Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технології

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Древинський В. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Коваленко О. О.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Крупельницький Л. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

О. Н. Романюк

«21» березня 2025 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Древинському Владиславу Валентиновичу

1. Тема роботи – Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технології

Керівник роботи: Коваленко Олена Олексіївна, кандидат техн. наук, доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025

3. Вихідні дані до роботи: відомі методи формування системи комунікацій, передачі повідомлень; методи використання блокчейну для систем комунікацій та документообігу.

4. Зміст текстової частини: вступ; аналіз стану систем корпоративних комунікацій та постановка задач розробки; розробка структури та моделі програмного продукту; розробка програмних модулів; тестування програми; висновки.

5. Перелік ілюстративної частини: актуальність розробки; наукова новизна; аналіз аналогів; структура та модель програмного продукту; основні програмні модулі; результати тестування; апробація результатів; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Коваленко О.О., доц. каф. ПЗ	24.03.25	30.05.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз стану систем корпоративних комунікацій та постановка задач розробки	25.03.25 – 10.04	вик
2	Розробка структури та моделі програмного продукту	11.04.25 – 22.04.25	вик
3	Розробка програмних модулів	23.04.25 – 15.05.25	вик
4	Тестування програми	16.05.25-22.05.25	вик
5	Оформлення матеріалів до захисту БКР	23.05.25– 30.05.25	вик

Студент _____ **В.В. Древинський**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **О.О. Коваленко**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 55 сторінок формату А4, на яких є 17 рисунків, 2 таблиці, список використаних джерел містить 27 найменувань.

У бакалаврській кваліфікаційній роботі розроблено програмні засоби соціальної мережі для забезпечення безпечного листування за допомогою блокчейн технології. Робота охоплює створення модулів шифрування повідомлень, автентифікації користувачів, цифрового підпису та розподіленого зберігання даних із застосуванням смарт-контрактів, що гарантують цілісність операцій. Реалізоване програмне забезпечення забезпечує високий рівень конфіденційності, безпеки та прозорості передачі інформації в режимі реального часу.

Програмне рішення реалізовано засобами Webstorm мови програмування JavaScript та середовища Node JS, а також фреймворку React. У результаті було створено засоби редагування та обробки даних із використанням веб ресурсів для реалізації інтерфейсу користувача; засоби збереження даних у базі даних MongoDB формату JSON.

Ключові слова: програмна інженерія, комунікації, блокчейн, цифровий підпис, корпоративна комунікація, розподілені системи, шифрування.

ABSTRACT

The bachelor's qualification work consists of 55 A4 pages, which contain 17 figures, 2 tables, and the list of sources used contains 27 names.

The bachelor's qualification work developed software tools for a social network to ensure secure correspondence using blockchain technology. The work includes the creation of modules for message encryption, user authentication, digital signature, and distributed data storage using smart contracts that guarantee the integrity of operations. The implemented software provides a high level of confidentiality, security, and transparency of information transmission in real time.

The software solution was implemented using the Webstorm JavaScript programming language and the Node JS environment, as well as the React framework. As a result, tools for editing and processing data using web resources were created to implement the user interface; tools for storing data in the MongoDB database in JSON format.

Keywords: software engineering, communications, blockchain, digital signature, corporate communication, distributed systems, encryption.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розвиток інформаційних технологій зумовлює необхідність постійного вдосконалення систем електронної комунікації, які є невід'ємною складовою сучасного корпоративного середовища. Зростання обсягів переданої інформації, а також вимоги до оперативності та конфіденційності обміну даними сприяють активному впровадженню цифрових рішень у сфері організаційної взаємодії. Однією з ключових вимог до таких систем є забезпечення захисту інформації, її автентичності та достовірності на всіх етапах передавання і зберігання [1].

Системи корпоративних комунікацій традиційно будуються на основі централізованих архітектур, що передбачає зосередження управління та зберігання даних на одному або кількох серверах. Такий підхід має ряд переваг, зокрема простоту адміністрування та швидкість впровадження. Проте в умовах постійного зростання кіберзагроз, а також загроз втрати або компрометації даних через людський або технічний фактор, централізовані рішення дедалі частіше демонструють вразливість [2].

Особливого значення набуває проблема достовірності корпоративного листування, що є компонентом внутрішньої діяльності організацій, на який треба звернути увагу. Порушення цілісності повідомлень або несанкціонований доступ до інформації може призвести до фінансових втрат, порушення репутації або зниження ефективності управлінських процесів. У зв'язку з цим виникає потреба у розробці децентралізованих рішень, які дозволяють гарантувати збереження автентичності, забезпечити прозорість обміну даними та унеможливити фальсифікацію повідомлень.

Блокчейн як механізм фіксації змін у розподілених журналах транзакцій забезпечує високий рівень довіри між користувачами та дозволяє організувати процес обміну повідомленнями без централізованого контролера. Незмінність записів, доступність історії змін, прозорість дій усіх учасників і математична

доказовість справжності – основні переваги, які зумовлюють актуальність використання цієї технології у системах корпоративного зв’язку.

Актуальність обраної теми полягає в необхідності створення безпечного та надійного інструменту корпоративної взаємодії, що забезпечить збереження історії обміну повідомленнями, підвищить рівень довіри до цифрових каналів комунікації та знизить ризик підробки чи зловживань. Проблематика формалізується в умовах зростаючих вимог до безпеки корпоративної інформації, а також потреби у технологічно обґрунтованих рішеннях, які дозволяють оптимізувати процес обміну повідомленнями у розподіленому середовищі.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета дослідження. Метою бакалаврської кваліфікаційної роботи є вдосконалення корпоративної комунікації шляхом розробки і використання веборієнтованої інформаційної системи, яка забезпечує захищений обмін повідомленнями між користувачами з використанням механізмів фіксації, автентифікації та верифікації даних на основі блокчейн-технологій.

Для досягнення поставленої мети передбачено вирішення таких завдань:

– проаналізувати сучасні засоби для корпоративної комунікації та засоби забезпечення їхньої безпеки;

– визначити вимоги до інформаційної системи з урахуванням актуальних загроз та особливостей середовища використання;

– удосконалити методи та визначити архітектуру програмної системи, включаючи модулі збереження, обміну та перевірки повідомлень;

– реалізувати програмні компоненти взаємодії користувачів, збереження даних та контролю цілісності на основі блокчейн;

– протестувати систему та оцінити її функціональну придатність, надійність і стійкість до порушень цілісності даних.

Об'єкт дослідження – процес розробки системи корпоративного електронного обміну повідомленнями.

Предмет дослідження – методи та засоби розроблення програмного забезпечення для захищеного збереження та обміну повідомленнями в корпоративному середовищі з використанням принципів розподілених реєстрів і цифрового підпису.

Методи дослідження. У процесі виконання роботи було використано методи аналізу та синтезу – для дослідження існуючих систем корпоративної комунікації й механізмів захисту даних, моделювання – для побудови логіки обміну повідомленнями в умовах децентралізованої інфраструктури, об’єктно-орієнтованого та логічного проєктування – для розробки структури системи та блокчейн-реєстрації повідомлень, а також емпіричного тестування – для перевірки працездатності реалізованого програмного продукту. Практичну реалізацію виконано за допомогою сучасних інструментів веброзробки, зокрема JavaScript, Node.js, React та MongoDB, із застосуванням криптографічних бібліотек для формування цифрового підпису та хешування даних.

Новизна отриманих результатів.

Запропоновано архітектуру програмного забезпечення корпоративної системи комунікацій, яка, на відміну від існуючих, базується на приватному блокчейні та забезпечує збереження автентичності повідомлень без використання централізованого сервера, що дозволяє підвищити швидкодію здійснення комунікацій з підтримкою високого рівня безпеки.

Удосконалено метод перевірки цілісності повідомлень у розподіленому середовищі, який, на відміну від існуючих, використовує методи криптографічного хешування в комбінації з цифровим підписом, що забезпечує верифікацію авторства без втрати продуктивності.

Подальшого розвитку набули методи до розробки безпечних вебзастосунків для внутрішніх комунікацій, які, на відміну від існуючих, використовують механізми перевірки достовірності повідомлень через

блокчейн-записи без залучення проміжних вузлів, що збільшує швидкодію без втрати рівня достовірності.

Практична цінність отриманих результатів полягає у можливості застосування розробленої інформаційної системи для створення або модернізації корпоративних каналів комунікації, що мають підвищені вимоги до безпеки та достовірності переданої інформації.

Особистий внесок здобувача. Усі наукові та прикладні результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто.

За результатами досліджень опубліковано двоє тез доповідей [3;4]:

1. Древинський В.В. Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технологій. Збірник матеріалів міжнародної науково-практичної конференції «Інформаційні технології в освіті та науці». URL: https://mdpu.org.ua/nauka/konferentsiyi-ta-seminary/

2. Древинський В.В. Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технологій. Збірник матеріалів міжнародної конференції «Молодь в науці 2025». URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025

Технічне завдання наведено в додатку А.

1 АНАЛІЗ СТАНУ СИСТЕМ КООРПОРАТИВНИХ КОМУНІКАЦІЙ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз стану систем корпоративних комунікацій

Електронні системи корпоративної комунікації становлять важливу складову інфраструктури підприємств, установ і організацій різних форм власності. У сучасних умовах цифровізації бізнес-процесів зростає потреба у таких рішеннях, що забезпечують безпечний, швидкий та контрольований обмін інформацією між учасниками внутрішньої та зовнішньої взаємодії. Стандартною практикою залишається використання централізованих клієнт-серверних моделей, у межах яких ключові функції збереження, обробки та передачі повідомлень покладаються на спеціалізовані сервери, що адмініструються організацією [5].

Сучасні продукти для корпоративних комунікацій пропонують широкий функціонал для внутрішнього листування, відеоконференцій, спільного редагування документів та інтеграцій з бізнес-додатками. Проте попри очевидну зручність, більшість з них залишаються вразливими до зовнішніх загроз, внутрішніх порушень політик доступу та маніпуляцій із боку користувачів, що мають розширені повноваження в системі.

Окрему проблему становить обмежена можливість підтвердження автентичності повідомлень у таких системах. За відсутності незалежного механізму фіксації та перевірки історії змін користувачі змушені покладатися на довіру до сервера і його адміністраторів. У ситуаціях конфлікту або внутрішніх аудитів відсутність прозорого журналу дій та незмінної історії комунікації унеможливлює об’єктивне відтворення подій.

У наукових публікаціях наголошується на необхідності запровадження нових архітектурних моделей для комунікаційних систем, що базуються на розподілених технологіях. Серед таких найбільш перспективною вважається

технологія блокчейн, яка через децентралізовану природу дозволяє фіксувати кожну подію у формі транзакції, яка не може бути змінена без збою мережі [6].

На практиці блокчейн використовується для вирішення завдань збереження доказів, захисту цифрової ідентичності, побудови децентралізованих систем голосування, фінансових обчислень, управління логістикою тощо. Використання цієї технології в контексті організації захищеного листування дозволяє автоматично забезпечувати [7;8]:

- незмінність повідомлень після їх надсилання;
- автентичність джерела;
- можливість перевірки цілісності;
- формування доказової бази у разі суперечок.

Аналіз існуючих рішень показує, що більшість популярних корпоративних комунікаційних платформ не інтегрують механізми децентралізованої фіксації дій користувачів. Таким чином, обґрунтовується потреба в розробці нового підходу до проєктування систем корпоративної комунікації, в основі якого лежить розподілена архітектура та використання блокчейн для збереження повідомлень.

1.2 Порівняльний аналіз аналогів

На сучасному етапі розвитку цифрових технологій представлено значну кількість програмних продуктів, що забезпечують функціонування корпоративних систем комунікацій. Серед найбільш поширених можна виокремити Microsoft Teams, Slack, Mattermost, Rocket.Chat та Element. Кожен з цих сервісів має набір функцій, орієнтованих на спрощення внутрішньої взаємодії в організаціях. Водночас жоден із них не реалізує повноцінний механізм фіксації повідомлень у децентралізованому середовищі з використанням технології блокчейн.

Сервіс Microsoft Teams є складовою екосистеми Microsoft 365. Його перевагами є висока якість відеозв’язку, широкі можливості для спільної роботи

з документами та зручне адміністрування. Проте, всі повідомлення та дані зберігаються на хмарних серверах Microsoft, що передбачає централізовану модель обробки та покладає довіру до безпеки виключно на постачальника сервісу [9].

Інтерфейс програмного застосунку «Microsoft Teams» зображено на рисунку 1.1.

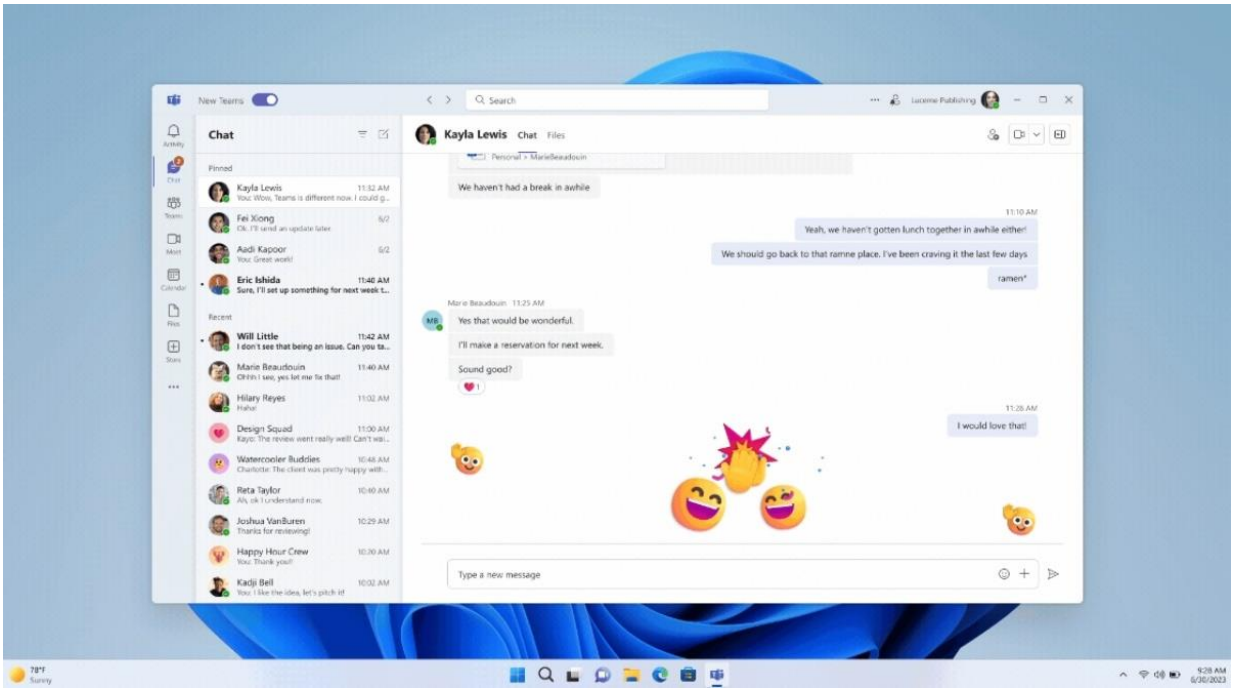


Рисунок 1.1 – Інтерфейс Microsoft Teams

Slack орієнтовано на використання у командній взаємодії. Система підтримує обмін повідомленнями, файлами, інтеграцію з зовнішніми сервісами та боти-автоматизатори. Незважаючи на функціональну гнучкість, Slack також використовує централізовану архітектуру. Механізми фіксації подій відсутні або реалізовані лише частково через додаткові платні функції, що не виключає потенційних маніпуляцій з боку адміністраторів [10].

Інтерфейс програмного застосунку «Slack» зображено на рисунку 1.2.

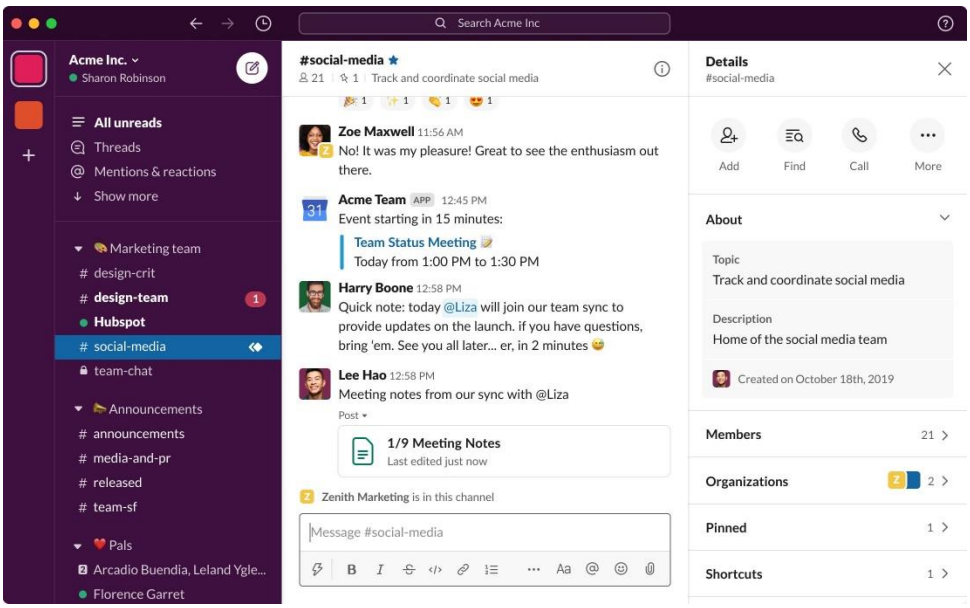


Рисунок 1.2 – Інтерфейс Slack

Mattermost позиціонується як безпечна альтернатива Slack із відкритим вихідним кодом. Система дозволяє розгортання на власних серверах організації, що частково усуває ризики, пов’язані із зовнішніми провайдерами та додає певні критерії безпеки всередині команди. Проте логіка роботи залишається централізованою, а контроль доступу та історії змін залежить від локального адміністрування. Підтримки блокчейн або інших розподілених механізмів у базовій версії не передбачено [11].

Інтерфейс програмного застосунку «Mattermost» зображено на рисунку 1.3.

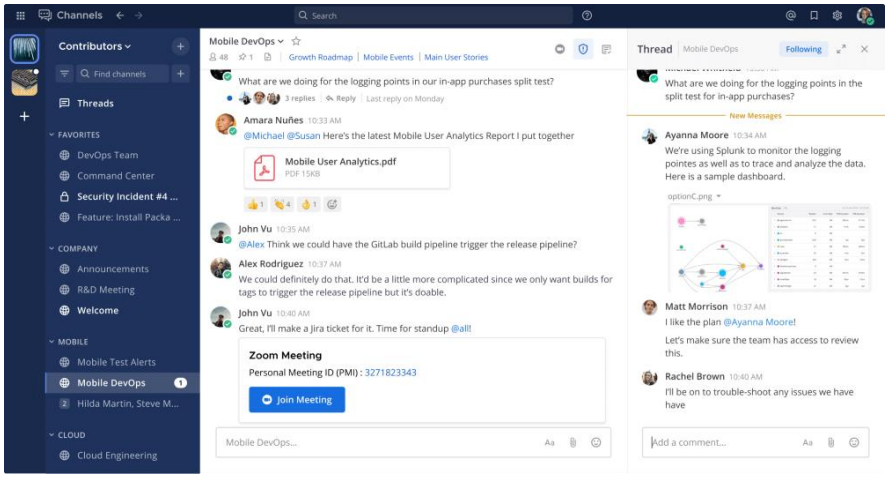


Рисунок 1.3 – Інтерфейс Mattermost

Rocket.Chat також є платформою з відкритим кодом, що дозволяє кастомізацію під потреби конкретної організації. Система реалізує базові інструменти комунікації та має розширення для шифрування повідомлень. Водночас прозорість історії взаємодії не забезпечується, оскільки всі дані зберігаються у звичайній базі даних, доступ до якої має адміністратор [12].

Інтерфейс програмного застосунку «Rocket.Chat» зображено на рисунку 1.4.

