.IOException;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.UnknownHostException;
import java.nio.ByteBuffer;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.nio.charset.StandardCharsets;
import java.security.InvalidAlgorithmParameterException;
```

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import static org.example.chatapplication.ClientSocketHandler.getTime;
import static org.example.chatapplication.ModeController.hostStatus;
public class ChatController {
    public static String keyAddress;
    public SocketChannel serverSocketChannel;
    @FXML
    private ListView<String> messageList;
    @FXML
    private Label statusLabel;
    @FXML
    private TextArea messageInput;
    private ObservableList<String> messages;
    @FXML
    public void initialize() {
        messageList.setStyle(".list-view.scroll-bar:horizontal.increment-arrow,\n" +
                ".list-view.scroll-bar:horizontal.decrement-arrow,\n" +
                ".list-view.scroll-bar:horizontal.increment-button,\n" +
                ".list-view.scroll-bar:horizontal.decrement-button {\n" +
                "    -fx-opacity: 0; \n" +
                "}\n");
        if(hostStatus){
            statusLabel.setText("Сервер");
        }else{
            statusLabel.setText("Клієнт");
        }
        Thread notFX = new Thread(() -> {
            if (hostStatus) {
                try {
                    serverStatus();
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            } else {
                listeningStatus();
            }
        });
        notFX.start();
    }
    public void listeningStatus() {
        ArrayList<String> servers = HttpServerRequest.getServers();
        if(servers !=null && servers.size()>0) {
            messages = FXCollections.observableArrayList(servers);
            messageList.setItems(messages);
            messageList.setOnMouseClicked(mouseEvent -> {
                if (mouseEvent.getClickCount() == 2) {
                    try {
                        serverSelection();
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                }
            });
        }else{
            System.out.println("empty");
        }
    }
```

```
private void serverSelection() throws IOException {
    System.out.println("Server selection " + Thread.currentThread());
    clientStatus(messageList.getSelectionModel().getSelectedItem());
}
public void serverStatus() throws IOException {
    ServerSocketChannel serverSocketChannel;
    try {
        serverSocketChannel = ServerSocketChannel.open();
        serverSocketChannel.bind(new InetSocketAddress(5000));
        serverSocketChannel.configureBlocking(true);
        String inetAddress = InetAddress.getLocalHost().getHostAddress();
        keyAddress = inetAddress;
        InetSocketAddress inetSocketAddress = new InetSocketAddress(inetAddress, 5000);
        HttpServerRequest httpPostServerRequest = new HttpServerRequest(inetAddress);
        httpPostServerRequest.hostServer();
        Thread accept = new Thread(() -> {
            while (true) {
                try {
                    SocketChannel socketChannel = serverSocketChannel.accept();
                    System.out.println(socketChannel.getRemoteAddress() + " have connected");
                    ClientSocketHandler clientSocket = new ClientSocketHandler(socketChannel, messageList);
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        });
        accept.start();
    } catch (IOException e) {
    }
}
private static void sendMessage(SocketChannel clientChannel, String message) throws IOException {
    ByteBuffer buffer = StandardCharsets.UTF_8.encode(message);
    clientChannel.write(buffer);
    buffer.clear();
}
private void clientStatus(String inetAddress) throws IOException {
    Thread clientStatus = new Thread(()->{
        keyAddress = inetAddress;
        System.out.println("Starting as a client");
        System.out.println("*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-.-*-\n");
        System.out.println(Thread.currentThread());
        try {
            serverSocketChannel = SocketChannel.open(new InetSocketAddress(inetAddress, 5000));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        System.out.println(new InetSocketAddress(inetAddress, 5000));
        Platform.runLater(()->{
            messageList.getItems().add("Установлено підключення \n");