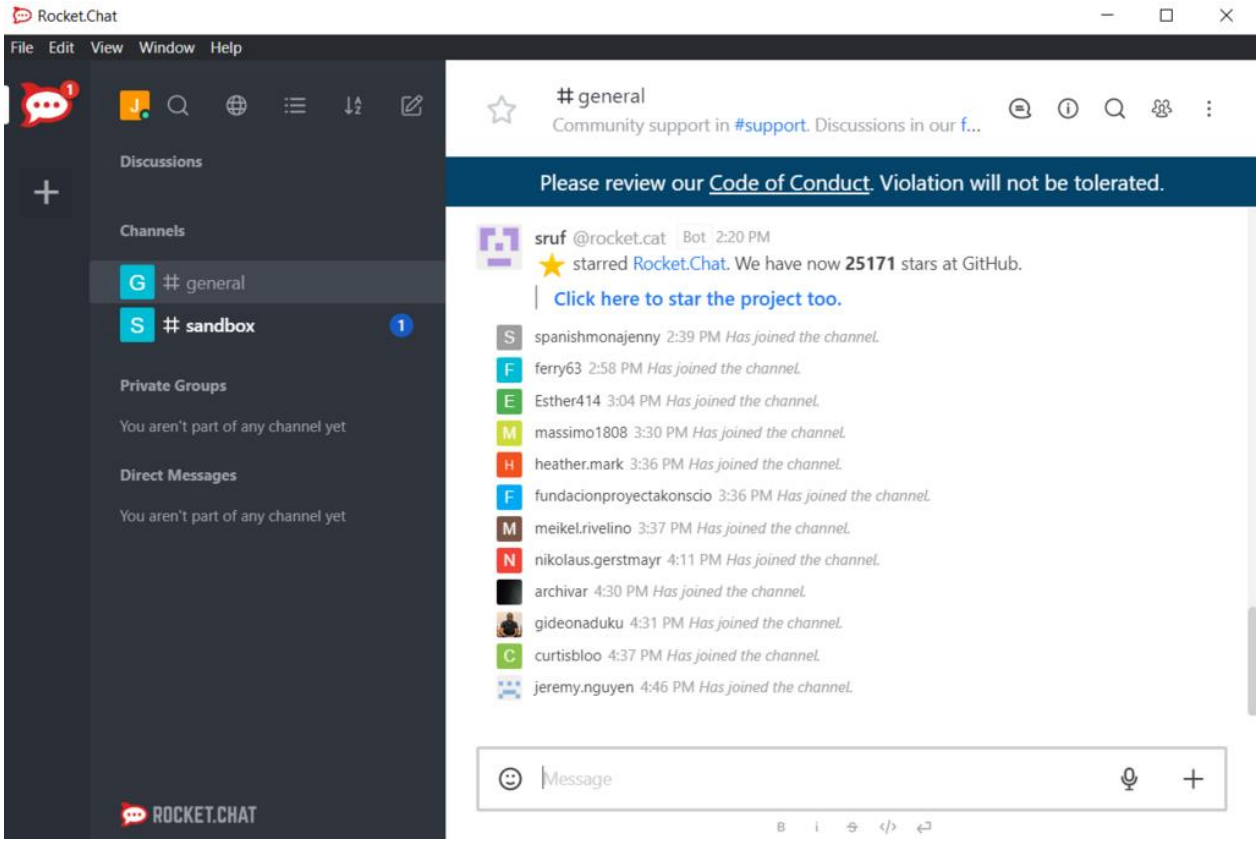


Рисунок 1.4 – Інтерфейс Rocket.Chat

Element, побудований на основі протоколу Matrix, частково підтримує децентралізацію через розподіл серверів (homeservers). Однак реалізація повноцінної незмінності повідомлень, типова для блокчейн-рішень, відсутня. Можливість їх редагування або видалення зберігається, що знижує рівень довіри до історії комунікації в різних середовищах та забезпечує нижий рівень інформаційної безпеки [13].

Інтерфейс програмного застосунку «Element» зображено на рисунку 1.5.

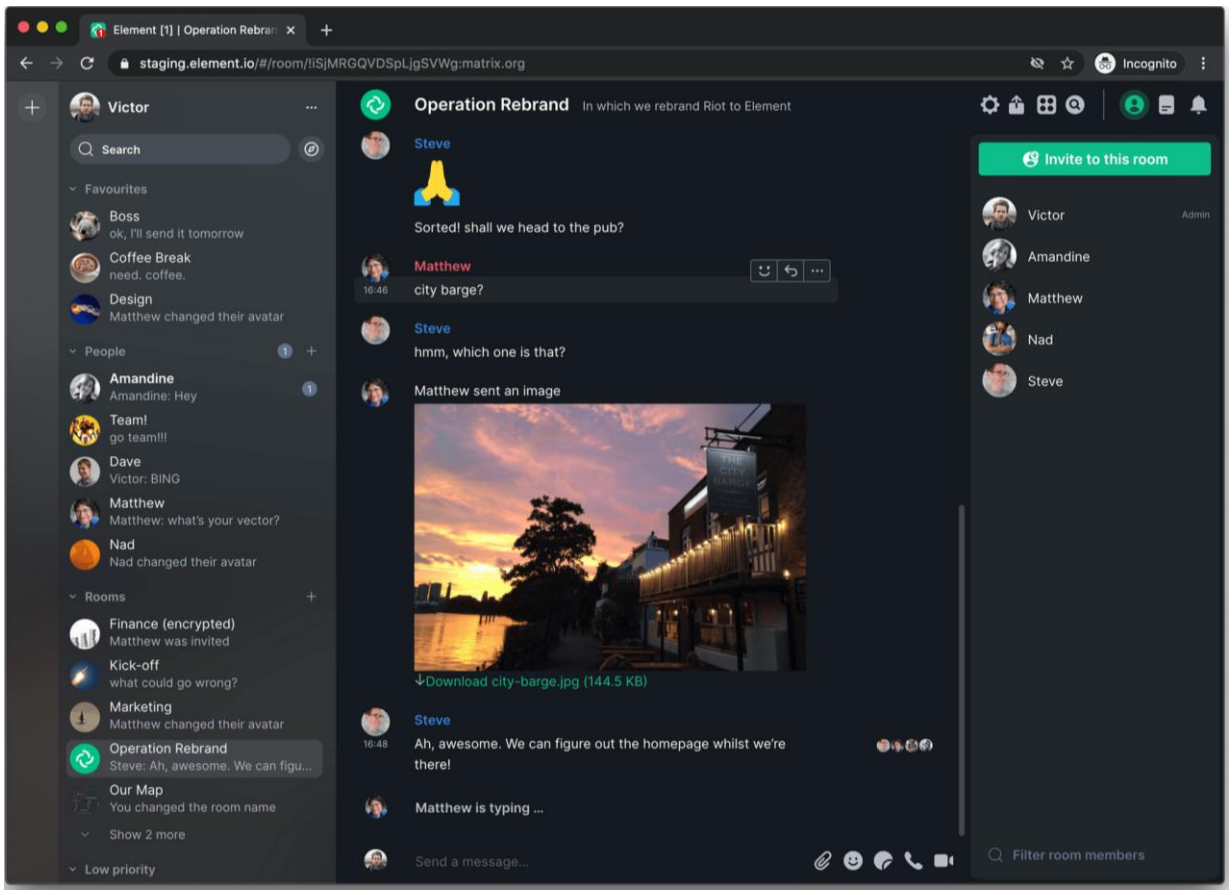


Рисунок 1.5 – Інтерфейс Element

Для узагальнення порівняльного аналізу розглянутих аналогів було сформовано таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз аналогічних систем корпоративної комунікації

Назва системи	Microsoft Teams	Slack	Mattermost	Rocket.Chat	Element	Власна розробка
1	2	3	4	5	6	7
Централізована архітектура	так	так	так	так	так	ні
Можливість локального розгортання	ні	ні	так	так	так	так
Підтримка шифрування	так	так	так	так	так	так

Продовження таблиці 1.1.

1	2	3	4	5	6	7

Журнал дій	ні	ні	ні	ні	ні	так
Блокчейн-фіксація	ні	ні	ні	ні	ні	так
Загальна оцінка	2	2	3	3	3	4

Аналіз дозволяє зробити висновок, що жодна з популярних платформ не забезпечує одночасно високий рівень прозорості, захищеності, децентралізованого зберігання та повної незмінності історії комунікації. Це обґрунтовує необхідність розробки нового рішення, яке б поєднувало зручність користування сучасними вебінструментами з перевагами блокчейн-технології для фіксації повідомлень у корпоративному середовищі.

1.3 Аналіз методів розв’язання задачі

На основі аналізу предметної області та розгляду існуючих аналогів систем корпоративної комунікації виявлено низку вимог до архітектури та функціональності програмного забезпечення. До таких вимог належать забезпечення цілісності повідомлень, захист від несанкціонованих змін, прозорість комунікаційного процесу, а також відсутність необхідності у централізованому контролі. Для розв’язання поставленої задачі доцільним є застосування методів, що поєднують технологічні принципи розподілених систем, криптографії та мережевих комунікацій.

Основним обґрунтованим підходом є впровадження технології блокчейн як базового механізму фіксації повідомлень у системі. Блокчейн дозволяє реалізувати зберігання інформації у вигляді послідовного ланцюга блоків, кожен з яких містить хеш попереднього, тим самим забезпечуючи незмінність історії подій. Для організації обміну повідомленнями кожна подія (відправлення повідомлення, його прочитання або редагування) може бути представлена у вигляді транзакції, що зберігається в розподіленому журналі [14].

Використання криптографічних хеш-функцій дозволяє фіксувати вміст повідомлень у стислому форматі, який не може бути зворотно розшифрований,

але легко перевіряється на автентичність. Таким чином, навіть мінімальні зміни у повідомленні призводять до зміни хешу, що дозволяє виявити спробу несанкціонованого редагування [15].

З метою забезпечення конфіденційності даних у процесі їх передавання доцільно застосовувати асиметричне шифрування з використанням відкритого та закритого ключів. Такий підхід гарантує, що лише адресат може розшифрувати повідомлення, навіть якщо його зафіксовано у публічному реєстрі. Крім того, за допомогою цифрового підпису можна підтвердити, що повідомлення надійшло саме від конкретного користувача, а не було згенероване сторонньою особою [16].

Для взаємодії між користувачем і системою рекомендовано використання веборієнтованої клієнт-серверної архітектури, в межах якої користувач взаємодіє з інтерфейсом через браузер, а логіка обробки запитів та генерації блоків здійснюється на серверному рівні. Такий підхід дозволяє забезпечити масштабованість, кросплатформенність і централізовану обробку запитів до блокчейн-модуля без порушення принципу децентралізованого зберігання [17].

На логічному рівні функціонування системи доцільно організувати у вигляді послідовності дій:

1. Користувач створює повідомлення.
2. Система формує хеш повідомлення та підписує його приватним ключем відправника.
3. Повідомлення додається до нового блоку з хешом попереднього.
4. Блок фіксується в реєстрі та поширюється серед інших вузлів системи.
5. У разі потреби перевірки – будь-який учасник може переконатися в незмінності повідомлення, звіривши хеші та цифрові підписи.

Таким чином, поєднання методів хешування, асиметричного шифрування, цифрового підпису та механізмів блокчейн дозволяє сформувати комплексне рішення, що задовольняє вимоги до сучасних безпечних систем корпоративної комунікації. Обґрунтованість такого вибору підтверджується успішним

застосуванням аналогічних підходів у фінансових технологіях, електронному документообігу та системах ідентифікації особистості.

1.4 Постановка задачі

У результаті виконання аналітичного огляду предметної області та оцінки актуального стану ринку засобів корпоративної комунікації виявлено потребу у створенні програмного забезпечення, яке б забезпечувало фіксацію обміну повідомленнями з гарантованою автентичністю, цілісністю та незмінністю даних.

Для успішної розробки системи необхідно реалізувати такі завдання:

- спроєктувати архітектуру програмного забезпечення, що реалізує визначений функціонал;
- розробити модель структури повідомлень, принципи формування блоків та їх зв’язку у ланцюг;
- реалізувати програмні модулі для надсилання, фіксації та перевірки повідомлень;
- забезпечити формування журналу подій та механізмів верифікації даних;
- здійснити тестування працездатності програмного продукту та оцінити його відповідність сформульованим вимогам.

Постановка зазначених задач дозволяє сформувати цілісний підхід до створення сучасної системи корпоративної взаємодії, яка функціонує у розподіленому середовищі та гарантує безпеку комунікаційних процесів у межах організації.

1.5 Висновки до першого розділу

У першому розділі бакалаврської кваліфікаційної роботи було здійснено комплексний аналіз предметної області корпоративних комунікацій. Проаналізовано сучасний стан систем корпоративної взаємодії, виявлено основні

недоліки централізованих моделей та визначено актуальні проблеми безпеки інформації.

Виконано порівняльний аналіз існуючих аналогів систем корпоративного листування, таких як Microsoft Teams, Slack, Mattermost, Rocket.Chat та Element. Проаналізовано їхні архітектури, функціональні можливості та ступінь забезпечення захисту даних.

Досліджено методи розв'язання задачі захищеної комунікації, зокрема впровадження технологій блокчейн, хешування, цифрового підпису та асиметричного шифрування.

На основі проведеного аналізу сформульовано задачі для подальшої розробки програмного продукту, спрямовані на створення безпечної системи корпоративної комунікації з децентралізованим збереженням повідомлень.

2 РОЗРОБКА СТРУКТУРИ ТА МОДЕЛІ ПРОГРАМНОГО ПРОДУКТУ

2.1 Визначення загальних характеристик системи корпоративної комунікації та удосконалення методів її реалізації за допомогою блокчейн

Проєктоване програмне забезпечення призначене для забезпечення корпоративної комунікації у форматі внутрішньої соціальної мережі. Основне функціональне призначення системи полягає в організації безпечного обміну повідомленнями між працівниками організації, веденні персонального профілю, взаємодії з іншими користувачами, створенні публікацій, участі в обговореннях, а також у підтримці цілісної та прозорої історії листування.

Система реалізується у вигляді вебзастосунку з клієнтсько-серверною архітектурою. Користувацька взаємодія відбувається через браузерний інтерфейс, що динамічно оновлюється залежно від дій користувача. Обробка запитів здійснюється серверною частиною, яка виконує логіку опрацювання повідомлень, керування обліковими записами та роботу з базою даних. Зберігання інформації реалізовано з використанням документно-орієнтованої моделі, що дозволяє гнучко структурувати записи користувачів, повідомлення та пов’язані з ними дії [18;19].

На відміну від традиційних рішень, у запропонованій системі реалізовано механізм збереження повідомлень у структурі блоків, що пов’язані між собою криптографічними хешами. Упровадження такої архітектури забезпечує прозору послідовність усіх дій у системі, що виключає можливість несанкціонованого редагування або видалення повідомлень. Кожна дія користувача, що пов’язана з передачею інформації, фіксується у вигляді транзакції, яка містить хеш вмісту, мітку часу та цифровий підпис відправника.

Структура взаємодії між клієнтською частиною, сервером та підсистемами зберігання даних представлена на рисунку 2.1.

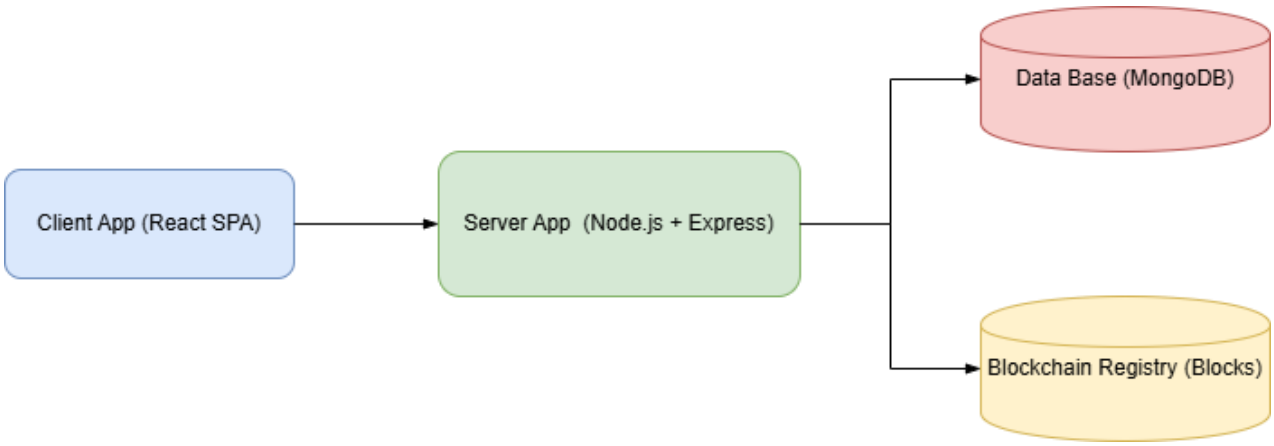


Рисунок 2.1 – Структурна схема загальної взаємодії компонентів системи

Технологія блокчейн у межах системи застосовується як інструмент забезпечення незмінності й автентичності комунікаційного процесу. Усі повідомлення перевіряються на цілісність за допомогою хеш-функцій, а достовірність джерела підтверджується цифровим підписом, що генерується приватним ключем користувача. Верифікація підписів здійснюється за допомогою відкритого ключа, що дозволяє перевірити автентичність без розкриття вмісту.

Такий метод підвищує рівень довіри до кожного елементу системи, забезпечуючи прозору й контрольовану взаємодію.

Запропонована архітектура програмного забезпечення корпоративної системи комунікацій базується на приватному блокчейні та забезпечує збереження автентичності повідомлень без використання централізованого сервера, що дозволяє підвищити швидкодію здійснення комунікацій з підтримкою високого рівня безпеки.

Метод перевірки цілісності повідомлень у розподіленому середовищі, використовує методи криптографічного хешування в комбінації з цифровим підписом, що забезпечує верифікацію авторства без втрати продуктивності.

Також використані методи до розробки безпечних вебзастосунків для внутрішніх комунікацій, які, на відміну від існуючих, використовують механізми перевірки достовірності повідомлень через

Інструментарій реалізації передбачає використання сучасних вебтехнологій для побудови інтерфейсу, а також бібліотек криптографії для підтримки блокчейн-функціоналу. Застосування таких рішень дозволяє адаптувати систему до потреб організацій з високими вимогами до захисту даних, прозорості внутрішньої взаємодії та достовірності обміну повідомленнями.

У результаті реалізації функціоналу очікується створення надійної платформи для внутрішньої корпоративної комунікації, яка, окрім стандартних інструментів взаємодії, надає гарантії незмінності, цілісності та доказовості кожного повідомлення.

2.2 Розробка моделі системи

Архітектура програмного забезпечення проєктованої системи базується на принципах модульності, розділення відповідальності та багаторівневої обробки даних. Система реалізується у вигляді веборієнтованого клієнт-серверного застосунку, в якому функціональні компоненти розподілені між клієнтською частиною, серверною логікою, підсистемою збереження даних та блоком верифікації повідомлень.

Клієнтська частина реалізує засоби інтерактивної взаємодії користувача з системою. Вона відповідає за формування інтерфейсу, відправлення повідомлень, перегляд публікацій, виконання пошуку, зміну профілю та інші дії. Кожна дія користувача призводить до формування HTTP або WebSocket-запиту, який передається на обробку серверу.

Серверна частина системи виконує логіку маршрутизації запитів, авторизації, збереження повідомлень, обробки даних користувачів та генерації блоків для фіксації інформації у внутрішньому ланцюгу блоків. Для кожного повідомлення сервер формує хеш-значення, цифровий підпис на основі відкритого ключа користувача та мітку часу. Зібрані дані додаються до нового

блоку, який зберігається у структурі блокчейн і доступний для перевірки цілісності та автентичності.

Окремим логічним компонентом виступає модуль верифікації. Його призначенням є перевірка достовірності повідомлень за інформацією, що зберігається в блокчейні. Цей модуль порівнює хеш-значення повідомлення із збереженим, перевіряє цифровий підпис та забезпечує користувача інформацією про те, чи було повідомлення змінене або підроблене.

Збереження інформації здійснюється двома способами: структуровані дані користувачів, публікацій і метадані зберігаються у базі даних, організованій за документно-орієнтованою моделлю; зафіксовані блоки повідомлень – у внутрішній структурі блокчейну. Це забезпечує розмежування оперативної та незмінної інформації, а також підвищує надійність системи загалом.

Додатково передбачено реалізацію механізмів безпеки та високої доступності: балансування навантаження між кількома сутностями серверної частини, кешування запитів до часто використовуваних ресурсів, а також шифрування передачі даних за допомогою SSL/TLS. У якості технологічної бази використовуються Node.js із фреймворком Express для обробки HTTP-запитів та WebSocket-підключень, MongoDB для зберігання документів та Redis для тимчасового зберігання сесій і черг повідомлень. Завдяки такому підходу забезпечується горизонтальне масштабування серверної частини, а також можливість розширення шляхом додавання мікросервісів, наприклад, окремого модуля аналітики чи служби нотифікацій.

На рисунку 2.2 представлено структурну UML-діаграму взаємодії компонентів системи корпоративної комунікації. Діаграма деталізує взаємодію між клієнтською частиною, серверною логікою (API Controller, Message Service, Blockchain Manager, User Manager) та підсистемами зберігання даних.