            System.out.println();
        });
        try {
            serverSocketChannel.configureBlocking(true);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        ByteBuffer buffer = ByteBuffer.allocate(1024);

        while (true) {
```

```
            int bytesRead = 0;
            try {
               bytesRead = serverSocketChannel.read(buffer);
            } catch (IOException e) {
               throw new RuntimeException(e);
            }
            if (bytesRead == -1) {
               break;
            }
            buffer.flip();
            String message = StandardCharsets.UTF_8.decode(buffer).toString();
            String last;
            try {
               last = decrypt(message);
            } catch (UnknownHostException | BadPaddingException | InvalidKeyException |
                  InvalidAlgorithmParameterException | NoSuchPaddingException | IllegalBlockSizeException |
                  NoSuchAlgorithmException e) {
               throw new RuntimeException(e);
            }
            StringBuilder result = new StringBuilder();
            for (int i = 0; i < last.length(); i += 57) {
               int endIndex = Math.min(i + 57, last.length());
               result.append(last, i, endIndex).append(System.lineSeparator());
            }
            String formattedString = result.substring(0, result.length() - 2);
            ;
            if(!last.isEmpty())
               Platform.runLater(()->{
                  messageList.getItems().add(formattedString);
               });
            buffer.clear();
         }
         try {
            serverSocketChannel.close();
         } catch (IOException e) {
            throw new RuntimeException(e);
         }
      });
      clientStatus.start();
   }
   public void sendMessage() throws IOException, InvalidAlgorithmParameterException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {
      if(hostStatus){
         sendMessageByHost();
      }else{
         sendMessageByClient();
      }
   }
   private void sendMessageByClient() throws IOException, InvalidAlgorithmParameterException,
NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException,
InvalidKeyException {
      String message = messageInput.getText();
      String last = encrypt(message);
      sendMessage(serverSocketChannel, last);
      messageInput.setText("");
   }
   private void sendMessageByHost( ) throws IOException, InvalidAlgorithmParameterException,
NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException,
InvalidKeyException {
```

```
        String message = messageInput.getText();
        Command.SERVERDEFAULTMESSAGE(message);
        messageList.getItems().add(getTime()+" | Server: " + message);
        messageInput.setText("");
    }
    public static String decrypt(String bufferContent) throws UnknownHostException,
InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        byte[] keyIv = KeccakHash.getHash512(keyAddress);
        byte[] key = new byte[24];
        byte[] iv = new byte[16];
        System.arraycopy(keyIv,0,key,0,24);
        System.arraycopy(keyIv,0,iv,0,16);
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
        String checkedBuffer = KeccakHash.checkHash(bufferContent);
        if(checkedBuffer!=null){
            return AES.decrypt(checkedBuffer,secretKeySpec,ivParameterSpec);
        }else{
            return "Помилка передачі";
        }
    }
    public static String encrypt(String message) throws IOException, InvalidAlgorithmParameterException,
NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException,
InvalidKeyException {
        byte[] keyIv = KeccakHash.getHash512(InetAddress.getLocalHost().getHostAddress());
        byte[] key = new byte[24];
        byte[] iv = new byte[16];
        System.arraycopy(keyIv,0,key,0,24);
        System.arraycopy(keyIv,0,iv,0,16);
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
        IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
        return KeccakHash.setHash(AES.encrypt(message,secretKeySpec,ivParameterSpec));
    }
}
package org.example.chatapplication;
import javafx.application.Platform;
import javafx.scene.control.ListView;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SocketChannel;
import java.nio.charset.StandardCharsets;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.time.Instant;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
public class ClientSocketHandler {
    public volatile boolean isOpen =true;
    public String name;
    public SocketChannel clientChannel;
    public Instant start;
    public static ArrayList<ClientSocketHandler> clients = new ArrayList<>();
    public static int kUser=0;
```

```
public volatile boolean readBoolean = true;
public String STOPWORD;
public static ListView<String> listView;
public static ClientSocketHandler getClientSocketHandler(SocketChannel socketChannel) {
   for(ClientSocketHandler ClientSocketHandler: clients){
      if(ClientSocketHandler.clientChannel==socketChannel){
         return ClientSocketHandler;
      }
   }
   return null;
}
public static ClientSocketHandler getClientSocketHandlerByName(String name){
   for(ClientSocketHandler ClientSocketHandler: clients){
      if(ClientSocketHandler.name.equals(name)){
         return ClientSocketHandler;
      }
   }
   return null;
}
public ClientSocketHandler(SocketChannel clientChannel, ListView<String> messageList){
   clients.add(this);
   this.clientChannel = clientChannel;
   listView = messageList;
   this.name = "User#" + kUser;
   kUser++;
   start = Instant.now();
   Thread clientThread = new Thread(new Runnable() {
      @Override
      public void run() {
         try {
            ByteBuffer buffer = ByteBuffer.allocate(1024);
            while (readBoolean) {
               int bytesRead = clientChannel.read(buffer);
               if (bytesRead == -1) {
                  break;
               }
               buffer.flip();
               String message = StandardCharsets.UTF_8.decode(buffer).toString();
               String last = ChatController.decrypt(message);
               if (!last.isEmpty()) {
                  String[] words = last.split(" ");
                  switch (words[0]) {
                     case "NAME" -> Command.NAME(clientChannel, last);
                     case "STAT" -> Command.STAT(clientChannel);
                     case "MESG" -> Command.MESG(clientChannel, last);
                     case "QUIT" -> Command.QUIT(clientChannel);
                     case "TIME" -> Command.TIME(clientChannel, last);*/
                     default -> Command.BCST(clientChannel, last);
                  }
                  if(getClientSocketHandler(clientChannel)!=null) {
                     last = getTime() + " | " + getClientSocketHandler(clientChannel).name + ": " + last;
                     StringBuilder result = new StringBuilder();
                     for (int i = 0; i < last.length(); i += 57) {
                        int endIndex = Math.min(i + 57, last.length());
                        result.append(last.substring(i, endIndex)).append(System.lineSeparator());
                     }
                     String formattedString = result.substring(0, result.length() - 2);
                     Platform.runLater(() -> {
                        messageList.getItems().add(formattedString);
```

```
                    });
                    buffer.clear();
                }
            }
        }
        clients.remove(clientChannel);
        clientChannel.close();
      } catch (IOException | InvalidAlgorithmParameterException | NoSuchPaddingException |
            IllegalBlockSizeException | NoSuchAlgorithmException | BadPaddingException |
            InvalidKeyException e) {
        e.printStackTrace();
      }
      finally {
        try {
          System.out.println(clientChannel.getLocalAddress() + " have disconnected");
          isOpen = false;
        } catch (IOException e) {
        }
      }
    }
  });
  clientThread.start();
 }
 public static void sendMessageByServer(SocketChannel clientChannel, String message) throws IOException {

  ByteBuffer buffer = StandardCharsets.UTF_8.encode(message);
  clientChannel.write(buffer);
  buffer.clear();
 }
 public static String getTime(){
  LocalTime currentTime = LocalTime.now();
  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");
  return currentTime.format(formatter);
 }
}
package org.example.chatapplication;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.io.IOException;
import java.nio.channels.SocketChannel;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.time.Duration;
import java.time.Instant;
import java.util.Random;
import static org.example.chatapplication.ClientSocketHandler.*;