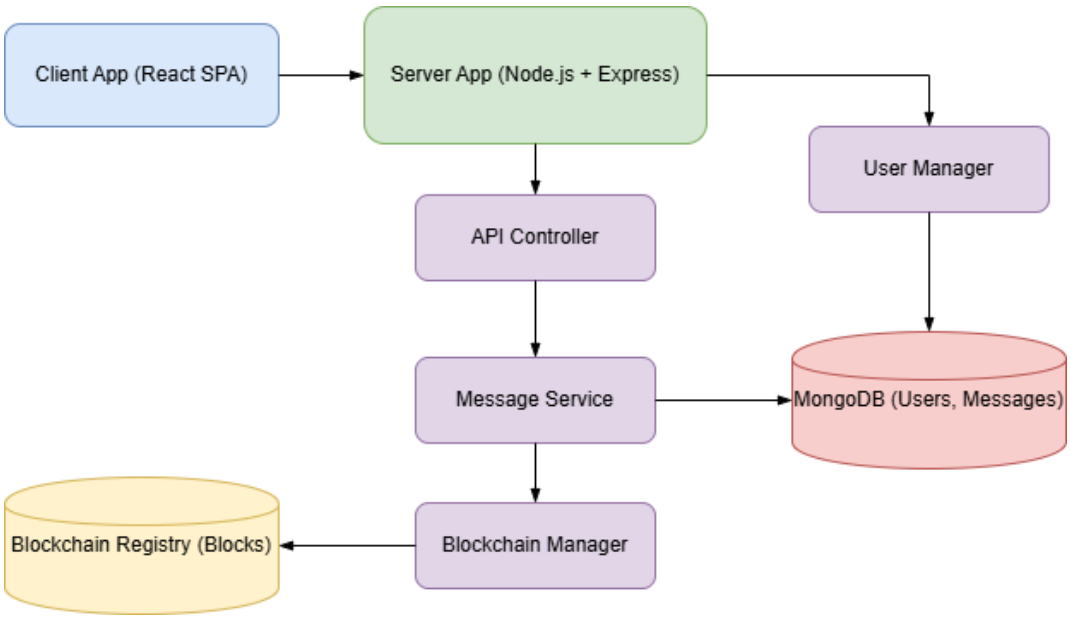


Рисунок 2.2 – Структурна UML-діаграма взаємодії компонентів системи

На діаграмі зображено взаємодію основних компонентів інформаційної системи корпоративної комунікації. Клієнтська частина, реалізована на базі технології React, здійснює обмін даними із серверною частиною за допомогою API-запитів. Серверна частина, побудована на платформі Node.js із використанням фреймворку Express, обробляє запити через спеціалізовані підсистеми: обробки повідомлень (Message Service), управління блокчейном (Blockchain Manager) та керування користувачами (User Manager).

Дані користувачів і повідомлень зберігаються в базі даних MongoDB, що забезпечує оперативне збереження та доступ до змінних даних. Незмінні записи про надсилання повідомлень зберігаються у внутрішньому блокчейн-реєстрі, що гарантує цілісність і автентичність інформації [20; 21].

Взаємодія між компонентами забезпечує узгоджену роботу підсистеми зберігання даних відповідно до вимог безпеки та надійності.

У результаті побудови архітектури визначено основні логічні зв’язки між компонентами системи, що дозволяє ефективно координувати їхню роботу та забезпечити відповідність функціональних вимог умовам корпоративного середовища. Застосована архітектура сприяє надійності та безпеці програмного рішення, а також забезпечує умови для його подальшого масштабування.

2.3 Аналіз засобів зберігання даних

Одним з ключових компонентів будь-якого веборієнтованого застосунку є система збереження даних, яка повинна забезпечувати надійне, гнучке та масштабоване управління інформацією. Для проєктованої системи корпоративної комунікації, яка передбачає інтенсивну взаємодію між користувачами, обробку великої кількості повідомлень, динамічну зміну стану об’єктів та збереження незмінних записів у блокчейні, необхідно застосовувати засоби зберігання, що дозволяють ефективно працювати як із структурованими, так і з незмінними даними.

З огляду на специфіку застосунку було прийнято рішення використати документно-орієнтовану базу даних MongoDB, яка реалізує зберігання у форматі BSON, сумісному з JSON. Такий підхід дає змогу зберігати кожен об’єкт у вигляді документа, що містить гнучку структуру полів, вкладених об’єктів та масивів, а також дозволяє уникнути складного нормалізованого поділу таблиць, як це властиво реляційним базам даних.

Застосування MongoDB дає змогу ефективно та швидко реалізувати зберігання даних про користувачів, повідомлення, публікації, файли, коментарі, а також метаданні блоків, що створюються для фіксації подій у блокчейні. Гнучка схема дозволяє адаптувати структуру документів до особливостей даних, що змінюються в процесі розробки та вдосконалення системи, а також зручно масштабувати її.

Для зберігання незмінних записів, які містять інформацію про транзакції, повідомлення, хеші, мітки часу та цифрові підписи використовується структура внутрішнього блокчейн-реєстру, яка функціонує як послідовність пов’язаних записів, де кожен блок містить посилання на хеш попереднього. Така структура зберігається окремо від основної бази даних і є доступною лише для читання та доповнення.

Внутрішнє зберігання блоків реалізовано у вигляді окремої файлової структури або колекції документів у MongoDB, із застосуванням обмеження на редагування існуючих записів. Кожен блок містить поля, які включають:

- хеш попереднього блоку;
- мітку часу;
- зашифроване повідомлення або хеш повідомлення;
- публічний ключ відправника;
- цифровий підпис;
- ідентифікатор транзакції.

Застосування двох паралельних моделей зберігання (оперативної бази даних та структури блоків) дозволяє забезпечити одночасно як продуктивність при обробці запитів, так і достовірність і незмінність збереженої інформації. Такий підхід також дає змогу розмежувати функції обробки даних у реальному часі та перевірки їх цілісності в рамках окремих модулів системи та системи в цілому.

Структуру бази даних зображено на рисунку 2.3.

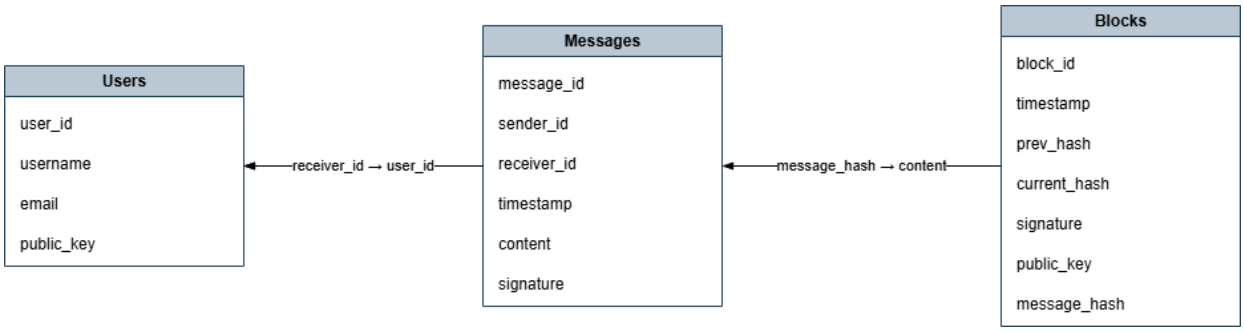


Рисунок 2.3 – Структура даних бази даних

На діаграмі представлено структуру даних у підсистемі зберігання інформації. Колекція Users містить основні відомості про користувачів, такі як ід користувача, його ім’я, елекронну пошту та публічний ключ, колекція Messages зберігає дані про текстові повідомлення разом із підписами та часовими мітками, а колекція Blocks фіксує незмінні записи повідомлень у блокчейн-реєстрі. Така система забезпечує високий рівень надійності завдяки тому, що

зв'язки між колекціями відображають відношення відправника й одержувача до користувачів та асоціацію повідомлень із блоками.

У результаті прийнятого рішення засоби зберігання повністю задовольняють функціональні вимоги до системи та дозволяють реалізувати логіку комунікації, фіксації та перевірки повідомлень у розподіленому інформаційному середовищі.

2.4 Аналіз технологічних засобів реалізації

Реалізація проєктованої системи корпоративної комунікації здійснюється з використанням сучасних технологій веброзробки, що забезпечують ефективну побудову клієнт-серверного застосунку з інтегрованим блокчейн-модулем. Під час вибору засобів розробки враховувалися такі фактори, як доступність, продуктивність, підтримка криптографічних операцій, зручність інтеграції та активна підтримка спільнотою розробників.

Перелік основних технологій системи наведено в таблиці 2.1.

Таблиця 2.1 – Перелік основних технологій та їх призначення в системі

Технологія	Призначення
Node.js	Реалізація серверної частини, обробка бізнес-логіки та API-запитів
Express.js	Побудова маршрутизації та обробка HTTP-запитів на сервері
React	Розробка клієнтського інтерфейсу у вигляді односторінкового застосунку (SPA)
MongoDB	Зберігання даних користувачів та повідомлень у документно-орієнтованому форматі
bcrypt/crypto	Хешування даних та створення цифрових підписів для забезпечення безпеки
Mongoose	Моделювання даних MongoDB у Node.js та забезпечення зв'язку між сервером і базою даних

Клієнтську частину розроблено за допомогою бібліотеки React, яка є однією з найпоширеніших у сфері фронтенд-розробки. React дозволяє створювати динамічні інтерфейси користувача на основі компонувального

підходу, забезпечує ефективне оновлення вмісту за рахунок використання віртуального DOM та має багатий екосистемний набір додаткових інструментів. Застосування React дало змогу реалізувати адаптивний, швидкий і масштабований інтерфейс системи.

Серверна частина реалізована з використанням середовища виконання Node.js, що дозволяє запускати JavaScript на боці сервера. Node.js забезпечує обробку великої кількості одночасних з'єднань завдяки неблокуючій моделі введення-виведення. У поєднанні з фреймворком Express.js реалізовано маршрутизацію запитів, обробку даних, управління сеансами користувачів і взаємодію з базою даних та блокчейн-модулем [22-24].

У якості засобу збереження даних використовується MongoDB – документ-орієнтована нереляційна база даних. MongoDB зберігає дані у форматі BSON, що дозволяє ефективно працювати з вкладеними структурами даних і забезпечує гнучкість у побудові моделі. Взаємодія з базою даних здійснюється за допомогою бібліотеки Mongoose, яка реалізує рівень абстракції для створення схем і валідації даних [25].

Для реалізації криптографічного функціоналу застосовано такі бібліотеки:

- CryptoJS – для генерації хеш-функцій (SHA-256), які використовуються під час формування блоків у блокчейні;
- Elliptic – для реалізації цифрових підписів на основі еліптичних кривих (ECDSA);
- jsonwebtoken (JWT) – для підпису токенів авторизації та перевірки автентичності запитів;
- uuid – для створення унікальних ідентифікаторів повідомлень і транзакцій.

Інструменти розробки включають середовище WebStorm, яке забезпечує зручну інтеграцію з Node.js та React, має потужні засоби налагодження, контроль версій та автодоповнення коду. Контроль версій проєкту здійснюється за допомогою системи Git.

Такий вибір технологій дозволяє не лише реалізувати заявлену функціональність системи, а й забезпечити її надійність, масштабованість і можливість подальшого розвитку. Усі використані засоби є відкритими та активно підтримуються спільнотами розробників, що сприяє довготривалій підтримці проєкту створення програмного продукту.

2.5 Проєктування структури збереження даних у блокчейні

Для забезпечення незмінності, прозорості та доказовості історії повідомлень у системі корпоративної комунікації було реалізовано механізм збереження даних у вигляді внутрішньої блокчейн-структури. Така структура забезпечує формування ланцюга блоків, кожен з яких містить зафіксовану дію користувача – зокрема надсилання повідомлення.

Кожен блок містить обмежену кількість даних і зв’язується з попереднім через хеш, що генерується з його вмісту. Таким чином, будь-яке втручання в дані блоку автоматично призводить до зміни хешу, що порушує цілісність усього ланцюга. Це дозволяє виявити навіть незначні спроби фальсифікації історії листування [26].

Формат одного блоку у системі представлено у вигляді фіксованої структури, яка включає такі поля:

- timestamp – мітка часу створення блоку;
- senderId – ідентифікатор користувача-відправника;
- receiverId – ідентифікатор отримувача повідомлення (або групи);
- messageHash – SHA-256 хеш повідомлення;
- previousHash – хеш попереднього блоку;
- digitalSignature – цифровий підпис користувача, згенерований на основі приватного ключа;
- publicKey – відкритий ключ користувача, необхідний для верифікації підпису;
- blockId – унікальний ідентифікатор блоку.

Формування нового блоку виконується на стороні сервера після надходження повідомлення від клієнта. Спершу перевіряється коректність цифрового підпису, що додає додатковий рівень автентифікації та гарантує походження даних від заявленого користувача. Далі вміст повідомлення хешується алгоритмом SHA-256, що дозволяє одержати унікальне фіксованої довжини значення для кожного окремого повідомлення. До отриманого хешу додається мітка часу (timestamp), яка забезпечує точне хронологічне відстеження подій у системі. Після цього формується структура блоку, у яку входить: власне хеш повідомлення, цифровий підпис користувача, мітка часу, а також посилання на хеш попереднього блоку (previous hash). Таким чином, новий блок містить як свої власні метадані, так і вказівку на попередню одиницю ланцюга. У результаті формується послідовний ланцюг, у якому жоден блок не може бути змінений без порушення цілісності всієї структури та виявлення несанкціонованих модифікацій. Завдяки такому підходу кожна спроба несанкціонованого втручання призведе до невідповідності хеш-значень, що робить можливим оперативне виявлення та запобігання будь-яким спробам підробки історії повідомлень.

У блокчейн-реєстрі кожен новий блок зберігає у своїй структурі цифровий підпис, власний хеш та хеш попереднього блока. Таким чином забезпечується ланцюгова залежність, що гарантує незмінність і достовірність переданої інформації, а також можливість проведення незалежного аудиту. Побудову взаємопов'язаного ланцюга блоків наведено на рисунку 2.5.

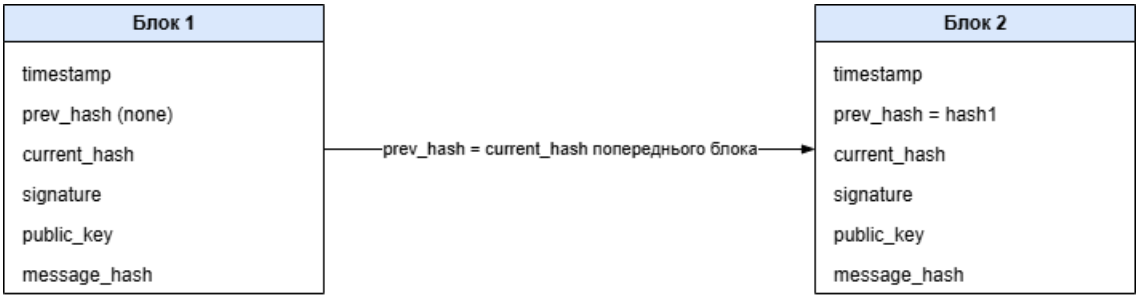


Рисунок 2.5 – Схема побудови ланцюга блоків

Процес додавання нового блока до блокчейн-реєстру передбачає кілька основних етапів: отримання даних для збереження, формування блока з відповідними полями, обчислення хешу, встановлення посилання на попередній блок через поле prev_hash, підписування блока приватним ключем користувача та збереження блока у ланцюзі. Алгоритм додавання нового блока у блокчейн-реєстр системи наведено на рисунку 2.6.

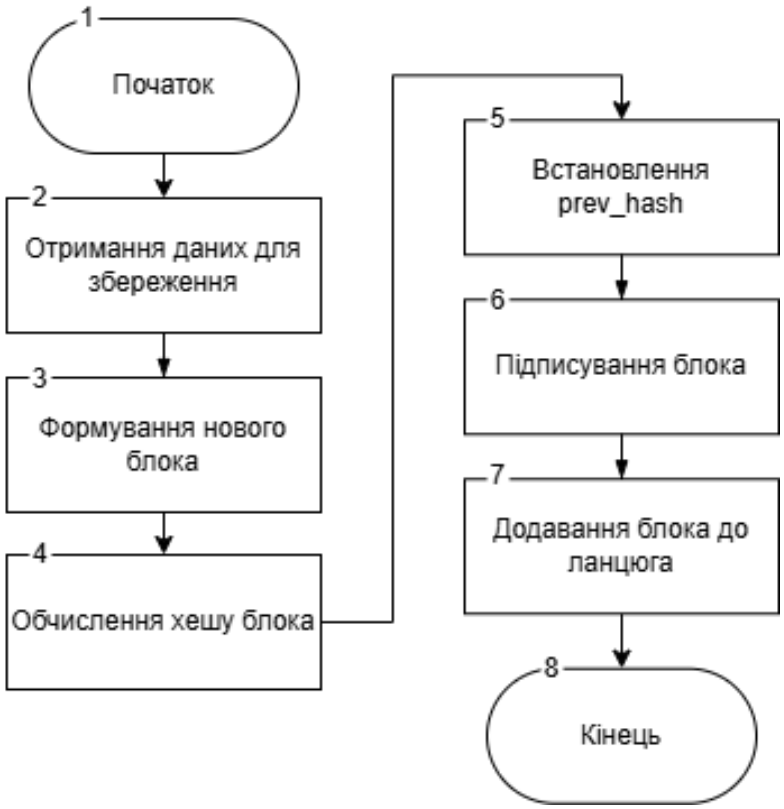


Рисунок 2.6 – Алгоритм додавання нового блока

Така структура є внутрішнім логічним реєстром, не призначеним для широкого зовнішнього розповсюдження, але забезпечує доказову базу для відновлення історії взаємодій у системі. Застосування даного підходу дозволяє поєднати продуктивність централізованої обробки з незмінністю децентралізованого зберігання ключових дій.

2.6 Висновки до другого розділу

У другому розділі роботи здійснено розроблення структури та моделі програмного продукту. Визначено загальні характеристики системи, її призначення та основні функціональні можливості.

Створено модель архітектури системи, яка передбачає розподіл функціональних обов'язків між клієнтською частиною, серверною логікою, підсистемою збереження даних та модулем верифікації повідомлень.

Обґрунтовано вибір технологій реалізації програмного забезпечення, зокрема застосування MongoDB для збереження даних, використання Node.js та Express.js для серверної частини, React для клієнтської частини та відповідних криптографічних бібліотек для забезпечення безпеки даних.

Формалізовано структуру блокчейну для фіксації повідомлень та забезпечення їх незмінності й автентичності.

Сформовані технічні рішення дозволяють перейти до реалізації програмних модулів, що буде здійснено в наступному розділі.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Програмне забезпечення реалізовано з використанням мови програмування JavaScript та сучасних засобів створення клієнт-серверних застосунків. Структура проєкту сформована таким чином, щоб забезпечити розділення відповідальності між окремими модулями, модифікованість та підтримуваність. Основу розробки становить використання фреймворку React для реалізації клієнтської частини та середовища Node.js для побудови серверної логіки.

Клієнтську частину реалізовано у вигляді односторінкового застосунку, в межах якого визначено набір функціональних компонентів, що відповідають за відображення стрічки публікацій, сторінки повідомлень, профілю користувача, навігації та обробки подій. Застосування бібліотеки React дозволяє реалізувати оновлення інтерфейсу без повного перезавантаження сторінки та забезпечує розділення вмісту на логічні компоненти.

У клієнтському застосунку використано механізм маршрутизації, що дає змогу перемикатися між основними розділами (головна сторінка, повідомлення, профіль) за допомогою адресного рядка браузера. Стан користувача зберігається в глобальному контексті та використовується для управління інтерфейсом. Взаємодія з сервером реалізується через HTTP-запити за допомогою бібліотеки fetch.

Серверну частину побудовано на базі фреймворку Express.js. Вона обробляє вхідні запити, виконує авторизацію, верифікацію даних, взаємодіє з базою даних і виконує формування блоків у структурі блокчейну. Реалізовано основні маршрути для обробки публікацій, повідомлень, авторизації користувачів, а також окремі ендпоінти для верифікації повідомлень.

У структурі проєкту створено окремі папки для моделей даних, контролерів, служб взаємодії з базою даних та логіки блокчейн-блоків. Це дозволяє дотримуватись принципів інкапсуляції й повторного використання коду. Базу даних реалізовано на основі MongoDB, що забезпечує зберігання інформації у вигляді документів у форматі JSON. Для збереження блоків використано окрему колекцію, яка містить послідовність об’єктів, що відповідають структурі блоку.

У кожному блоці зберігається інформація про час відправлення повідомлення, хеш повідомлення, цифровий підпис, публічний ключ, а також хеш попереднього блоку. Така структура дозволяє формувати незмінну послідовність транзакцій, де кожен блок підтверджує цілісність попереднього та забезпечує незмінність даних.

Формування цифрового підпису повідомлення реалізовано з використанням бібліотеки elliptic. Для цього генерується пара ключів (приватний і публічний), де перший використовується для підпису повідомлення, а другий – для перевірки достовірності вмісту. У процесі верифікації серверна частина використовує відкритий ключ для того, щоб підтвердити відповідність повідомлення підпису, що гарантує його цілісність.

Кожне повідомлення після відправки обробляється наступним чином: створюється хеш повідомлення, генерується підпис, формується об’єкт блоку, який додається до локальної структури блокчейну. Після цього блок зберігається у відповідній колекції бази даних. Цей підхід дозволяє не лише відтворити історію взаємодії, а й перевірити кожну дію на достовірність.

Застосування подібної архітектури дозволило реалізувати систему, що поєднує функціональність соціальної мережі для корпоративного середовища з високим рівнем безпеки повідомлень. Усі дії користувачів фіксуються у блоках, що гарантує прозорість та незмінність історії комунікацій.

3.2 Розробка модуля обміну повідомленнями з використанням блокчейну

Модуль обміну повідомленнями є одним із центральних функціональних компонентів системи корпоративної комунікації. Його реалізацію виконано з урахуванням необхідності збереження кожного повідомлення у вигляді транзакції, яка фіксується у внутрішньому блокчейн-реєстрі. Такий підхід дозволяє забезпечити автентичність, незмінність і цілісність історії листування між користувачами.

Відправка повідомлення ініціюється з клієнтської частини, де користувач заповнює текстове поле. Клієнтський застосунок, реалізований у середовищі React, формує запит до серверного API з вмістом повідомлення та маркером авторизації. Передача здійснюється через HTTP-запит типу POST або WebSocket, залежно від реалізації з'єднання в реальному часі.