//STOPWORD NAME
public class Command {
 public static void NAME(SocketChannel socketChannel, String message) throws IOException {
  String[] words = message.split(" ");
  if (words.length > 1)
   if(clients.contains(getClientSocketHandlerByName(words[1]))){
    getClientSocketHandler(socketChannel).name = generateNick();
   }else{
    getClientSocketHandler(socketChannel).name = words[1];
   }
  else
```

```
            getClientSocketHandler(socketChannel).name = generateNick();
        sendMessageByServer(socketChannel, "Now your nick is - " + getClientSocketHandler(socketChannel).name);
    }
    public static void TIME(SocketChannel socketChannel, String message) throws IOException {
        String[] words = message.split(" ");
        if(words.length<2){
            sendMessageByServer(socketChannel, "Try valid form: TIME <name>");
        }else{
            if(getClientSocketHandlerByName(words[1])!=null){
                Instant sessionEnd = Instant.now();
                Duration duration = Duration.between(getClientSocketHandlerByName(words[1]).start, sessionEnd);
                sendMessageByServer(socketChannel, "The duration of asked session is - " + duration.getSeconds() + "
seconds");
            }else{
                sendMessageByServer(socketChannel, "There is no such person on the server");
            }
        }

    }
    public static void MESG(SocketChannel socketChannel, String message) throws IOException {
        String[] words = message.split(" ");
        boolean sent = false;
        StringBuilder content= new StringBuilder();
        for(int i = 2; i<words.length;i++){
            content.append(words[i]).append(" ");
        }
        String rest = new String(content);
        if(words.length>=3){
            for(ClientSocketHandler ClientSocketHandler: clients){
                if(ClientSocketHandler.name.equals(words[1])){
                    sendMessageByServer(ClientSocketHandler.clientChannel,  getTime()+"|" +
getClientSocketHandler(socketChannel).name + ": " + rest );
                    if(getClientSocketHandlerByName(words[1]).STOPWORD!=null){
                        String word = getClientSocketHandlerByName(words[1]).STOPWORD;
                        if(message.contains(word)){
                            String[] commands = message.split(" ");
                            getClientSocketHandlerByName(commands[1]).clientChannel.close();
                        }
                    }
                    return;
                }
            }
            sendMessageByServer(socketChannel, "I don't know a person with that name. :( Try another one.");
        }

    }
    public static void BCST(SocketChannel socketChannel, String message) throws IOException,
InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        for(ClientSocketHandler ClientSocketHandler: clients){
            sendMessageByServer(ClientSocketHandler.clientChannel, ChatController.encrypt(getTime()+" | " +
getClientSocketHandler(socketChannel).name + ": " +message));
        }
    }
    public static void STAT(SocketChannel socketChannel) throws IOException {
        StringBuilder names = new StringBuilder();
        sendMessageByServer(socketChannel, "There are some active client on the server:");
        for(ClientSocketHandler ClientSocketHandler: clients){
            if(ClientSocketHandler.name.equals(getClientSocketHandler(socketChannel).name)){
```

```
            names.append(ClientSocketHandler.name).append(" - is You").append("\n");
          }else{
            names.append(ClientSocketHandler.name).append("\n");
          }
        }
        names.deleteCharAt(names.length()-1);
        sendMessageByServer(socketChannel, names.toString());
    }
    public static void QUIT(SocketChannel socketChannel) throws IOException {
        System.out.println(socketChannel.getLocalAddress() + " have disconnected from the server!");
        sendMessageByServer(socketChannel, "You`ve got disconnected");
        getClientSocketHandler(socketChannel).readBoolean=false;
        clients.remove(getClientSocketHandler(socketChannel));
        socketChannel.close();
    }
    public static String generateNick(){
        String[] prefix = {"MC", "OG", "MVP", "LIL", "PNB", "MF", "GOAT", "A$AP", "YG"};
        String[] main = {"Chepsik", "p0pa", "$a$hka", "Bebrik", "Tuzik", "Chicha", "Java", "Enj0oeyr"};
        Random random = new Random();
        String name = prefix[random.nextInt(prefix.length)] + main[random.nextInt(main.length)];
        return name;
    }
    public static void SERVERDEFAULTMESSAGE(String message) throws IOException,
InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        String[] words = message.split(" ");
        StringBuilder stringBuilder = new StringBuilder();
        for(int i =0;i< words.length; i++){
            stringBuilder.append(words[i]).append(" ");
        }
        for(ClientSocketHandler ClientSocketHandler: clients){
            String response = ChatController.encrypt(getTime() +" | "+ "Server: "+ stringBuilder);
            sendMessageByServer(ClientSocketHandler.clientChannel, response);
        }
    }
}
package org.example.chatapplication;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPatch;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.json.JSONArray;
import org.json.JSONObject;
import java.io.IOException;
import java.util.ArrayList;
public class HttpServerRequest {
    private String inetAddress;
    private final int PORT = 5000;
    protected final static String ADD_ENDPOINT = "http://localhost:8080/pool/addServer";
    protected final static String DELETE_ENDPOINT = "http://localhost:8080/pool/deleteServer";
    protected final static String GET_ENDPOINT = "http://localhost:8080/pool/getServers";
    protected final static String REGISTER_ENDPOINT = "http://localhost:8080/auth/register";
    protected final static String LOGIN_ENDPOINT = "http://localhost:8080/auth/login";
```

```
public HttpServerRequest(String inetAddress) {
   this.inetAddress = inetAddress;
}
public void hostServer() {
   HttpClient httpClient = HttpClients.createDefault();
   // Set the request parameters
   String requestBody = "{\"inetAddress\": \"" + inetAddress + "\"}";
   System.out.println(requestBody);
   // Create HttpPost instance
   HttpPost httpPost = new HttpPost(ADD_ENDPOINT);
   httpPost.setHeader("Authorization", "Bearer " + AuthController.token);
   httpPost.setHeader("Content-Type", "application/json");
   try {
      StringEntity requestEntity = new StringEntity(requestBody);
      httpPost.setEntity(requestEntity);
      HttpResponse response = httpClient.execute(httpPost);
      if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
         System.out.println("Host server request successful");
      } else {
         System.out.println("Request failed with status code: " + response.getStatusLine().getStatusCode());
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
}
public void unhostServer() {
   HttpClient httpClient = HttpClients.createDefault();
   // Set the request parameters
   String requestBody = "{\"inetAddress\": \"" + inetAddress + "\"}";
   // Create HttpPost instance
   HttpPatch httpPatch = new HttpPatch(DELETE_ENDPOINT);
   httpPatch.setHeader("Content-Type", "application/json");
   try {
      StringEntity requestEntity = new StringEntity(requestBody);
      httpPatch.setEntity(requestEntity);
      HttpResponse response = httpClient.execute(httpPatch);
      if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
         System.out.println("Request successful");
      } else {
         System.out.println("Request failed with status code: " + response.getStatusLine().getStatusCode());
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
}
public static ArrayList<String> getServers(){
   HttpClient httpClient = HttpClients.createDefault();
   // Create HttpPost instance
   HttpGet httpGet = new HttpGet(GET_ENDPOINT);
   System.out.println(AuthController.token);
   httpGet.setHeader("Authorization", "Bearer " + AuthController.token);
   httpGet.setHeader("Content-Type", "application/json");
   try {
      HttpResponse response = httpClient.execute(httpGet);
      if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
         System.out.println("Request successful");
         HttpEntity responseEntity = response.getEntity();
         String responseBody = EntityUtils.toString(responseEntity);
         JSONArray jsonArray = new JSONArray(responseBody);
```

```
            System.out.println(jsonArray);
            ArrayList<String> list = new ArrayList<>();
            for(int i = 0; i < jsonArray.length(); i++) {
               list.add(jsonArray.getJSONObject(i).getString("inetAddress"));