На серверній стороні після отримання повідомлення воно підлягає попередній обробці. У разі успішної авторизації користувача сервер здійснює хешування повідомлення за допомогою алгоритму SHA-256, генерує цифровий підпис повідомлення та створює блок відповідно до визначеної структури. Блок включає хеш попереднього блоку, мітку часу, відкритий ключ користувача, хеш самого повідомлення та створений підпис.

У системі реалізовано два паралельні процеси: один – збереження повідомлення у колекції повідомлень для оперативного доступу, інший – створення блоку у внутрішній структурі блокчейну.

Для створення ключів і підписів використано криптографічну бібліотеку elliptic. Після збереження блоку у базі даних MongoDB здійснюється підтвердження успішної транзакції. Оскільки кожне повідомлення має відповідний запис у блокчейні, будь-яка модифікація або спроба підробки змісту призведе до порушення цілісності ланцюга, що може бути легко виявлено під час верифікації.

Для перевірки достовірності повідомлення реалізовано окрему функцію, яка приймає відкритий ключ, хеш повідомлення та підпис, після чого засобами

бібліотеки перевіряє коректність підпису. Ця перевірка може бути виконана як автоматично при завантаженні листування, так і вручну, у разі необхідності аудиту.

Схему процесу обробки повідомлення з формуванням блоку наведено на рисунку 3.1

Рисунок 3.1 – Схема формування повідомлення та блоку у внутрішньому блокчейні

Таким чином, модуль обміну повідомленнями забезпечує не лише передачу даних у режимі реального часу, а й їх фіксацію з гарантією незмінності.

Це дозволяє використовувати систему для важливої корпоративної комунікації, де кожна дія може бути підтверджена незалежно.

3.3 Розробка модуля перевірки достовірності повідомлень

Модуль перевірки достовірності повідомлень призначений для забезпечення механізму контролю цілісності кожного повідомлення, яке зберігається у внутрішньому блокчейні системи. Його робота ґрунтується на використанні пари криптографічних ключів та алгоритму верифікації цифрового підпису, що гарантує відповідність повідомлення його джерелу.

Процес перевірки ініціюється після надходження запиту від клієнта на перегляд повідомлення або під час здійснення аудиту. Сервер отримує хеш повідомлення, цифровий підпис та публічний ключ. На основі цих даних виконується математична перевірка відповідності підпису вхідному хешу. У разі успішної перевірки повідомлення визнається достовірним, у протилежному випадку – визначається як модифіковане або підроблене.

У цьому прикладі використовується алгоритм ECDSA, реалізований бібліотекою elliptic. Для верифікації формується хеш отриманого повідомлення, після чого з використанням публічного ключа виконується перевірка, чи відповідає підпис даному хешу. У випадку зміни навіть одного символу в повідомленні перевірка завершиться невдачею.

Окрім базової перевірки підпису, також може виконуватися порівняння хешу повідомлення з відповідним полем у блоці. Це дозволяє впевнитися, що вміст повідомлення не було змінено після його фіксації у структурі блокчейну. Така подвійна перевірка дозволяє не тільки підтвердити справжність повідомлення, а й гарантувати, що його зміст не зазнав змін під час передачі або зберігання.

Результат перевірки використовується для інформування користувача про статус повідомлення. У разі успішної верифікації повідомлення в інтерфейсі

відображається як надійне, тоді як у разі невідповідності – система може попередити про можливу зміну або втрату достовірності.

На рисунку 3.2 наведено схему перевірки достовірності повідомлення за допомогою цифрового підпису та публічного ключа.

Рисунок 3.2 – Схема перевірки достовірності повідомлення на основі цифрового підпису

Після отримання повідомлення здійснюється обчислення його хешу, а потім верифікація цифрового підпису за допомогою відкритого ключа відправника. Успішна перевірка підтверджує справжність даних.

З технічної точки зору, перевірка справжності є ключовим елементом інформаційної безпеки, що дозволяє запобігти компрометації системи. Особливо це актуально в умовах багатокористувацького середовища, де кожен учасник

може виконувати важливі дії. Механізм верифікації дозволяє не лише забезпечити безпеку, а й служить доказовою базою при виникненні конфліктів, внутрішніх розслідувань або потребі перевірки дій конкретного користувача.

Таким чином, реалізований механізм дозволяє захистити систему від спроб підробки повідомлень, забезпечує прозору історію взаємодії та підвищує довіру до використання у корпоративному середовищі.

3.4 Розробка модуля збереження блоків у базі даних

Збереження блоків є ключовим етапом функціонування підсистеми. Після формування блоку на основі повідомлення, цифрового підпису та метаданих, він підлягає збереженню у внутрішній структурі бази даних. У системі реалізовано зберігання блоків у вигляді документів у базі MongoDB, що дозволяє забезпечити гнучке структурування та швидкий доступ до даних.

Кожен блок, який створюється після надсилання повідомлення, зберігається у вигляді окремого документа з фіксованою структурою полів. До блоку входять такі елементи: хеш попереднього блоку, хеш повідомлення, цифровий підпис, публічний ключ, мітка часу, індекс у ланцюгу та сам хеш поточного блоку. Така структура дозволяє побудувати повноцінний ланцюг, який може бути перевірений на цілісність.

Зберігання блоків реалізовано у вигляді окремої колекції у базі даних під назвою blocks. Колекція не призначена для прямої взаємодії користувача та використовується виключно для внутрішньої перевірки та фіксації транзакцій.

Після створення блоку в оперативній пам’яті об’єкт зберігається до бази даних через метод .save(), що гарантує незмінність збереженої інформації. Структура документа не передбачає механізмів оновлення або видалення – доступ дозволений лише на читання та додавання нових записів. Таким чином, досягається логічна незмінність історії транзакцій.

Схема даного алгоритму зображена на рисунку 3.3.

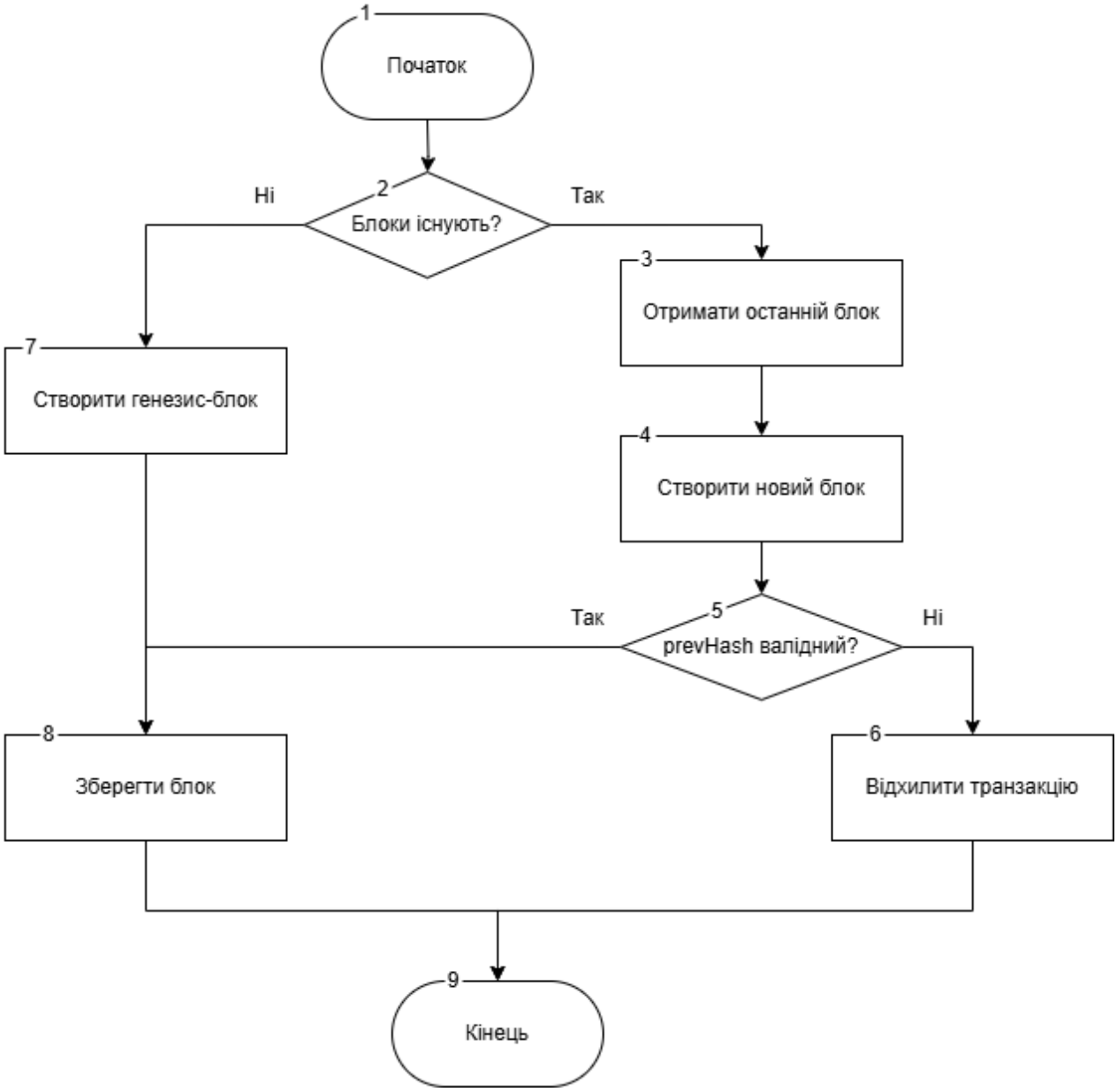


Рисунок 3.3 – Схема алгоритму створення нового блоку

Збереження блоків супроводжується попередньою перевіркою цілісності попереднього блоку. У разі відсутності очікуваного хешу або дублювання індексу транзакція скасовується. Це дозволяє уникнути розгалуження ланцюга або несанкціонованого втручання у структуру.

Крім цього, під час запуску системи здійснюється ініціалізація ланцюга блоків. У випадку повної відсутності записів створюється "генезис-блок" з нульовими значеннями полів, що слугує базовою точкою початку блокчейну.

З метою підтримки швидкого доступу до останнього блоку реалізовано механізм вибірки останнього запису за індексом. Це дозволяє уникнути повного перегляду ланцюга при кожному додаванні нової транзакції.

Таким чином, реалізований модуль забезпечує надійне та структуроване збереження блоків, виключаючи можливість модифікації збережених даних, що є важливим для забезпечення принципу незмінності в системі корпоративного обміну повідомленнями.

3.5 Розробка модуля генерації цифрового підпису

Генерація цифрового підпису є невід’ємною складовою захисту повідомлень у системі корпоративної комунікації. Цей підпис дозволяє ідентифікувати автора повідомлення та гарантує, що вміст не був змінений після його створення. У системі реалізовано алгоритм ECDSA (Elliptic Curve Digital Signature Algorithm), який забезпечує високий рівень криптографічної стійкості навіть при відносно коротких ключах.

Для кращого розуміння логіки створення підпису, на рисунку 3.4 подано схему алгоритму генерації підпису за ECDSA.

Алгоритм ECDSA заснований на еліптичних кривих і дозволяє створити пару ключів – приватний і публічний. Приватний ключ використовується для підписування повідомлень, а публічний – для перевірки підпису. Генерація ключів здійснюється автоматично.

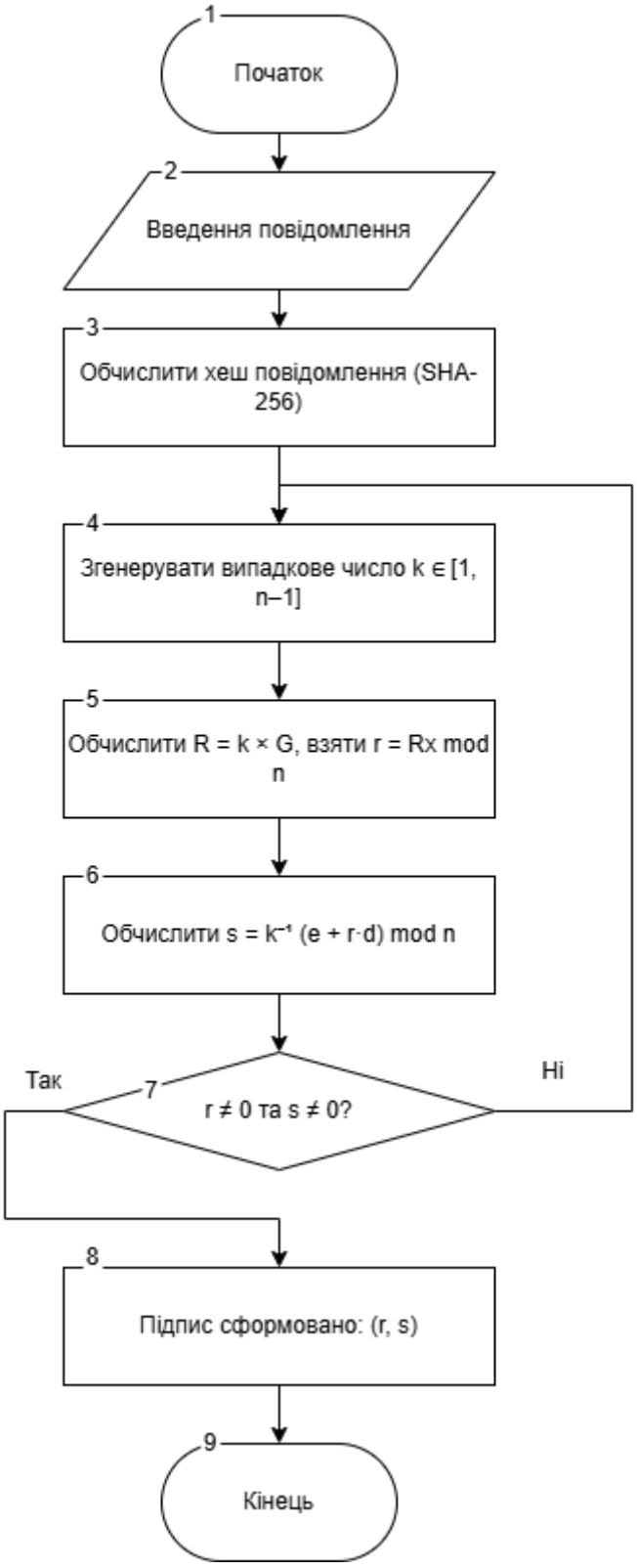


Рисунок 3.4 – Схема генерації цифрового підпису ECDSA

Після створення ключів, для кожного повідомлення обчислюється хеш (SHA-256), який підписується за допомогою приватного ключа. У результаті

формується криптографічний підпис, який додається до повідомлення і зберігається разом із ним у блоці блокчейну.

Процес генерації підпису включає кілька етапів: генерацію випадкового значення k, обчислення координат точки $R = k \times G$, формування значення r як $Rx \bmod n$, та обчислення $s = k^{-1} (e + r \cdot d) \bmod n$, де d – приватний ключ, e – хеш повідомлення. Перевірка умов $r \neq 0$ та $s \neq 0$ є обов'язковою для прийняття підпису.

Ця схема відображає послідовність усіх необхідних операцій, починаючи від введення повідомлення до завершення процедури створення підпису. Вона дозволяє візуально представити логіку роботи модуля та продемонструвати роль кожного етапу.

Алгоритм ECDSA є стандартом у багатьох криптографічних протоколах, включаючи Bitcoin та інші блокчейн-проєкти, що підтверджує його надійність та ефективність. У межах даної системи він дозволяє забезпечити надійну фіксацію авторства повідомлень без необхідності зберігати зміст повідомлення у відкритому вигляді.

Таким чином, реалізований модуль створення цифрового підпису відповідає сучасним вимогам до безпеки комунікаційних систем, дозволяє підтверджувати достовірність повідомлень та забезпечує правову значущість кожної дії в системі.

3.6 Розробка модуля перевірки цілісності блокчейну

Цілісність блокчейну є одним з основних механізмів забезпечення довіри до інформаційної системи, у якій застосовується розподілене зберігання повідомлень. Порушення послідовності блоків або модифікація будь-якого елемента блоку неминуче призводить до втрати валідності всієї ланки. Саме тому у розробленому рішенні реалізовано окремий модуль, відповідальний за перевірку цілісності блокчейну.

Для кращого розуміння роботи модуля перевірки, на рисунку 3.5 представлено схему алгоритму перевірки цілісності блокчейну

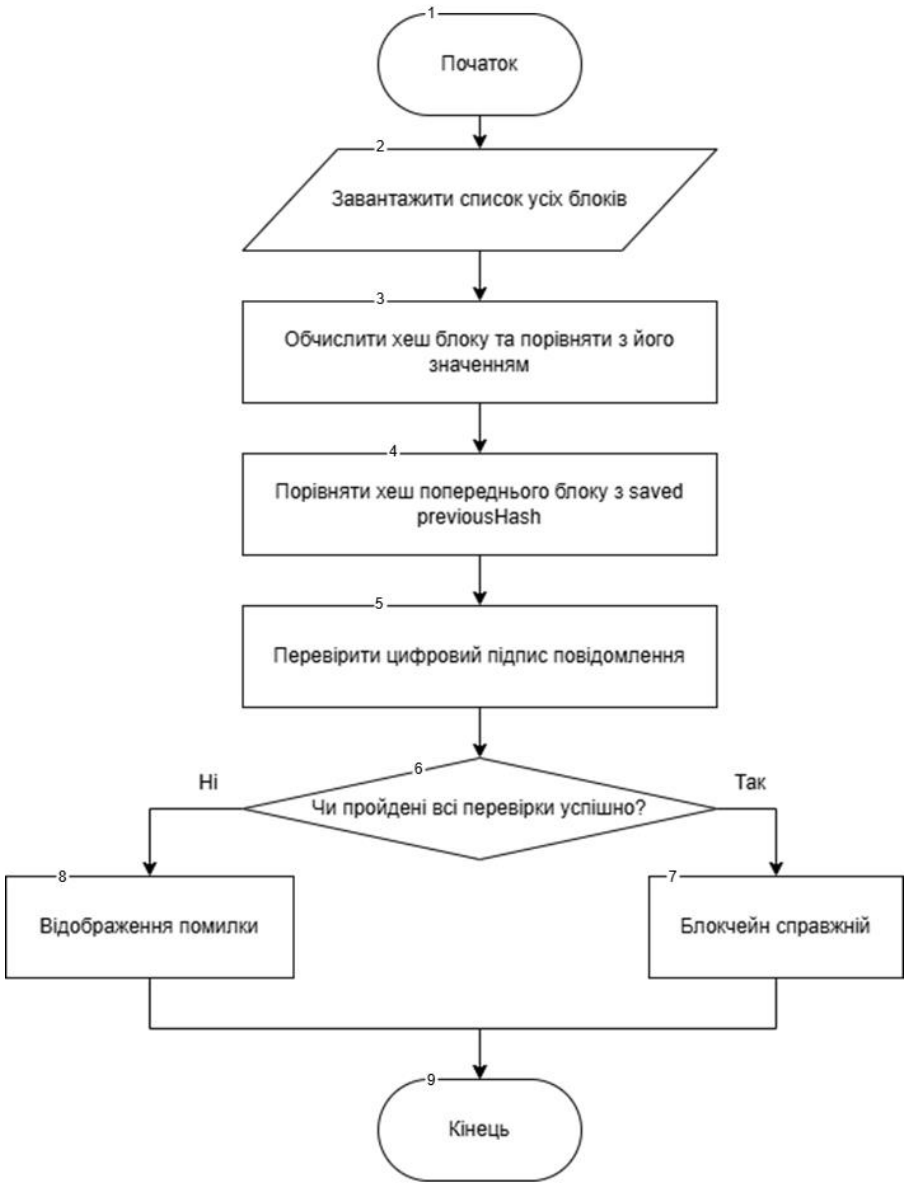


Рисунок 3.5 – Схема перевірки цілісності блокчейну

Вона ілюструє логіку перевірки кожного блока, перевірку цифрового підпису та повернення результату про справжність або зміну даних.

Основне призначення модуля полягає в тому, щоб проходити всі блоки послідовно, починаючи з другого (оскільки перший – генезис-блок), і для кожного з них перевіряти:

- відповідність поля previousHash хешу попереднього блоку;

- правильність обчисленого хешу на основі вмісту блоку;
- дійсність цифрового підпису повідомлення, що входить до блоку.

Кожна з перевірок відповідає критеріям. Порушення одного з критеріїв вважається ознакою втрати цілісності та потенційного несанкціонованого втручання. Застосування цифрового підпису унеможливлює модифікацію вмісту без відома власника приватного ключа, а наявність хеш-зв'язків між блоками гарантує незмінність структури.

Усі блоки зчитуються з бази даних і зберігаються у вигляді масиву. Далі виконується перевірка. У разі виявлення порушень система повертає логічне значення або виводить відповідне повідомлення для адміністратора.

У завершальному етапі перевірки результат логічного аналізу використовується для повідомлення користувача або адміністратора. У разі виявлення порушень система може блокувати певний функціонал, попереджати про необхідність аудиту або ініціювати архівування пошкоджених даних.

Таким чином, реалізований модуль перевірки цілісності є важливим елементом архітектури системи, що гарантує її стійкість до зовнішніх і внутрішніх порушень, підтримує достовірність повідомлень та забезпечує відповідність вимогам безпечного обміну інформацією.

3.7 Розробка інтерфейсу користувача

Інтерфейс користувача є ключовим елементом взаємодії з програмним забезпеченням і визначає зручність використання системи. Під час проєктування інтерфейсу враховано вимоги до ергономіки, логічної послідовності дій, візуальної простоти та адаптивності до різних типів пристроїв.

Користувацький інтерфейс реалізовано у вигляді односторінкового застосунку, що забезпечує динамічне оновлення вмісту без перезавантаження сторінки. Загальна структура включає навігаційну панель, стрічку публікацій, розділ для особистих повідомлень, а також інтерактивні елементи для пошуку та фільтрації контенту.

Головна сторінка вебзастосунку є центральним елементом навігації у системі. Вона надає користувачу доступ до основних функцій, таких як перегляд новин, взаємодія з контактами та управління профілем. Інтерфейс організовано з урахуванням потреб корпоративного середовища, що передбачає чіткий розподіл функціональних зон для оптимізації користувацького досвіду.

На рисунку 3.6 зображено інтерфейс головної сторінки, що відображає основну інформацію користувача, список контактів, навігаційне меню та стрічку новин. Інтерфейс побудовано з урахуванням логіки корпоративного середовища – з чітким поділом на функціональні зони та збереженням візуального балансу.

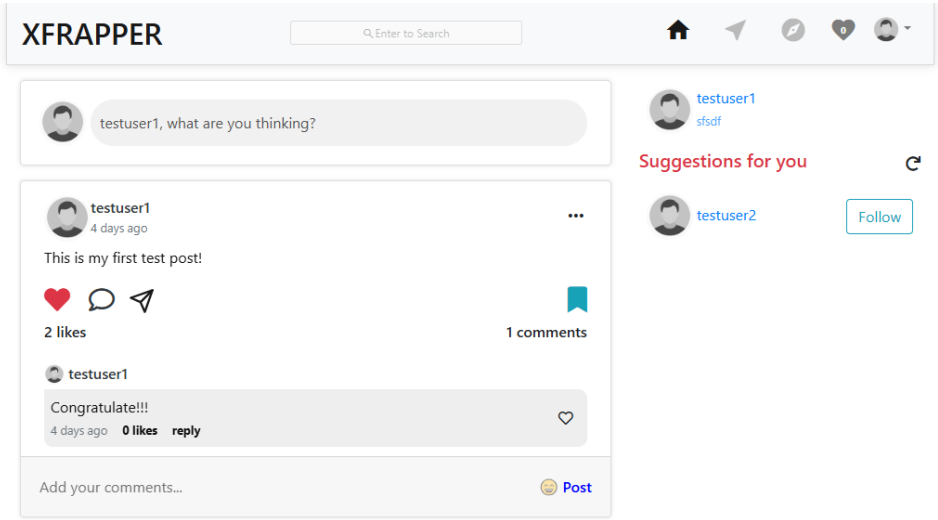


Рисунок 3.6 – Інтерфейс головної сторінки вебзастосунку

Сторінка повідомлень реалізує механізм обміну текстовими повідомленнями між користувачами в режимі діалогу. В інтерфейсі передбачено область перегляду історії листування, поле введення нового повідомлення та панель керування. Така структура забезпечує інтуїтивно зрозумілу взаємодію та дозволяє легко відстежувати стан комунікації.

На рисунку 3.7 наведено приклад інтерфейсу сторінки повідомлень.

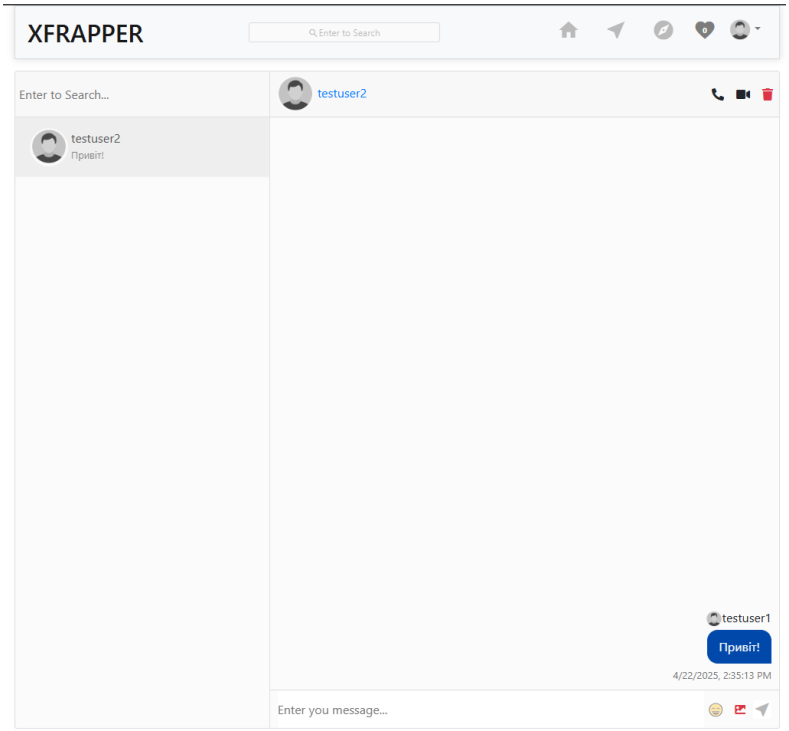


Рисунок 3.7 – Інтерфейс сторінки повідомлень

Візуальне оформлення інтерфейсу витримано в нейтральних кольорах з акцентами на ключових елементах. Застосовано сучасні принципи верстки, що забезпечують гармонійне поєднання змістовного блоку та допоміжних функцій.

Таким чином, побудований інтерфейс користувача відповідає вимогам до корпоративних систем, забезпечує зручність навігації, простоту введення даних та ефективну роботу з повідомленнями.

3.8 Висновки до третього розділу

У межах третього розділу проведено реалізацію ключових модулів програмного забезпечення, призначеного для забезпечення захищеного обміну повідомленнями в корпоративному середовищі. Було реалізовано клієнт-серверну архітектуру, у якій модуль обміну повідомленнями дозволяє здійснювати передавання інформації між користувачами з подальшим фіксуванням у структурі блоків. Кожне повідомлення підлягає попередньому

хешуванню та цифровому підпису, що гарантує достовірність джерела та неможливість зміни вмісту.

Збереження повідомлень та блоків здійснюється з використанням бази даних MongoDB. Для цього створено окрему колекцію, у якій зберігаються блоки з інформацією про попередні хеші, підписи та хронологічні мітки. Це забезпечує послідовність і незмінність усіх дій, зафіксованих у системі.

Особливу увагу приділено модулю перевірки достовірності повідомлень, у якому реалізовано перевірку хешу повідомлення та валідацію цифрового підпису. Завдяки цьому кожне повідомлення можна незалежно перевірити, не маючи доступу до закритих ключів.

Крім цього, створено модуль перевірки цілісності блокчейну, який дозволяє автоматично виявляти будь-які спроби зміни в історії повідомлень. Реалізована схема проходить кожен блок, порівнює попередні хеші та перевіряє підписи, формуючи загальний висновок щодо стану всієї структури.

Використання алгоритму ECDSA дало змогу досягти високого рівня захисту при збереженні продуктивності системи. Усі реалізовані елементи були побудовані з урахуванням вимог безпеки, масштабованості та адаптивності.

Розроблено концепцію інтерфейсу користувача відповідно до вимог зручності використання та ергономіки.

Таким чином, результати реалізації засвідчили працездатність концепції, забезпечили надійність обміну повідомленнями та підтвердили доцільність застосування блокчейн-технологій для підвищення інформаційної безпеки у межах вебсистеми корпоративних комунікацій.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування розробленої системи проводиться з метою перевірки її працездатності, відповідності вимогам до функціонування, а також виявлення можливих помилок у реалізації окремих модулів. Основну увагу зосереджено на коректності реалізації цифрового підпису, обробці повідомлень, формуванні блоків, підтримці зв’язків між ними та захисті даних від фальсифікації.

У процесі тестування використано ручні та автоматизовані підходи. Ручні методи включали перевірку результатів роботи алгоритмів безпосередньо у середовищі Node.js з виведенням інформації у консоль. Автоматизовані перевірки здійснювалися шляхом моделювання типових сценаріїв використання системи через браузерний інтерфейс користувача, а також за допомогою юніт-тестів.

У першу чергу перевірено модуль обробки повідомлень. Надсилання повідомлення формує його хеш. Для підтвердження цілісності реалізації хеш-функції обчислено контрольні значення для кількох вхідних повідомлень. Було встановлено, що при зміні хоча б одного символу у тексті повідомлення формується зовсім інший хеш, що підтверджує криптографічну стійкість та незворотність обраного алгоритму.

Перед безпосередньою перевіркою хеш-функцій було протестовано коректність генерації та верифікації цифрових підписів. Створено кілька пар ключів для різних користувачів: при підписі одного й того самого хешу за допомогою різних приватних ключів отримано різні сигнатури, а при верифікації кожна відповідала правильному публічному ключу. Також перевірено, що жодна сигнатура не проходить верифікацію за іншим хешем або іншим ключем, що підтверджує надійність механізму підпису та автентичність повідомлень (див. рисунок 4.1).

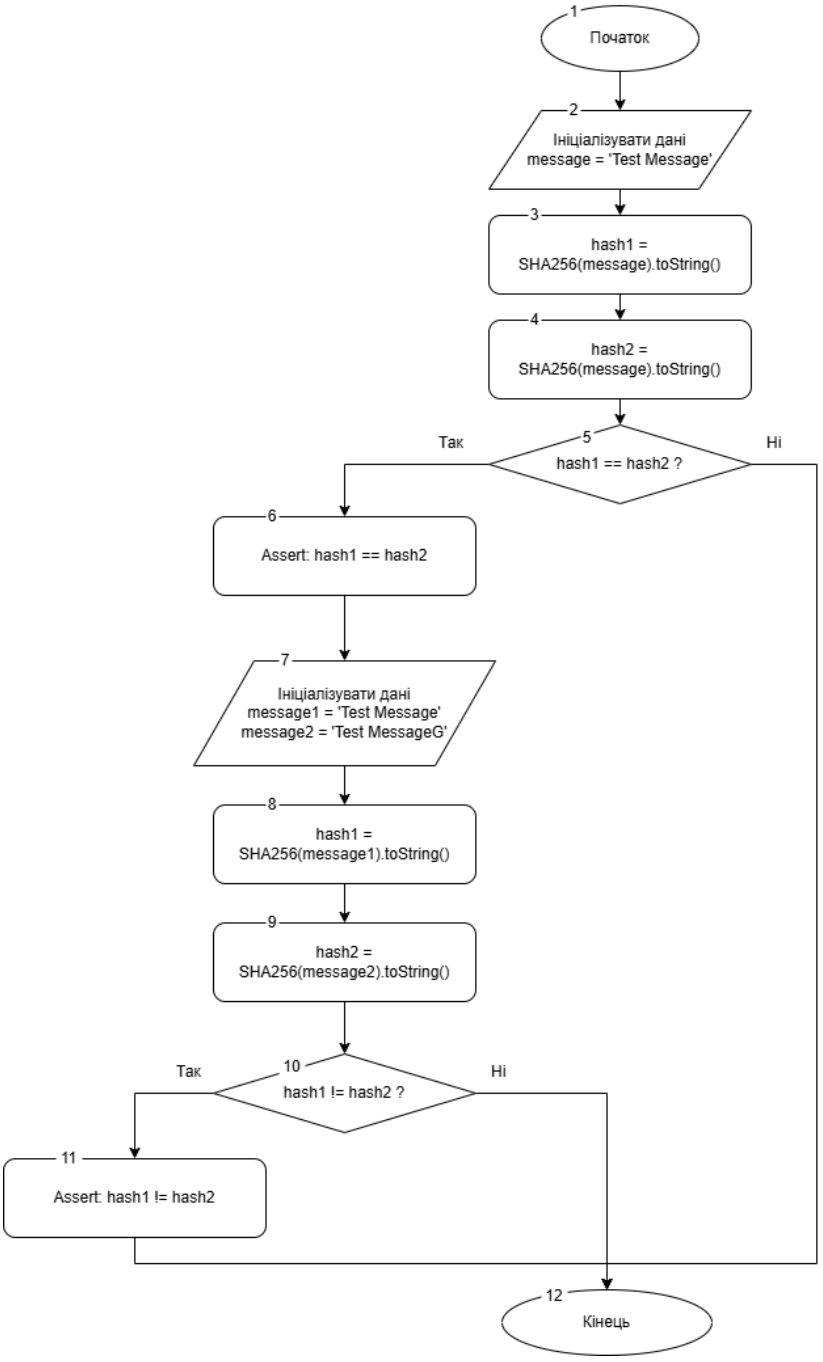


Рисунок 4.1 – Перевірка правильності генерації хешу повідомлення

Крім хешування, особливу увагу приділено реалізації алгоритму цифрового підпису. У системі застосовується алгоритм ECDSA, який забезпечує високу надійність захисту при помірних обчислювальних витратах. Було змодельовано ситуації формування підпису та перевірки його дійсності. Для цього використовувалась функція підписування повідомлення, яка приймає як

аргументи вміст повідомлення та приватний ключ користувача, а також функція перевірки підпису, яка аналізує правильність зіставлення відкритого ключа, хешу повідомлення та підпису (див. рисунок 4.2).

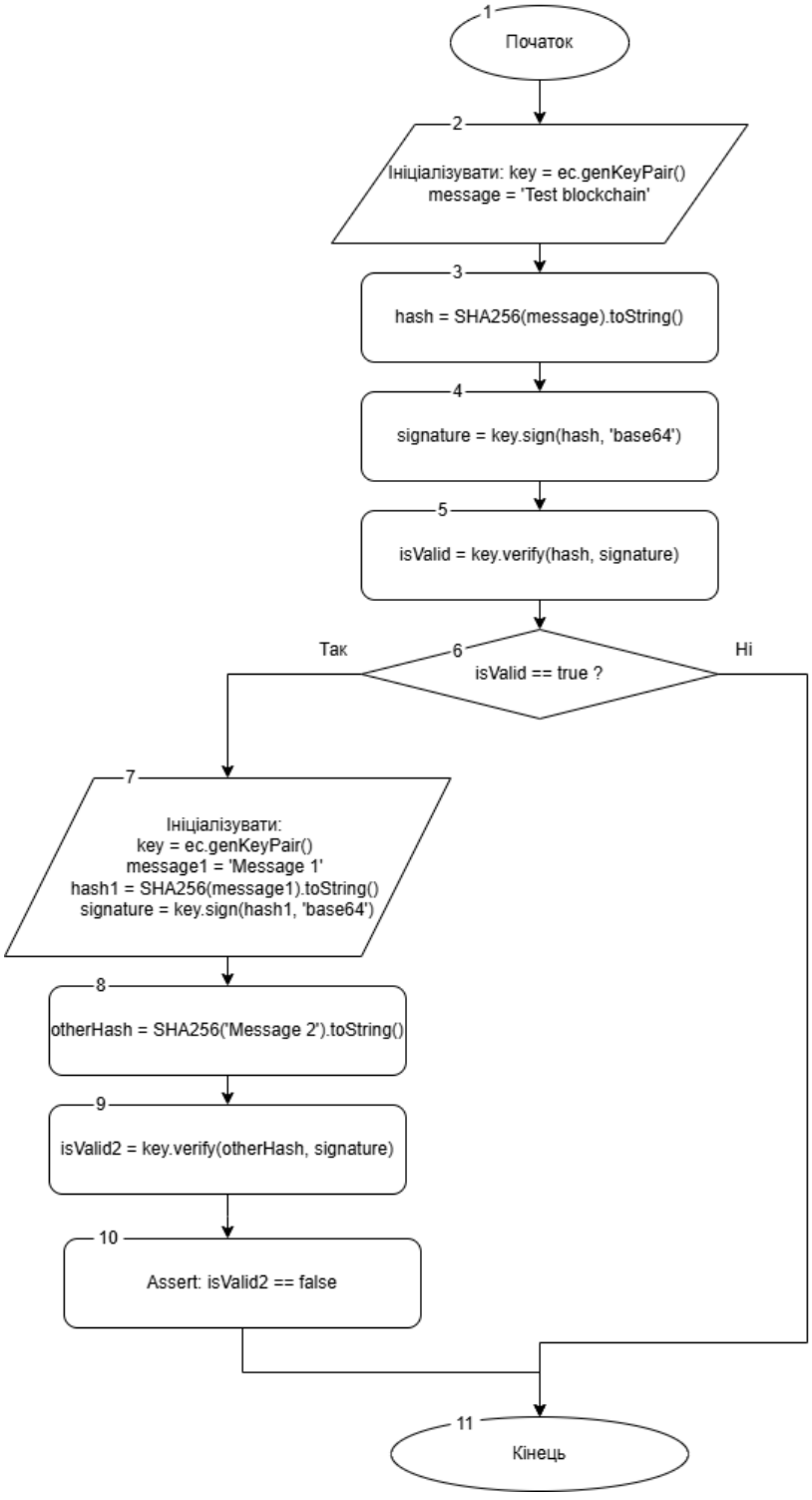


Рисунок 4.2 – Перевірка створення та верифікації цифрового підпису

Збереження повідомлень відбувалося з урахуванням реального часу, що дозволило перевірити логіку роботи з мітками часу та їх хронологічну

послідовність. Було встановлено, що при кожному новому надходженні повідомлення формується новий блок, який містить хеш попереднього, що є базовим механізмом незмінності всієї історії передачі даних у системі.

У межах наступного етапу тестування було здійснено перевірку роботи модуля збереження блоків та побудови зв’язного ланцюга. Основним завданням тесту є підтвердження того, що кожен блок містить хеш попереднього, а зміна вмісту будь-якого блоку порушує цілісність усієї структури.

Було змодельовано ситуацію, за якої вручну змінено вміст одного з раніше створених блоків у базі даних. Після цього запущено функцію перевірки цілісності ланцюга, яка аналізує:

- правильність збереженого хешу повідомлення;
- відповідність поля previousHash хешу попереднього блоку;
- валідність цифрового підпису.

У результаті перевірки отримано негативний висновок про справжність ланцюга, що підтверджує коректність реалізації відповідного алгоритму. При цьому на консоль сервера було виведено повідомлення про помилку у верифікації.

На основі цього тесту сформовано окремий сценарій для негативного тестування, що дозволяє виявляти несанкціоновані втручання у збережену історію комунікації.

Загальну взаємодію користувача із системою перевірено через інтерфейс. Після реєстрації та входу в систему користувач отримує доступ до головної сторінки, де відображається панель навігації, кнопки створення публікацій, доступ до повідомлень, профілю та списку колег по організації.

Головна сторінка виконує роль центрального вузла навігації системи. Вона надає користувачу зручний доступ до особистого профілю, функцій створення публікацій, списку контактів і стрічки новин. Усі основні елементи інтерфейсу згруповано відповідно до функціональної належності для забезпечення інтуїтивної взаємодії (див. рисунок 4.3).

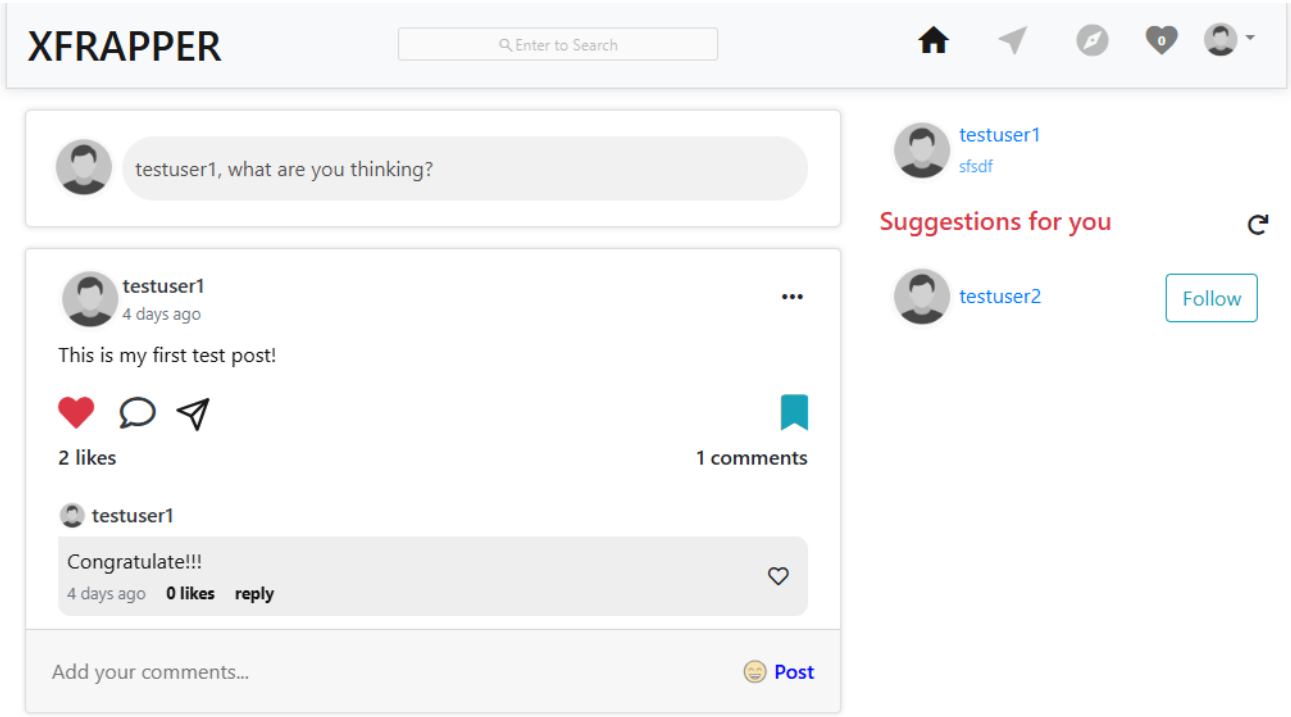


Рисунок 4.3 – Головна сторінка користувача після входу

Подальша перевірка проводилась у розділі обміну повідомленнями. Усі повідомлення надсилалися через вебінтерфейс, з обробкою у реальному часі. В ході тесту перевірено:

- правильність оновлення списку повідомлень;
- формування повідомлень у базі;
- коректність обчислення хешу;
- прикріплення цифрового підпису.