               System.out.println(jsonArray.getJSONObject(i).getString("inetAddress"));
            }
            return list;

         } else {
            System.out.println("Request failed with status code: " + response.getStatusLine().getStatusCode());
         }
      } catch (IOException e) {
         e.printStackTrace();
      }
      return null;
   }
   public static HttpResponse registerClient(String login, String password) throws IOException {
      HttpClient httpClient = HttpClients.createDefault();
      HttpPost httpPost = new HttpPost(REGISTER_ENDPOINT);
      httpPost.setHeader("Content-Type", "application/json");
      String requestBody = "{\"username\": \"" + login + "\", \"password\": \"" + password + "\"}";
      System.out.println(requestBody);
      StringEntity requestEntity = new StringEntity(requestBody);
      httpPost.setEntity(requestEntity);
      return httpClient.execute(httpPost);
   }
   public static String loginClient(String login, String password) throws IOException {
      HttpClient httpClient = HttpClients.createDefault();
      HttpPost httpPost = new HttpPost(LOGIN_ENDPOINT);
      httpPost.setHeader("Content-Type", "application/json");
      String requestBody = "{\"username\": \"" + login + "\", \"password\": \"" + password + "\"}";
      StringEntity requestEntity = new StringEntity(requestBody);
      httpPost.setEntity(requestEntity);
      HttpResponse response = httpClient.execute(httpPost);
      if (response.getStatusLine().getStatusCode() >= 200 && response.getStatusLine().getStatusCode() < 300) {
         String responseBody = EntityUtils.toString(response.getEntity());
         JSONObject jsonObject = new JSONObject(responseBody);
         String token = jsonObject.getString("jwt");
         System.out.println("Login request successful");
         System.out.println(responseBody);
         return token;
      } else {
         System.out.println("Register request failed with status code: " + response.getStatusLine().getStatusCode());
         return null;
      }
   }
}
package org.example.chatapplication;
import org.bouncycastle.jcajce.provider.digest.Keccak;
import org.bouncycastle.util.encoders.Hex;
import java.nio.charset.StandardCharsets;
public class KeccakHash {
   public static String getHash256(String content){
      Keccak.Digest256 digest256 = new Keccak.Digest256();
      byte[] hashbytes = digest256.digest(content.getBytes());
      String sha3Hex = new String(Hex.encode(hashbytes));
      return sha3Hex;
   }
   public static byte[] getHash512(String content){
```

```
        Keccak.Digest512 digest512 = new Keccak.Digest512();
        return digest512.digest(content.getBytes(StandardCharsets.UTF_8));
    }
    public static String setHash(String content){
        return content+getHash256(content);
    }
    public static String checkHash(String content){
        String main = content.substring(0, content.length()-64);
        String hash = content.substring(content.length()-64);
        if(getHash256(main).equals(hash)){
            return main;
        }else{
            return null;
        }
    }
}
package org.example.chatapplication;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import java.io.IOException;
public class ModeController {
    @FXML
    Button hostButton;
    @FXML
    Button clientButton;
    public static boolean hostStatus;

    public void hostButtonClicked(ActionEvent event) throws IOException {
        Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        currentStage.close();
        hostStatus = true;
        FXMLLoader loader = new FXMLLoader(ChatApp.class.getResource("chatForm.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        Stage newFormStage = new Stage();
        newFormStage.setTitle("Чат серверу");
        newFormStage.setScene(scene);
        newFormStage.show();
    }
    public void clientButtonClicked(ActionEvent event) throws IOException {
        Stage currentStage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        currentStage.close();
        hostStatus = false;
        FXMLLoader loader = new FXMLLoader(ChatApp.class.getResource("chatForm.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        Stage newFormStage = new Stage();
        newFormStage.setTitle("Чат клієнта");
        newFormStage.setScene(scene);
        newFormStage.show();
    }
}
```