Повідомлення одразу відображаються у правій частині інтерфейсу, мають позначку з іменем відправника, датою та часом, що підтверджує динамічність взаємодії.

Вікно обміну повідомленнями дозволяє здійснювати миттєве листування між користувачами системи. Інтерфейс передбачає окрему панель для списку діалогів і простір для активного перегляду повідомлень у вибраному чаті, що забезпечує швидкий доступ до актуальної інформації та зручність спілкування.

Вікно обміну повідомленнями зображено на рисунку 4.4.

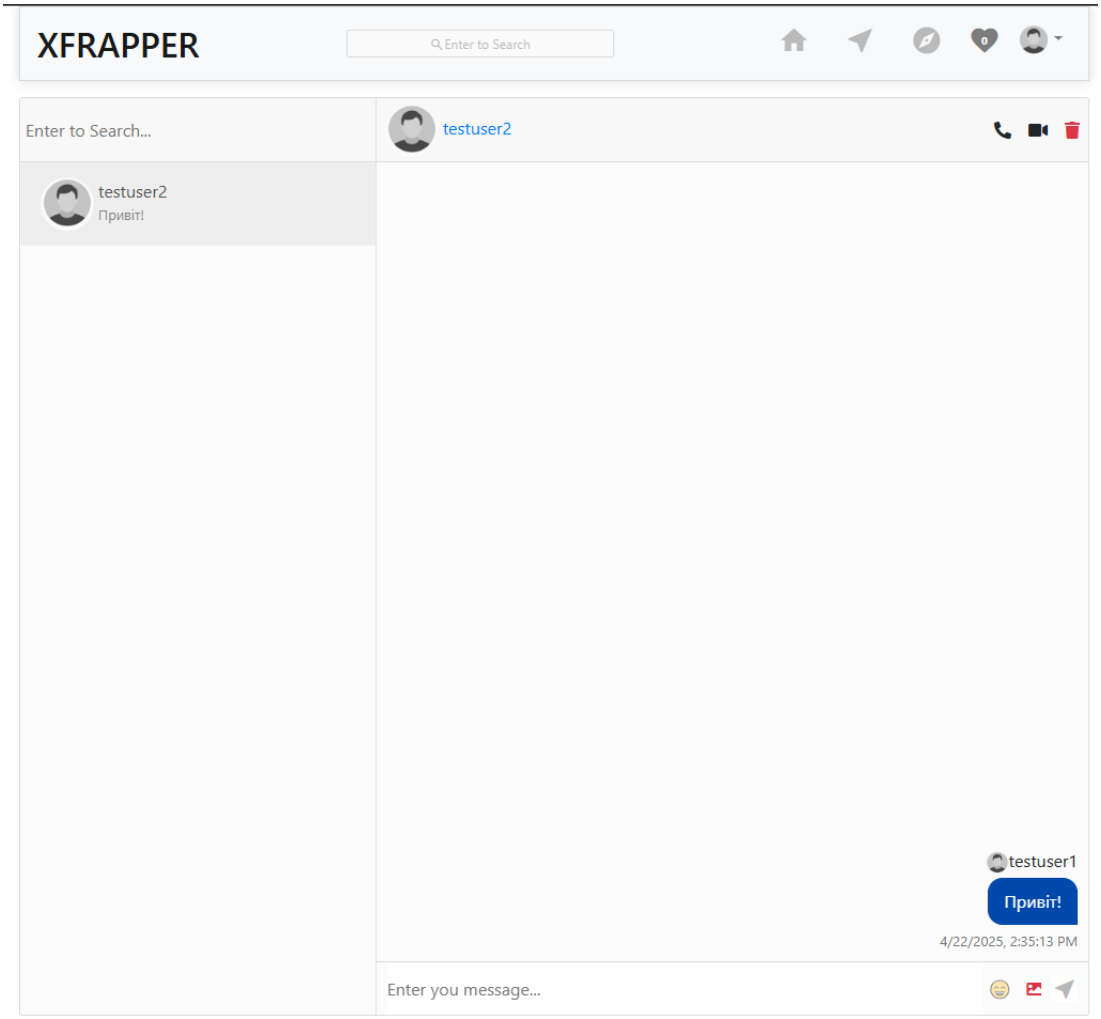


Рисунок 4.4 – Вікно обміну повідомленнями між користувачами

Крім цього, протестовано поведінку системи у разі відключення сервера або помилок бази даних. У таких ситуаціях система виводить відповідне повідомлення про неможливість надсилання, повідомлення не фіксується у ланцюгу, а функція перевірки хешів повертає порожній результат. Це дозволяє користувачам уникати втрати повідомлень та зберігати цілісність історії листування.

За результатами всіх етапів тестування встановлено, що система працює стабільно, виконує поставлені функції, надає коректні відповіді у разі помилок, а всі модулі, включаючи блокчейн-компоненти, працюють згідно з проектним завданням. Реалізація відповідає вимогам до інформаційної безпеки та корпоративного збереження даних.

На завершальному етапі тестування виконано перевірку стабільності та стійкості системи при тривалому використанні, великій кількості повідомлень, повторюваних діях користувачів і потенційних збоїв у збереженні даних. Особливу увагу приділено поведінці системи в умовах навантаження, коли в базу даних надходить значна кількість блоків, а кожен користувач взаємодіє з модулем обміну повідомленнями в реальному часі.

Було створено кілька десятків тестових повідомлень, які генерувалися вручну та надсилалися у короткий проміжок часу. Встановлено, що система коректно реагує на повторювані дії, не допускає дублювання хешів, а кожне повідомлення формується у вигляді окремого блоку з унікальними параметрами – навіть у разі повторення вмісту. Таким чином, доведено, що цифровий підпис та хеш залежать не лише від тексту повідомлення, а й від часу його створення та конкретного сеансу взаємодії.

У межах перевірки було вручну змінено дані одного з блоків, щоб моделювати спробу втручання. Після цього функція перевірки цілісності ланцюга повернула негативний результат. Подібним чином, при навмисному видаленні одного з проміжних блоків система виявляє порушення структури. Така поведінка підтверджує, що механізм валідації реагує не лише на зміну вмісту, а й на порушення зв’язності ланцюга. Водночас інші частини системи не припиняють функціонування, а лише повідомляють про втрату цілісності.

Усі перевірки проводилися у звичному середовищі запуску, за допомогою емуляції дій користувача через браузер, а також шляхом взаємодії з API та консольною частиною сервера. Аналіз відповіді бази даних після кожної дії підтверджує, що структура блоків формується відповідно до логіки системи, поля присутні у правильному форматі, а функції зчитування та запису виконуються без втрат.

У процесі перевірки продуктивності встановлено, що система стабільно обробляє нові повідомлення без суттєвих затримок. Створення блоку, підписування, обчислення хешу та збереження у базу займає не більше 150 мс на

операцію. Усі дії виконуються у реальному часі, а зміни одразу відображаються на інтерфейсі користувача.

В результаті тестуванні можна констатувати, що система поводиться стабільно, надійно захищає інформацію користувачів, забезпечує прозорість взаємодії та дозволяє фіксувати історію повідомлень у вигляді незмінного ланцюга. Така реалізація робить програмний продукт придатним до використання у середовищах з підвищеними вимогами до інформаційної безпеки, включаючи внутрішні корпоративні мережі.

4.2 Розробка інструкції користувача

Після запуску вебзастосунку користувач отримує доступ до головного інтерфейсу системи корпоративної комунікації. Для початку роботи необхідно здійснити авторизацію, яка передбачає введення логіну та пароля, збережених у системі. У разі успішного входу відкривається головна сторінка профілю, де відображається персональна інформація користувача, доступ до основних функцій та блок навігації.

У верхній частині екрана розміщено панель керування, за допомогою якої можна переходити між розділами, переглядати публікації колег, відкривати список повідомлень, а також змінювати налаштування профілю. Підключення до системи здійснюється автоматично, і при першому відкритті розділу обміну повідомленнями виконується синхронізація локального інтерфейсу з базою даних.

Для надсилання повідомлення необхідно перейти у відповідний розділ, де виводиться список попередніх діалогів. У нижній частині вікна розміщено поле введення тексту, після заповнення якого достатньо натиснути кнопку відправлення. Система автоматично обробляє повідомлення, виконує його підписування приватним ключем користувача та зберігає у базу даних у вигляді окремого блоку з відповідною хеш-структурою.

Після відправлення повідомлення миттєво з’являється в інтерфейсі, а також зберігається у розподіленому ланцюзі. У разі отримання нового

повідомлення від іншого користувача система автоматично оновлює діалог та виводить повідомлення на екран, супроводжуючи його візуальним індикатором та міткою часу. Таким чином, обмін повідомленнями відбувається у реальному часі, без необхідності перезавантаження сторінки.

Користувач може у будь-який момент переглянути свій профіль, змінити основні дані або ознайомитися з активністю інших учасників. Усі повідомлення є доступними для перегляду, але не можуть бути змінені або видалені – це забезпечує незмінність комунікації, яка гарантується застосуванням технології блокчейн.

Інтерфейс побудовано таким чином, щоб зменшити навантаження на користувача та спростити навігацію. Всі елементи мають зрозуміле розташування, відображаються однаково на різних пристроях і не потребують додаткових інструкцій. Система функціонує у фоновому режимі, фіксуючи всі дії у вигляді записів, які можуть бути перевірені адміністративними модулями при потребі.

Таким чином, робота із застосунком не вимагає спеціальної підготовки. Усі ключові функції доступні через інтуїтивно зрозумілий вебінтерфейс, а обробка повідомлень, їх збереження, підписування та перевірка цілісності виконується без необхідності участі користувача в технічних аспектах реалізації.

4.3 Висновки до четвертого розділу

У четвертому розділі роботи здійснено комплексне тестування розробленої інформаційної системи на відповідність технічним вимогам та критеріям працездатності.

Виконано модульне тестування окремих компонентів системи, зокрема модуля обміну повідомленнями, модуля перевірки достовірності повідомлень, модуля генерації цифрового підпису та модуля перевірки цілісності блокчейну. Перевірено правильність функціонування кожного модуля в умовах окремого середовища.

Проведено інтеграційне тестування взаємодії модулів між собою у складі єдиної системи. Тестування підтвердило коректну роботу усіх основних сценаріїв: надсилання повідомлення, верифікація повідомлення, збереження повідомлень у блокчейні та перевірка цілісності ланцюга блоків.

Проведено перевірку на відповідність обробки помилок: перевірено стійкість системи до некоректних даних, неправильних цифрових підписів, порушення послідовності блоків та атаки на збереження даних.

Оцінено продуктивність системи в умовах підвищеного навантаження, протестовано швидкість обробки повідомлень і фіксації транзакцій у блокчейні.

Окремо розроблено інструкцію користувача, що містить опис процесу реєстрації, авторизації, надсилання повідомлень та перевірки достовірності комунікацій.

У процесі роботи з застосунком не зафіксовано помилок або збоїв, які б впливали на роботу модулів та системи. Тестування підтвердило надійність реалізації та відповідність програмного продукту очікуваним критеріям. Система повністю готова до використання у корпоративному середовищі як інструмент безпечної внутрішньої комунікації із застосуванням технології блокчейн.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було розроблено вебсистему корпоративних комунікацій з використанням блокчейн технології. Основною метою роботи було створення безпечного середовища для обміну повідомленнями в межах організації з фіксацією всіх дій у захищеному незмінному ланцюгу.

У ході виконання завдання проведено аналіз існуючих систем електронного листування та захисту даних, виявлено їхні недоліки, що стали підґрунтям для формування технічних вимог до власного рішення. Розглянуто сучасні підходи до побудови блокчейн-архітектури, обґрунтовано вибір алгоритмів хешування та підписування повідомлень, досліджено можливості інтеграції криптографічних методів у межах вебзастосунку.

Запропоновано архітектуру програмного забезпечення корпоративної системи комунікацій, яка базується на приватному блокчейні та забезпечує збереження автентичності повідомлень без використання централізованого сервера, що дозволяє підвищити швидкодію здійснення комунікацій з підтримкою високого рівня безпеки.

Удосконалено метод перевірки цілісності повідомлень у розподіленому середовищі, який використовує методи криптографічного хешування в комбінації з цифровим підписом, що забезпечує верифікацію авторства без втрати продуктивності.

Подальшого розвитку набули методи до розробки безпечних вебзастосунків для внутрішніх комунікацій, які використовують механізми перевірки достовірності повідомлень через блокчейн-записи без залучення проміжних вузлів, що збільшує швидкодію без втрати рівня достовірності.

Проєктна реалізація охоплює модулі створення, підписування, збереження та перевірки повідомлень. У розробленій системі кожне повідомлення фіксується у вигляді окремого блоку, що містить хеш, цифровий підпис та посилання на попередній блок. Це дозволяє гарантувати незмінність інформації,

відстежувати усі дії користувача та формувати довірене середовище корпоративної взаємодії.

Розроблений застосунок створено у вигляді клієнтсько-серверної системи з використанням сучасних технологій, зокрема JavaScript, Node.js, React та MongoDB. У клієнтському інтерфейсі реалізовано функціонал авторизації, перегляду профілю, створення публікацій і безпечного листування. Серверна частина обробляє запити, виконує операції підписування та перевірки повідомлень, взаємодіє з базою даних.

Проведено повноцінне тестування функціональних модулів. Результати перевірки засвідчили стабільну роботу системи, правильність формування блоків, достовірність підписів, а також надійність механізму виявлення порушень у структурі блокчейну. Інтерфейс забезпечує зручність користування, не вимагає спеціальних знань і підходить для широкого кола користувачів.

Завдання роботи виконано в повному обсязі. Реалізоване програмне забезпечення може бути використане у внутрішніх інформаційних системах підприємств, де потрібен захищений обмін повідомленнями та надійне зберігання історії взаємодії. Отримані результати створюють підґрунтя для подальшого розвитку системи, розширення її функціоналу та інтеграції з іншими корпоративними сервісами.

Бакалаврську дипломну роботу оформлено відповідно до методичних вказівок [27].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Загорулько Є. О., Олтаржевський Д. О. Корпоративні комунікації: свіжий погляд : монографія. Київ : Арт Економі, 2023. 360 с.

2. Чубаєвський В. І. Корпоративна інформаційна безпека : монографія. Київ : Держ. торг.-екон. ун-т, 2022. 272 с.

3. Древинський В.В. Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технологій. Збірник матеріалів міжнародної науково-практичної конференції «Інформаційні технології в освіті та науці». URL: https://mdpu.org.ua/nauka/konferentsiyi-ta-seminary/

4. Древинський В.В. Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технологій. Збірник матеріалів міжнародної конференції «Молодь в науці 2025». URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025

5. Шаматієнко Ю.В., Коваленко О.О, Тенденції проектування сучасних інформаційних систем діловодства Коваленко https://conferences.vntu.edu.ua/index.php/mn/mn2020/paper/view/10556

6. Коваленко О.О., Черначук Н.В. Розробка системи моніторингу та автоматизованого управління учасниками “Blockchain” мережі. Електронні інформаційні ресурси: створення, використання, доступ: Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 9-10 листопада 2020 р. – Суми/Вінниця : НІКО/ВНТУ, 2020. – 280 с. - ISBN: 978-617-7422-13-5. – С. 133-136.

7. Kushch S, Baryshev Y, Ranise S Blockchain tree as solution for distributed storage of personal id data and document access control. *Sensors,* 2020. https://www.mdpi.com/1424-8220/20/13/3621

8. Голубенко О. П. Криптографічні засоби захисту інформації : підручник. Київ : НаУКМА, 2022. 288 с.

9. Microsoft Corporation. Microsoft Teams Documentation [Електронний ресурс]. URL: https://docs.microsoft.com/teams (дата звернення 02.06.2025).

10. Slack Technologies, Inc. Slack Security Whitepaper [Електронний ресурс]. URL: https://slack.com/security (дата звернення 02.06.2025).

11. Mattermost, Inc. Mattermost Documentation [Електронний ресурс]. URL: https://docs.mattermost.com (дата звернення 02.06.2025).

12. Rocket.Chat. Rocket.Chat Documentation [Електронний ресурс]. URL: https://docs.rocket.chat (дата звернення 02.06.2025).

13. Element (Matrix.org Foundation). Element (Matrix) Documentation [Електронний ресурс]. URL: https://element.io/help (дата звернення 02.06.2025).

14. Antonopoulos A. Mastering Bitcoin: Programming the Open Blockchain. 3-є вид. Sebastopol : O'Reilly Media, 2023. 354 с.

15. Katz J., Lindell Y. Introduction to Modern Cryptography: Principles and Protocols. 3-є вид. Boca Raton : CRC Press, 2020. 511 с.

16. Жилін А. В., Шаповал О. М., Успенський О. А. Технології захисту інформації в інформаційно-телекомунікаційних системах : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 213 с.

17. Koibichuk V., Rozhkova M. Research for application of blockchain technologies in the world's enterprises activity: methodical approach. Pryazovskyi economic herald. 2020. № 4(21). URL: https://doi.org/10.32840/2522-4263/2020-4-20 (дата звернення: 03.06.2025).

18. Richards M., Ford N. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020. 432 с.

19. Juneau J. RESTful web services. Jakarta EE recipes. Berkeley, CA, 2020. С. 669–704. URL: https://doi.org/10.1007/978-1-4842-5587-2_13 (дата звернення: 04.06.2025).

20. Chodorow K., Brazil E., Bradshaw S. MongoDB: the definitive guide: powerful and scalable data storage. 3-тє вид. Sebastopol : O'Reilly Media, 2020. 512 с.

21. Chellappan S., Ganesan D. MongoDB recipes. Berkeley, CA : Apress, 2020. URL: https://doi.org/10.1007/978-1-4842-4891-1 (дата звернення: 04.06.2025).

22. Node.js Foundation. Node.js Documentation [Електронний ресурс]. URL: https://nodejs.org/en/docs (дата звернення 02.06.2025).

23. Express.js. Express.js Documentation [Електронний ресурс]. URL: https://expressjs.com (дата звернення 02.06.2025).

24. Socket.io. Socket.io Documentation [Електронний ресурс]. URL: https://socket.io/docs (дата звернення 02.06.2025).

25. Kaufmann M., Meier A. SQL and NoSQL Databases. Cham : Springer Nature Switzerland, 2023. URL: https://doi.org/10.1007/978-3-031-27908-9 (дата звернення: 04.06.2025).

26. Hyperledger Fabric. Hyperledger Fabric Documentation [Електронний ресурс]. URL: https://hyperledger-fabric.readthedocs.io (дата звернення 02.06.2025).

27. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» . уклад. : О. Н. Романюк, Г. О. Черноволик. Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

62

Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф
_____О. Н. Романюк
« 21 » березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технології» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доц. каф. ПЗ О.О. Коваленко
«21» березня 2025р.

Виконав:
_____ студент гр. 2ПІ-21б В. В. Древинський
«21» березня 2025р

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технології».

Галузь застосування – корпоративний сектор.

2. Підстава для розробки.

Індивідуальне завдання на бакалавську кваліфікаційну роботу (БКР) та наказ №97 від 20 березня 2025 року про затвердження тем БКР

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є вдосконалення корпоративної комунікації шляхом розробки і використання веборієнтованої інформаційної системи, яка забезпечує захищений обмін повідомленнями між користувачами з використанням механізмів фіксації, автентифікації та верифікації даних на основі блокчейн-технологій.

Призначення роботи – розробка системи корпоративного електронного обміну повідомленнями.

4. Вихідні дані для проведення НДР

1. Kushch S, Baryshev Y, Ranise S Blockchain tree as solution for distributed storage of personal id data and document access control. *Sensors*, 2020. https://www.mdpi.com/1424-8220/20/13/3621

2. Голубенко О. П. Криптографічні засоби захисту інформації : підручник. Київ : НаУКМА, 2022. 288 с.

3. Richards M., Ford N. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020. 432 с.

5. Технічні вимоги

Комп’ютер з 64-розрядним процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 2 ГБ, розмір жорсткого диску 32 ГБ. Операційна система Windows 7/8/10.

6. Конструктивні вимоги.