Додаток В. Ілюстративний матеріал

Розробка захищеного додатку обміну конфіденційними даними в корпоративній мережі на основі технології сокетів

Виконав студент 2КІТС-20б Ільчук Р.В.
Науковий керівник: к.т.н., доцент каф. МБІС Карпінець В.В.

АКТУАЛЬНІСТЬ РОБОТИ

З розвитком інформаційних технологій та збільшенням обсягів корпоративних даних, забезпечення безпеки при обміні конфіденційною інформацією в корпоративних мережах стало критично важливим.

Об'єктом дослідження є процес розробки захищеного додатку для обміну конфіденційними даними на основі технології сокетів.

Метою цієї роботи є створення захищеного додатку для обміну конфіденційними даними в корпоративній мережі за допомогою технології сокетів, що забезпечує безпеку, приватність і надійність комунікацій.

ПРЕДМЕТОМ ДОСЛІДЖЕННЯ

є розробка додатку для обміну конфіденційними даними в корпоративній мережі за допомогою сокетів. Включає проектування архітектури, створення безпечного комунікаційного протоколу та інтеграцію у корпоративний додаток.

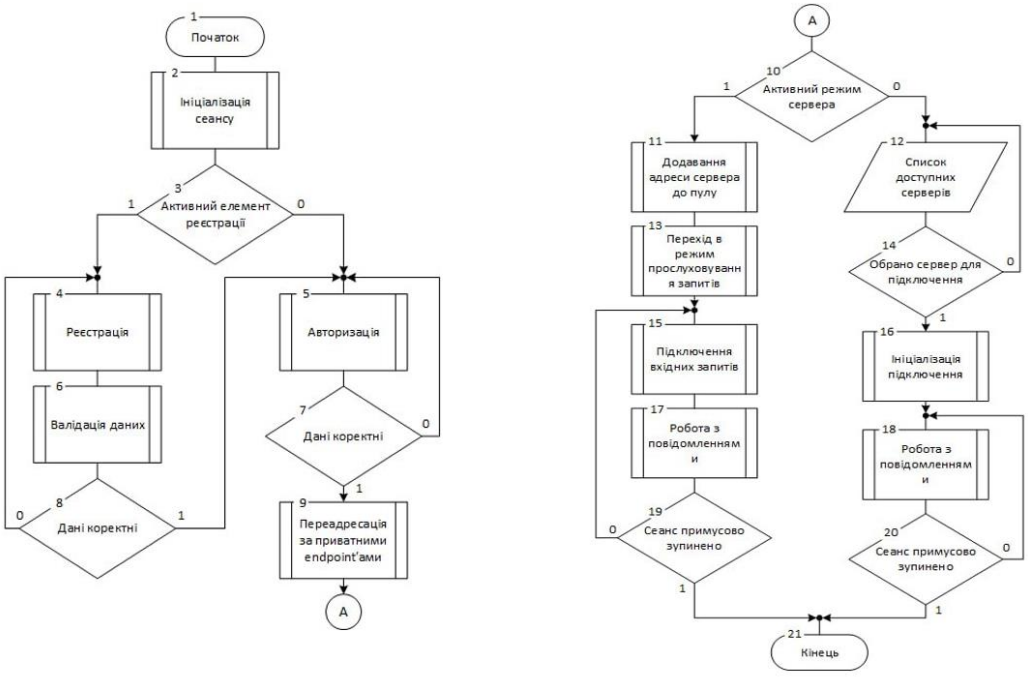
НОВИЗНОЮ ДАНОЇ РОБОТИ

У даній роботі новаторським є поєднання технології сокетів із сучасними методами криптографічного захисту даних. Використання шифрування AES, автентифікації через Spring Security та хешування SHA-3 ускладнює перехоплення та розшифрування інформації, забезпечуючи високий рівень безпеки й конфіденційності у корпоративних мережах.

ПРАКТИЧНА ЦІННІСТЬ

Результати цієї роботи корисні для розробників та користувачів корпоративних додатків. Новий додаток на основі сокетів забезпечує вищий рівень безпеки та конфіденційності, завдяки використанню сучасних методів захисту. Це сприяє зменшенню ризику витоку даних і підвищенню загального рівня кібербезпеки в корпоративних мережах.

АЛГОРИТМ РОБОТИ ДОДАТКУ



ВИКОРИСТАНІ АЛГОРИТМИ ШИФРУВАННЯ ТА ХЕШУВАННЯ

AES є одним з найбільш поширених симетричних алгоритмів шифрування, який відомий своєю ефективністю та безпекою. Він використовується для шифрування конфіденційних даних у багатьох сучасних системах.

SHA-3 (Keccak) є одним з найбільш стійких алгоритмів хешування, який відомий своєю стійкістю до криптографічних атак. Він використовується для забезпечення цілісності даних та автентифікації.