Додаток має відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану систем корпоративних комунікацій та постановка задач розробки	25.03.25 – 10.04
2	Розробка структури та моделі програмного продукту	11.04.25 – 22.04.25
3	Розробка програмних модулів	23.04.25 – 15.05.25
4	Тестування програми	16.05.25-22.05.25
5	Оформлення матеріалів до захисту БКР	23.05.25– 30.05.25

10 Порядок контролю та прийняття.

Усі етапи бакалаврської дипломної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б - Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технології

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ Кафедра програмного забезпечення, ФІТКІ, 2ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2.7 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Коваленко О.О., к.т.н доцент кафедри ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Древинський В.В.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

```
require('dotenv').config()
const express = require('express')
const mongoose = require('mongoose')
const cors = require('cors')
const cookieParser = require('cookie-parser')
const SocketServer = require('./socketServer')
const { ExpressPeerServer } = require('peer')
const path = require('path')



const app = express()
app.use(express.json())
app.use(cors())
app.use(cookieParser())


// Socket
const http = require('http').createServer(app)
const io = require('socket.io')(http)

io.on('connection', socket => {
    SocketServer(socket)
})

// Create peer server
ExpressPeerServer(http, { path: '/' })


// Routes
app.use('/api', require('./routes/authRouter'))
app.use('/api', require('./routes/userRouter'))
app.use('/api', require('./routes/postRouter'));
app.use('/api', require('./routes/commentRouter'))
app.use('/api', require('./routes/notifyRouter'))
app.use('/api', require('./routes/messageRouter'))


const URI = process.env.MONGODB_URL
mongoose.connect(URI, {
    useCreateIndex: true,
    useFindAndModify: false,
    useNewUrlParser: true,
    useUnifiedTopology: true
}, err => {
    if(err) throw err;
    console.log('Connected to mongodb')
})

if(process.env.NODE_ENV === 'production'){
```

```
    app.use(express.static('client/build'))
    app.get('*', (req, res) => {
        res.sendFile(path.join(__dirname, 'client', 'build',
'index.html'))
    })
}


const port = process.env.PORT || 5000
http.listen(port, () => {
    console.log('Server is running on port', port)
})

let users = []

const EditData = (data, id, call) => {
    const newData = data.map(item =>
        item.id === id ? {...item, call} : item
    )
    return newData;
}

const SocketServer = (socket) => {
    // Connect - Disconnect
    socket.on('joinUser', user => {
        users.push({id: user._id, socketId: socket.id, followers:
user.followers})
    })

    socket.on('disconnect', () => {
        const data = users.find(user => user.socketId ===
socket.id)
        if(data){
            const clients = users.filter(user =>
                data.followers.find(item => item._id === user.id)
            )

            if(clients.length > 0){
                clients.forEach(client => {

socket.to(`${client.socketId}`).emit('CheckUserOffline', data.id)
                })
            }

            if(data.call){
                const callUser = users.find(user => user.id ===
data.call)
                if(callUser){
                    users = EditData(users, callUser.id, null)

socket.to(`${callUser.socketId}`).emit('callerDisconnect')
                }
            }
```

```
        }

        users = users.filter(user => user.socketId !== socket.id)
    })


    // Likes
    socket.on('likePost', newPost => {
        const ids = [...newPost.user.followers, newPost.user._id]
        const clients = users.filter(user =>
ids.includes(user.id))

        if(clients.length > 0){
            clients.forEach(client => {

socket.to(`${client.socketId}`).emit('likeToClient', newPost)
            })
        }
    })

    socket.on('unLikePost', newPost => {
        const ids = [...newPost.user.followers, newPost.user._id]
        const clients = users.filter(user =>
ids.includes(user.id))

        if(clients.length > 0){
            clients.forEach(client => {

socket.to(`${client.socketId}`).emit('unLikeToClient', newPost)
            })
        }
    })


    // Comments
    socket.on('createComment', newPost => {
        const ids = [...newPost.user.followers, newPost.user._id]
        const clients = users.filter(user =>
ids.includes(user.id))

        if(clients.length > 0){
            clients.forEach(client => {

socket.to(`${client.socketId}`).emit('createCommentToClient',
newPost)
            })
        }
    })

    socket.on('deleteComment', newPost => {
        const ids = [...newPost.user.followers, newPost.user._id]
        const clients = users.filter(user =>
ids.includes(user.id))
```

```
        if(clients.length > 0){
            clients.forEach(client => {

socket.to(`${client.socketId}`).emit('deleteCommentToClient',
newPost)
            })
        }
    })


    // Follow
    socket.on('follow', newUser => {
        const user = users.find(user => user.id === newUser._id)
        user &&
socket.to(`${user.socketId}`).emit('followToClient', newUser)
    })

    socket.on('unFollow', newUser => {
        const user = users.find(user => user.id === newUser._id)
        user &&
socket.to(`${user.socketId}`).emit('unFollowToClient', newUser)
    })


    // Notification
    socket.on('createNotify', msg => {
        const client = users.find(user =>
msg.recipients.includes(user.id))
        client &&
socket.to(`${client.socketId}`).emit('createNotifyToClient', msg)
    })

    socket.on('removeNotify', msg => {
        const client = users.find(user =>
msg.recipients.includes(user.id))
        client &&
socket.to(`${client.socketId}`).emit('removeNotifyToClient', msg)

    })


    // Message
    socket.on('addMessage', msg => {
        const user = users.find(user => user.id === msg.recipient)
        user &&
socket.to(`${user.socketId}`).emit('addMessageToClient', msg)
    })


    // Check User Online / Offline
    socket.on('checkUserOnline', data => {
        const following = users.filter(user =>
```

```
            data.following.find(item => item._id === user.id)
        )
        socket.emit('checkUserOnlineToMe', following)

        const clients = users.filter(user =>
            data.followers.find(item => item._id === user.id)
        )

        if(clients.length > 0){
            clients.forEach(client => {

socket.to(`${client.socketId}`).emit('checkUserOnlineToClient',
data._id)
            })
        }

    })


    // Call User
    socket.on('callUser', data => {
        users = EditData(users, data.sender, data.recipient)

        const client = users.find(user => user.id ===
data.recipient)

        if(client){
            if(client.call){
                socket.emit('userBusy', data)
                users = EditData(users, data.sender, null)
            }else{
                users = EditData(users, data.recipient,
data.sender)

socket.to(`${client.socketId}`).emit('callUserToClient', data)
            }
        }
    })

    socket.on('endCall', data => {
        const client = users.find(user => user.id === data.sender)

        if(client){

socket.to(`${client.socketId}`).emit('endCallToClient', data)
            users = EditData(users, client.id, null)

            if(client.call){
                const clientCall = users.find(user => user.id ===
client.call)
                clientCall &&
socket.to(`${clientCall.socketId}`).emit('endCallToClient', data)
```

```
                users = EditData(users, client.call, null)
            }
        }
    })
}

module.exports = SocketServer
import { useEffect } from 'react'
import {BrowserRouter as Router, Route} from 'react-router-dom'

import PageRender from './customRouter/PageRender'
import PrivateRouter from './customRouter/PrivateRouter'

import Home from './pages/home'
import Login from './pages/login'
import Register from './pages/register'

import Alert from './components/alert/Alert'
import Header from './components/header/Header'
import StatusModal from './components/StatusModal'

import { useSelector, useDispatch } from 'react-redux'
import { refreshToken } from './redux/actions/authAction'
import { getPosts } from './redux/actions/postAction'
import { getSuggestions } from './redux/actions/suggestionsAction'

import io from 'socket.io-client'
import { GLOBALTYPES } from './redux/actions/globalTypes'
import SocketClient from './SocketClient'

import { getNotifies } from './redux/actions/notifyAction'
import CallModal from './components/message/CallModal'
import Peer from 'peerjs'

function App() {
  const { auth, status, modal, call } = useSelector(state =>
state)
  const dispatch = useDispatch()

  useEffect(() => {
    dispatch(refreshToken())

    const socket = io()
    dispatch({type: GLOBALTYPES.SOCKET, payload: socket})
    return () => socket.close()
  },[dispatch])

  useEffect(() => {
    if(auth.token) {
      dispatch(getPosts(auth.token))
      dispatch(getSuggestions(auth.token))
      dispatch(getNotifies(auth.token))
    }
```

```
  }, [dispatch, auth.token])


  useEffect(() => {
    if (!("Notification" in window)) {
      alert("This browser does not support desktop notification");
    }
    else if (Notification.permission === "granted") {}
    else if (Notification.permission !== "denied") {
      Notification.requestPermission().then(function (permission)
{
        if (permission === "granted") {}
      });
    }
  },[])


  useEffect(() => {
    const newPeer = new Peer(undefined, {
      path: '/', secure: true
    })

    dispatch({ type: GLOBALTYPES.PEER, payload: newPeer })
  },[dispatch])


  return (
    <Router>
      <Alert />

      <input type="checkbox" id="theme" />
      <div className={`App ${(status || modal) && 'mode'}`}>
        <div className="main">
          {auth.token && <Header />}
          {status && <StatusModal />}
          {auth.token && <SocketClient />}
          {call && <CallModal />}

          <Route exact path="/" component={auth.token ? Home :
Login} />
          <Route exact path="/register" component={Register} />

          <PrivateRouter exact path="/:page"
component={PageRender} />
          <PrivateRouter exact path="/:page/:id"
component={PageRender} />

        </div>
      </div>
    </Router>
  );
}
```

```
export default App;
const Posts = require('../models/postModel')
const Comments = require('../models/commentModel')
const Users = require('../models/userModel')

class APIfeatures {
    constructor(query, queryString){
        this.query = query;
        this.queryString = queryString;
    }

    paginating(){
        const page = this.queryString.page * 1 || 1
        const limit = this.queryString.limit * 1 || 9
        const skip = (page - 1) * limit
        this.query = this.query.skip(skip).limit(limit)
        return this;
    }
}

const postCtrl = {
    createPost: async (req, res) => {
        try {
            const { content, images } = req.body

            // if(images.length === 0)
            // return res.status(400).json({msg: "Please add your
photo."})

            const newPost = new Posts({
                content, images, user: req.user._id
            })
            await newPost.save()

            res.json({
                msg: 'Created Post!',
                newPost: {
                    ...newPost._doc,
                    user: req.user
                }
            })
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    getPosts: async (req, res) => {
        try {
            const features =  new APIfeatures(Posts.find({
                user: [...req.user.following, req.user._id]
            }), req.query).paginating()

            const posts = await features.query.sort('-createdAt')
```

```
            .populate("user likes", "avatar username fullname
followers")
            .populate({
                path: "comments",
                populate: {
                    path: "user likes",
                    select: "-password"
                }
            })

            res.json({
                msg: 'Success!',
                result: posts.length,
                posts
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    updatePost: async (req, res) => {
        try {
            const { content, images } = req.body

            const post = await Posts.findOneAndUpdate({_id:
req.params.id}, {
                content, images
            }).populate("user likes", "avatar username fullname")
            .populate({
                path: "comments",
                populate: {
                    path: "user likes",
                    select: "-password"
                }
            })

            res.json({
                msg: "Updated Post!",
                newPost: {
                    ...post._doc,
                    content, images
                }
            })
        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    likePost: async (req, res) => {
        try {
            const post = await Posts.find({_id: req.params.id,
likes: req.user._id})
            if(post.length > 0) return res.status(400).json({msg:
"You liked this post."})
```

```
            const like = await Posts.findOneAndUpdate({_id:
req.params.id}, {
                $push: {likes: req.user._id}
            }, {new: true})

            if(!like) return res.status(400).json({msg: 'This post
does not exist.'})

            res.json({msg: 'Liked Post!'})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    unLikePost: async (req, res) => {
        try {

            const like = await Posts.findOneAndUpdate({_id:
req.params.id}, {
                $pull: {likes: req.user._id}
            }, {new: true})

            if(!like) return res.status(400).json({msg: 'This post
does not exist.'})

            res.json({msg: 'UnLiked Post!'})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    getUserPosts: async (req, res) => {
        try {
            const features = new APIfeatures(Posts.find({user:
req.params.id}), req.query)
            .paginating()
            const posts = await features.query.sort("-createdAt")

            res.json({
                posts,
                result: posts.length
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    getPost: async (req, res) => {
        try {
            const post = await Posts.findById(req.params.id)
            .populate("user likes", "avatar username fullname
followers")
```

```
            .populate({
                path: "comments",
                populate: {
                    path: "user likes",
                    select: "-password"
                }
            })

            if(!post) return res.status(400).json({msg: 'This post
does not exist.'})

            res.json({
                post
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    getPostsDicover: async (req, res) => {
        try {

            const newArr = [...req.user.following, req.user._id]

            const num  = req.query.num || 9

            const posts = await Posts.aggregate([
                { $match: { user : { $nin: newArr } } },
                { $sample: { size: Number(num) } },
            ])

            return res.json({
                msg: 'Success!',
                result: posts.length,
                posts
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    deletePost: async (req, res) => {
        try {
            const post = await Posts.findOneAndDelete({_id:
req.params.id, user: req.user._id})
            await Comments.deleteMany({_id: {$in: post.comments
}})

            res.json({
                msg: 'Deleted Post!',
                newPost: {
                    ...post,
                    user: req.user
```

```
                }
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    savePost: async (req, res) => {
        try {
            const user = await Users.find({_id: req.user._id,
saved: req.params.id})
            if(user.length > 0) return res.status(400).json({msg:
"You saved this post."})

            const save = await Users.findOneAndUpdate({_id:
req.user._id}, {
                $push: {saved: req.params.id}
            }, {new: true})

            if(!save) return res.status(400).json({msg: 'This user
does not exist.'})

            res.json({msg: 'Saved Post!'})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    unSavePost: async (req, res) => {
        try {
            const save = await Users.findOneAndUpdate({_id:
req.user._id}, {
                $pull: {saved: req.params.id}
            }, {new: true})

            if(!save) return res.status(400).json({msg: 'This user
does not exist.'})

            res.json({msg: 'unSaved Post!'})

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
    getSavePosts: async (req, res) => {
        try {
            const features = new APIfeatures(Posts.find({
                _id: {$in: req.user.saved}
            }), req.query).paginating()