Порівння з аналогами

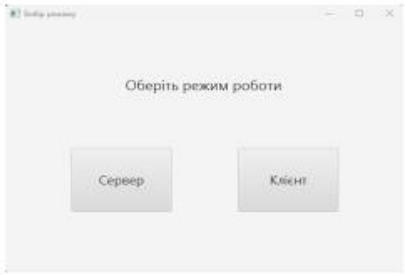
Месенджери	Trello	Asana	Slack	Signal	Padio	Розроблений месенджер
Безпека	Використовує шифрування SSL/TLS	Використовує шифрування SSL/TLS та двофакторну аунтефікацію	Використовує шифрування SSL/TLS, OAuth 2.0 для безпечної авторизації через сторонні додатки і сервіси	Використовує наскрізне шифрування End-to-End Encryption (E2EE)	Використовує шифрування SSl/TLS та шифрування (E2EE)	Використовує алгоритми AES і SHA-3 і фреймворк Spring Security
Робота мережі	Trello працює на базі хмарної інфраструктури	Доступ з будь-якого пристрою з інтернетом	Slack використовує WebSockets для обміну повідомленнями в реальному часі.	Signal використовує децентралізовану хмарну інфраструктуру для обробки повідомлень і дзвінків	Додаток Padio використовує хмарні сервіси	Використовує ServerSocketChannel та SocketChannel
Інтерфейс та зручність використання	Додаток має інтуїтивний дизайн: Зрозумілий і доступний інтерфейс.	Інтуїтивно зрозумілий інтерфейс зі зручними функціями	Зручний та легко зрозумілий навіть для нових користувачів	Складний інтерфейс, зорієнтований на бізнес-користувачів	Легкий для освоєння інтерфейс	Простий та зручний інтерфейс

ТЕХНОЛОГІЇ

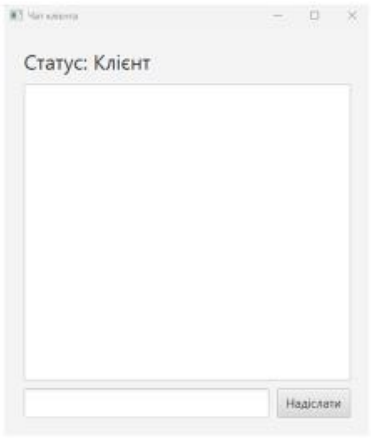
Java PostgreSQL JavaFX SpringBoot Spring Security

ВІКНО АВТОРИЗАЦІЇ І РЕЄЕСТРАЦІЇ

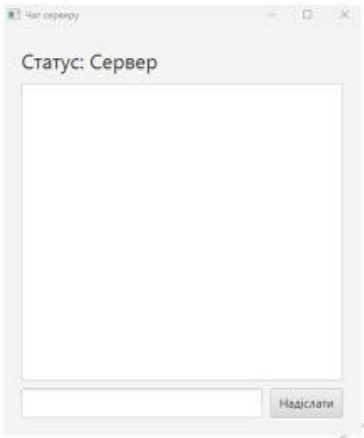
ВІКНО ВИБОРУ РЕЖИМУ РОБОТИ



ВІКНО ЧАТУ КЛІЄНТА



ВІКНО ЧАТУ СЕРВЕРА



ВИСНОВКИ

Отже, можна зробити висновок, що розроблений додаток для обміну конфіденційними даними є ефективним рішенням для забезпечення захисту інформації в корпоративній мережі. Використання технології сокетів у поєднанні з алгоритмами AES та SHA-3, а також фреймворком Spring Security, забезпечує високий рівень безпеки, конфіденційності та цілісності переданих даних, що є критично важливим для захисту корпоративної інформації.

Дякую за увагу

Додаток Г. Протокол перевірки на атиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ
ЗАПОЗИЧЕНЬ

Назва роботи: Розробка захищеного додатку обміну конфіденційними даними в корпоративній мережі на основі технології сокетів

Тип роботи: бакалаврська дипломна робота
(БДР. МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра. факультет)

Показники звіту подібності Unicheck

Оригінальність 95 % Схожість 5 %

Аналіз звіту подібності (відмітити потрібне):

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

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. 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.

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Ільчук Р.В.
(підпис) (прізвище, ініціали)

Керізник роботи _____ Карпінець В.В.
(підпис) (прізвище, ініціали)