            const savePosts = await features.query.sort("-
createdAt")
```

```
            res.json({
                savePosts,
                result: savePosts.length
            })

        } catch (err) {
            return res.status(500).json({msg: err.message})
        }
    },
}

module.exports = postCtrl
const router = require('express').Router()
const postCtrl = require('../controllers/postCtrl')
const auth = require('../middleware/auth')

router.route('/posts').get(auth, postCtrl.getPosts)

router.post('/post', auth, postCtrl.createPost);


router.route('/post/:id')
    .patch(auth, postCtrl.updatePost)
    .get(auth, postCtrl.getPost)
    .delete(auth, postCtrl.deletePost)

router.patch('/post/:id/like', auth, postCtrl.likePost)

router.patch('/post/:id/unlike', auth, postCtrl.unLikePost)

router.get('/user_posts/:id', auth, postCtrl.getUserPosts)

router.get('/post_discover', auth, postCtrl.getPostsDicover)

router.patch('/savePost/:id', auth, postCtrl.savePost)

router.patch('/unSavePost/:id', auth, postCtrl.unSavePost)

router.get('/getSavePosts', auth, postCtrl.getSavePosts)

module.exports = router
import React, { useState, useRef, useEffect } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { GLOBALTYPES } from '../redux/actions/globalTypes'
import { createPost, updatePost } from
'../redux/actions/postAction'
import Icons from './Icons'
import { imageShow, videoShow } from '../utils/mediaShow'

const StatusModal = () => {
    const { auth, theme, status, socket } = useSelector(state =>
state)
    const dispatch = useDispatch()
```

```
    const [content, setContent] = useState('')
    const [images, setImages] = useState([])

    const [stream, setStream] = useState(false)
    const videoRef = useRef()
    const refCanvas = useRef()
    const [tracks, setTracks] = useState('')

    const handleChangeImages = e => {
        const files = [...e.target.files]
        let err = ""
        let newImages = []

        files.forEach(file => {
            if(!file) return err = "File does not exist."

            if(file.size > 1024 * 1024 * 5){
                return err = "The image/video largest is 5mb."
            }

            return newImages.push(file)
        })

        if(err) dispatch({ type: GLOBALTYPES.ALERT, payload:
{error: err} })
        setImages([...images, ...newImages])
    }

    const deleteImages = (index) => {
        const newArr = [...images]
        newArr.splice(index, 1)
        setImages(newArr)
    }

    const handleStream = () => {
        setStream(true)
        if(navigator.mediaDevices &&
navigator.mediaDevices.getUserMedia){
            navigator.mediaDevices.getUserMedia({video: true})
            .then(mediaStream => {
                videoRef.current.srcObject = mediaStream
                videoRef.current.play()

                const track = mediaStream.getTracks()
                setTracks(track[0])
            }).catch(err => console.log(err))
        }
    }

    const handleCapture = () => {
        const width = videoRef.current.clientWidth;
        const height = videoRef.current.clientHeight;
```

```
        refCanvas.current.setAttribute("width", width)
        refCanvas.current.setAttribute("height", height)

        const ctx = refCanvas.current.getContext('2d')
        ctx.drawImage(videoRef.current, 0, 0, width, height)
        let URL = refCanvas.current.toDataURL()
        setImages([...images, {camera: URL}])
    }

    const handleStopStream = () => {
        tracks.stop()
        setStream(false)
    }

    const handleSubmit = (e) => {
        e.preventDefault()
        // if(images.length === 0)
        // return dispatch({
        //     type: GLOBALTYPES.ALERT, payload: {error: "Please
add your photo."}
        // })

        if(status.onEdit){
            dispatch(updatePost({content, images, auth, status}))
        }else{
            dispatch(createPost({content, images, auth, socket}))
        }


        setContent('')
        setImages([])
        if(tracks) tracks.stop()
        dispatch({ type: GLOBALTYPES.STATUS, payload: false})
    }

    useEffect(() => {
        if(status.onEdit){
            setContent(status.content)
            setImages(status.images)
        }
    },[status])




    return (
        <div className="status_modal">
            <form onSubmit={handleSubmit}>
                <div className="status_header">
                    <h5 className="m-0">Create Post</h5>
                    <span onClick={() => dispatch({
                        type: GLOBALTYPES.STATUS, payload: false
```

```
                    })}>
                        &times;
                    </span>
                </div>

                <div className="status_body">
                    <textarea name="content" value={content}
                    placeholder={`${auth.user.username}, what are
you thinking?`}
                    onChange={e => setContent(e.target.value)}
                    style={{
                        filter: theme ? 'invert(1)' : 'invert(0)',
                        color: theme ? 'white' : '#111',
                        background: theme ? 'rgba(0,0,0,.03)' :
'',
                    }} />

                    <div className="d-flex">
                        <div className="flex-fill"></div>
                        <Icons setContent={setContent}
content={content} theme={theme} />
                    </div>

                    <div className="show_images">
                        {
                            images.map((img, index) => (
                                <div key={index} id="file_img">
                                    {
                                        img.camera ?
imageShow(img.camera, theme)
                                        : img.url
                                            ?<>
                                                {

img.url.match(/video/i)
                                                    ?
videoShow(img.url, theme)
                                                    :
imageShow(img.url, theme)
                                                }
                                            </>
                                            :<>
                                                {

img.type.match(/video/i)
                                                    ?
videoShow(URL.createObjectURL(img), theme)
                                                    :
imageShow(URL.createObjectURL(img), theme)
                                                }
                                            </>
                                    }
```

```
                                    <span onClick={() =>
deleteImages(index)}>&times;</span>
                                </div>
                            ))
                        }
                    </div>

                    {
                        stream &&
                        <div className="stream position-relative">
                            <video autoPlay muted ref={videoRef}
width="100%" height="100%"
                            style={{filter: theme ? 'invert(1)' :
'invert(0)'}} />

                            <span
onClick={handleStopStream}>&times;</span>
                            <canvas ref={refCanvas}
style={{display: 'none'}} />
                        </div>
                    }

                    <div className="input_images">
                        {
                            stream
                            ? <i className="fas fa-camera"
onClick={handleCapture} />
                            : <>
                                <i className="fas fa-camera"
onClick={handleStream} />

                                <div className="file_upload">
                                    <i className="fas fa-image" />
                                    <input type="file" name="file"
id="file"
                                    multiple
accept="image/*,video/*" onChange={handleChangeImages} />
                                </div>
                            </>
                        }

                    </div>

                </div>

                <div className="status_footer">
                    <button className="btn btn-secondary w-100"
type="submit">
                        Post
                    </button>
                </div>

            </form>
```

```
        </div>
    )
}

export default StatusModal
import React, { useEffect, useRef } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { POST_TYPES } from './redux/actions/postAction'
import { GLOBALTYPES } from './redux/actions/globalTypes'
import { NOTIFY_TYPES } from './redux/actions/notifyAction'
import { MESS_TYPES } from './redux/actions/messageAction'

import audiobell from './audio/got-it-done-613.mp3'


const spawnNotification = (body, icon, url, title) => {
    let options = {
        body, icon
    }
    let n = new Notification(title, options)

    n.onclick = e => {
        e.preventDefault()
        window.open(url, '_blank')
    }
}

const SocketClient = () => {
    const { auth, socket, notify, online, call } =
useSelector(state => state)
    const dispatch = useDispatch()

    const audioRef = useRef()

    // joinUser
    useEffect(() => {
        socket.emit('joinUser', auth.user)
    },[socket, auth.user])

    // Likes
    useEffect(() => {
        socket.on('likeToClient', newPost =>{
            dispatch({type: POST_TYPES.UPDATE_POST, payload:
newPost})
        })

        return () => socket.off('likeToClient')
    },[socket, dispatch])

    useEffect(() => {
        socket.on('unLikeToClient', newPost =>{
            dispatch({type: POST_TYPES.UPDATE_POST, payload:
newPost})
```

```
        })

        return () => socket.off('unLikeToClient')
    },[socket, dispatch])


    // Comments
    useEffect(() => {
        socket.on('createCommentToClient', newPost =>{
            dispatch({type: POST_TYPES.UPDATE_POST, payload:
newPost})
        })

        return () => socket.off('createCommentToClient')
    },[socket, dispatch])

    useEffect(() => {
        socket.on('deleteCommentToClient', newPost =>{
            dispatch({type: POST_TYPES.UPDATE_POST, payload:
newPost})
        })

        return () => socket.off('deleteCommentToClient')
    },[socket, dispatch])


    // Follow
    useEffect(() => {
        socket.on('followToClient', newUser =>{
            dispatch({type: GLOBALTYPES.AUTH, payload: {...auth,
user: newUser}})
        })

        return () => socket.off('followToClient')
    },[socket, dispatch, auth])

    useEffect(() => {
        socket.on('unFollowToClient', newUser =>{
            dispatch({type: GLOBALTYPES.AUTH, payload: {...auth,
user: newUser}})
        })

        return () => socket.off('unFollowToClient')
    },[socket, dispatch, auth])


    // Notification
    useEffect(() => {
        socket.on('createNotifyToClient', msg =>{
            dispatch({type: NOTIFY_TYPES.CREATE_NOTIFY, payload:
msg})

            if(notify.sound) audioRef.current.play()
```

```
            spawnNotification(
                msg.user.username + ' ' + msg.text,
                msg.user.avatar,
                msg.url,
                'xfrapper'
            )
        })

        return () => socket.off('createNotifyToClient')
    },[socket, dispatch, notify.sound])

    useEffect(() => {
        socket.on('removeNotifyToClient', msg =>{
            dispatch({type: NOTIFY_TYPES.REMOVE_NOTIFY, payload:
msg})
        })

        return () => socket.off('removeNotifyToClient')
    },[socket, dispatch])


    // Message
    useEffect(() => {
        socket.on('addMessageToClient', msg =>{
            dispatch({type: MESS_TYPES.ADD_MESSAGE, payload: msg})

            dispatch({
                type: MESS_TYPES.ADD_USER,
                payload: {
                    ...msg.user,
                    text: msg.text,
                    media: msg.media
                }
            })
        })

        return () => socket.off('addMessageToClient')
    },[socket, dispatch])

    // Check User Online / Offline
    useEffect(() => {
        socket.emit('checkUserOnline', auth.user)
    },[socket, auth.user])

    useEffect(() => {
        socket.on('checkUserOnlineToMe', data =>{
            data.forEach(item => {
                if(!online.includes(item.id)){
                    dispatch({type: GLOBALTYPES.ONLINE, payload:
item.id})
                }
            })
        })
```

```
        return () => socket.off('checkUserOnlineToMe')
    },[socket, dispatch, online])

    useEffect(() => {
        socket.on('checkUserOnlineToClient', id =>{
            if(!online.includes(id)){
                dispatch({type: GLOBALTYPES.ONLINE, payload: id})
            }
        })

        return () => socket.off('checkUserOnlineToClient')
    },[socket, dispatch, online])

    // Check User Offline
    useEffect(() => {
        socket.on('CheckUserOffline', id =>{
            dispatch({type: GLOBALTYPES.OFFLINE, payload: id})
        })

        return () => socket.off('CheckUserOffline')
    },[socket, dispatch])


    // Call User
    useEffect(() => {
        socket.on('callUserToClient', data =>{
            dispatch({type: GLOBALTYPES.CALL, payload: data})
        })

        return () => socket.off('callUserToClient')
    },[socket, dispatch])

    useEffect(() => {
        socket.on('userBusy', data =>{
            dispatch({type: GLOBALTYPES.ALERT, payload: {error:
`${call.username} is busy!`}})
        })

        return () => socket.off('userBusy')
    },[socket, dispatch, call])



    return (
        <>
            <audio controls ref={audioRef} style={{display:
'none'}} >
                <source src={audiobell} type="audio/mp3" />
            </audio>
        </>
    )
}
```

```
export default SocketClient
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP,
getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка програмного забезпечення системи корпоративних комунікацій з використанням блокчейн технології

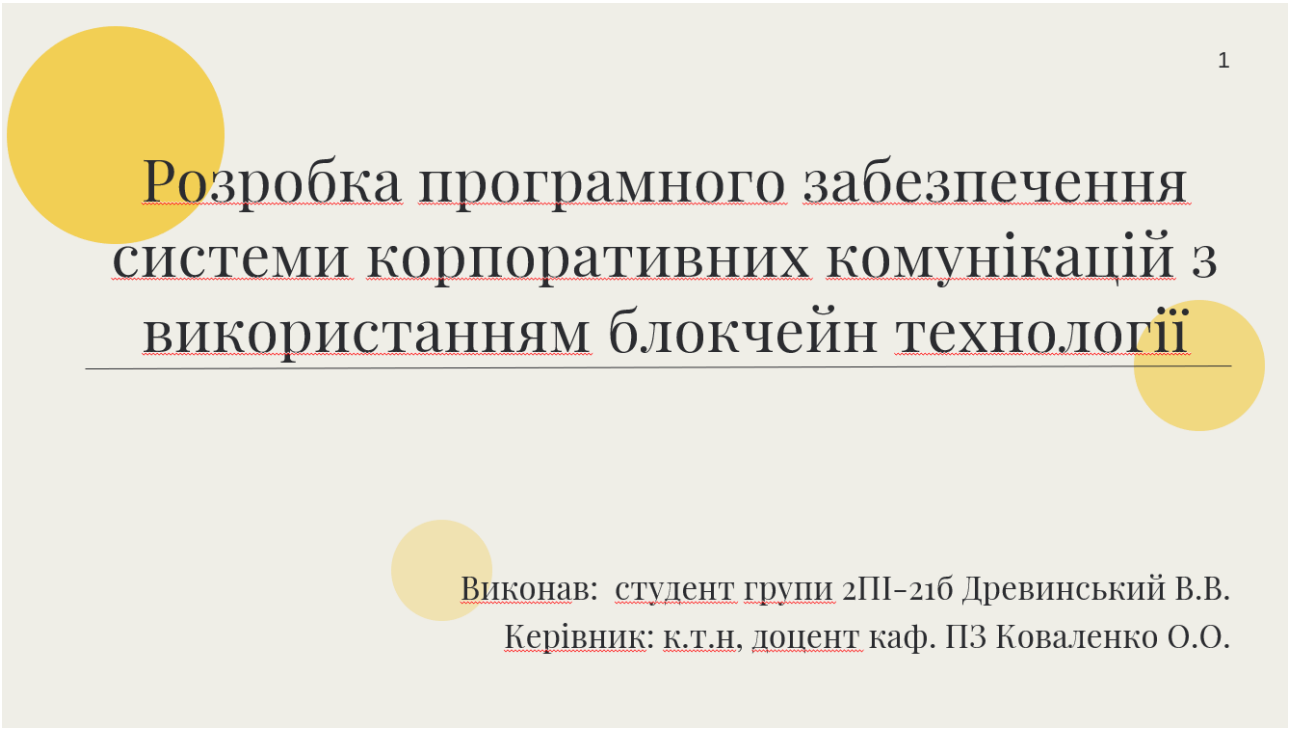


Рисунок Г.1 – Слайд «Титульний аркуш»

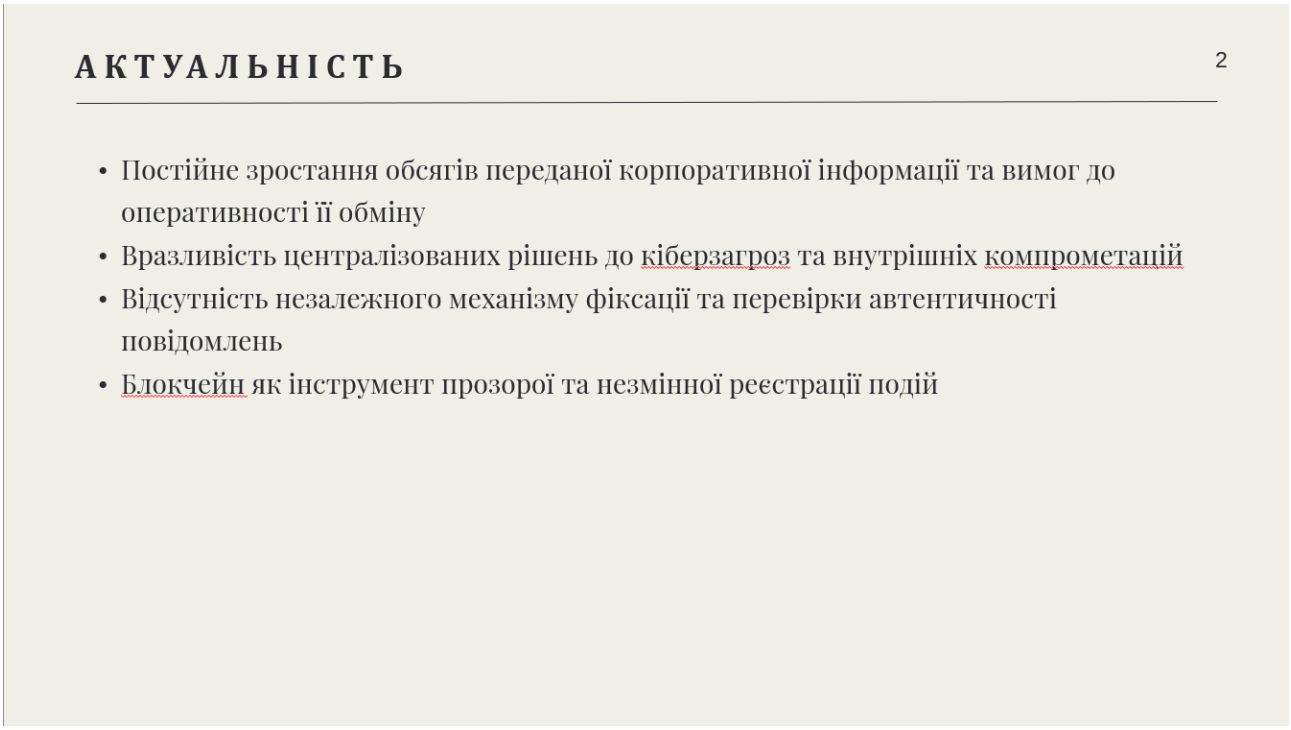


Рисунок Г.2 – Слайд «Актуальність»

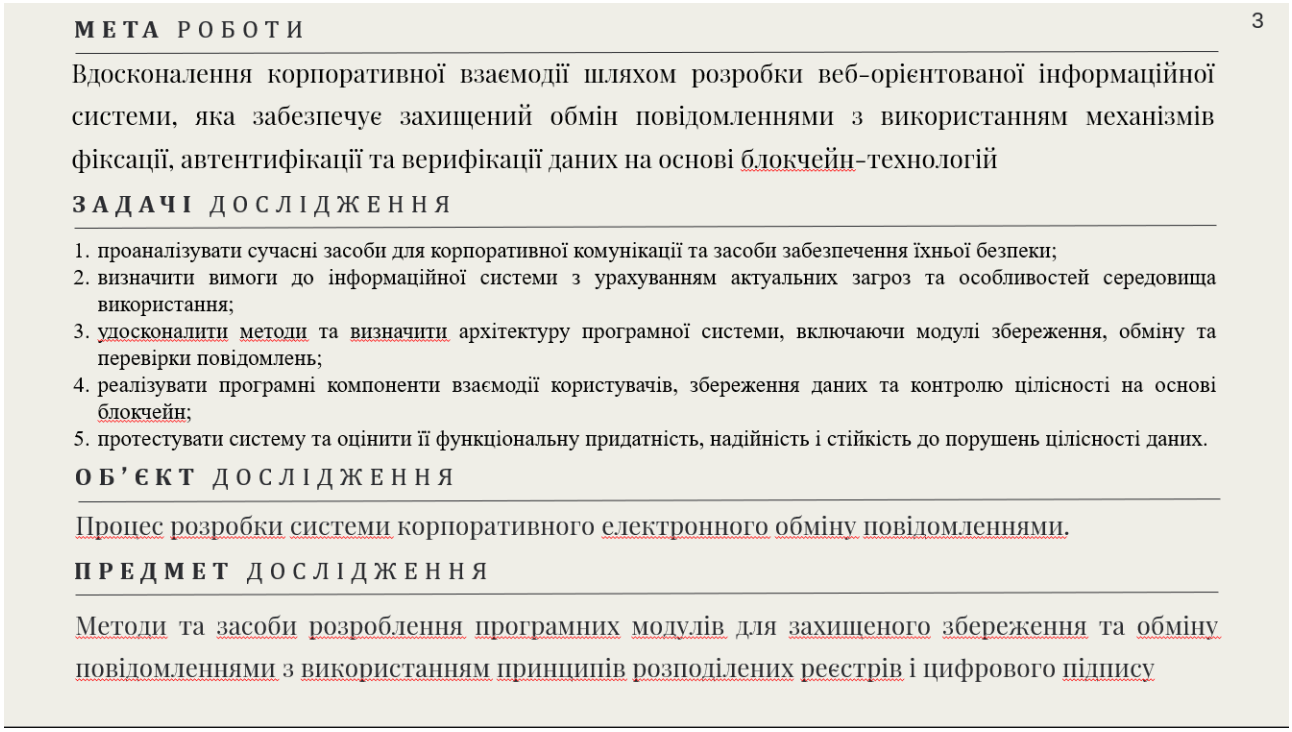


Рисунок Г.3 – Слайд «Мета роботи та задачі дослідження»

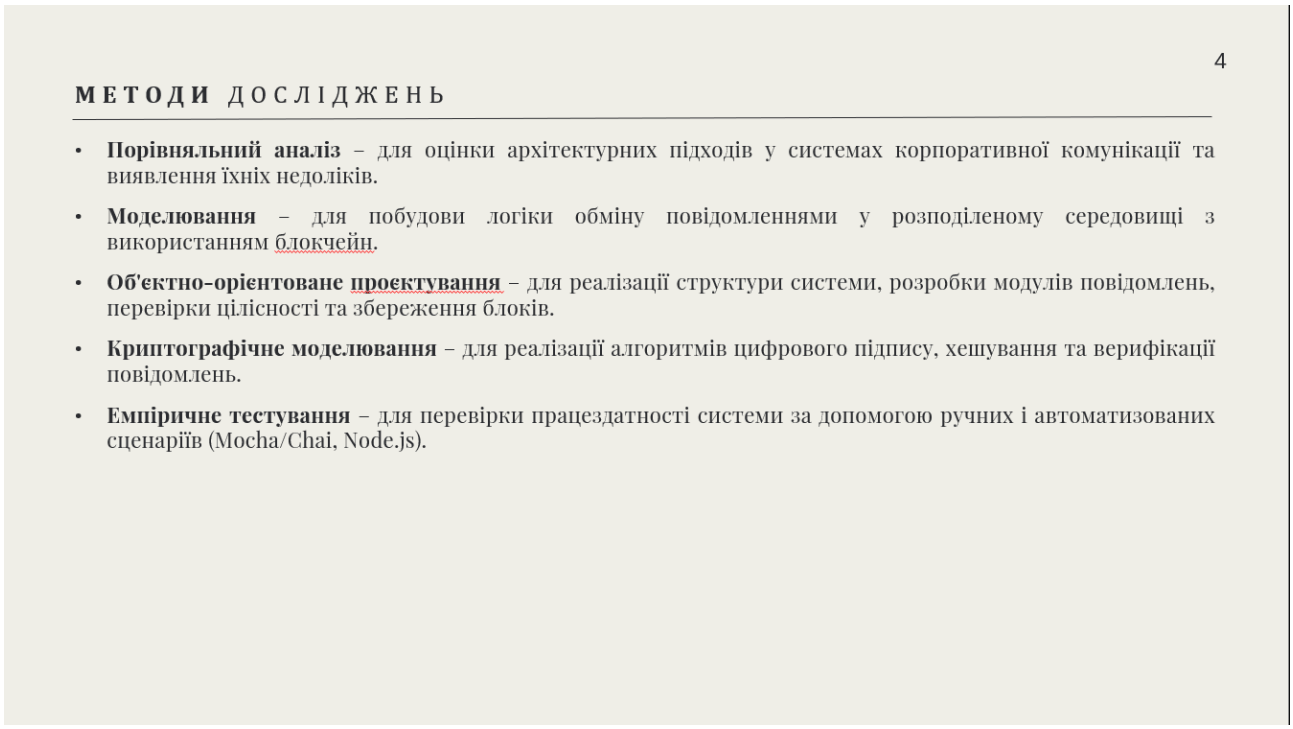


Рисунок Г.4 – Слайд «Методи досліджень»

5

НОВИЗНА ОТРИМАНИХ РЕЗУЛЬТАТІВ

- **Запропоновано** архітектуру програмного забезпечення корпоративної системи комунікацій, яка, **на відміну від** існуючих, базується на приватному блокчейні та забезпечує збереження автентичності повідомлень без використання централізованого сервера, **що дозволяє** підвищити швидкодію здійснення комунікацій з підтримкою високого рівня безпеки.
- **Удосконалено** метод перевірки цілісності повідомлень у розподіленому середовищі, який, **на відміну від** існуючих, використовує методи криптографічного хешування в комбінації з цифровим підписом, **що забезпечує** верифікацію авторства без втрати продуктивності.
- **Подальшого розвитку набули** методи до розробки безпечних вебзастосунків для внутрішніх комунікацій, які, **на відміну від** існуючих, використовують механізми перевірки достовірності повідомлень через блокчейн-записи без залучення проміжних вузлів, **що збільшує** швидкодію без втрати рівня достовірності.

ПРАКТИЧНА ЦІННІСТЬ

Практична цінність отриманих результатів полягає у можливості застосування розробленої інформаційної системи для створення або модернізації корпоративних каналів комунікації, що мають підвищені вимоги до безпеки та достовірності переданої інформації

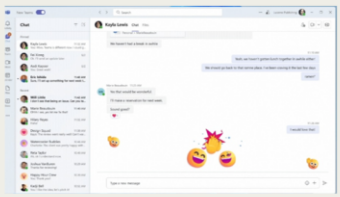
Рисунок Г.5 – Слайд «Новизна отриманих результатів та практична цінність»

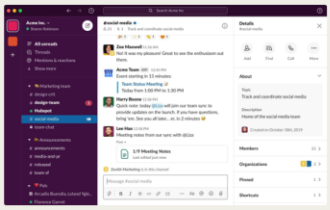
АНАЛІЗ ВІДОМИХ РІШЕНЬ

6

Microsoft teams

- частина екосистеми Microsoft 365, високоякісний відеозв'язок та спільна робота з документами; централізоване зберігання даних на серверах Microsoft без децентралізованої фіксації повідомлень





Slack

- обмін повідомленнями, файлами, інтеграції з ботами та зовнішніми сервісами; централізована архітектура, механізми фіксації подій реалізовані частково через платні доповнення

Mattermost

- Рішення з відкритим кодом для локального розгортання, часткове підвищення безпеки всередині організації; відсутня підтримка блокчейн-фіксації та незмінності історії

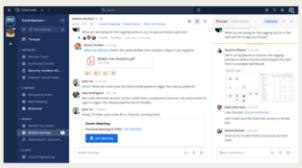


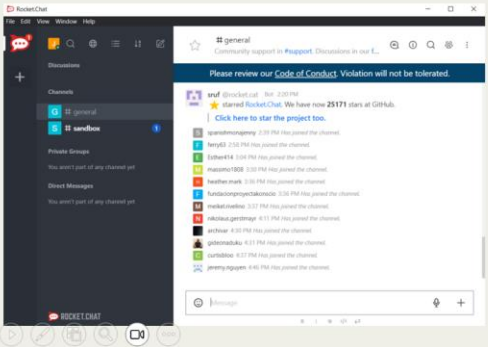
Рисунок Г.6 – Слайд «Аналіз відомих рішень (частина 1)»

АНАЛІЗ ВІДОМИХ РІШЕНЬ

7

Rocket.chat

- кастомізована платформа з базовим шифруванням, централізоване зберігання та управління історією без прозорого журналу дій



Element (Matrix)

- часткова децентралізація через розподілені сервери (homeservers); зберігаються можливості редагування й видалення повідомлень, тому незмінність історії не гарантується

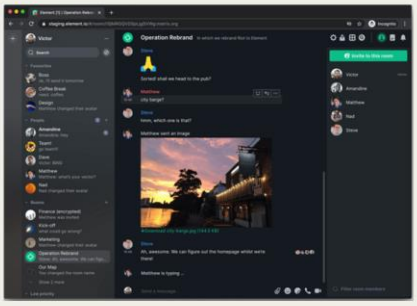


Рисунок Г.7 – Слайд «Аналіз відомих рішень (частина 2)»

ПОСТАНОВКА ЗАДАЧІ НА РОЗРОБКУ

8

У результаті виконання аналітичного огляду предметної області та оцінки актуального стану ринку засобів корпоративної комунікації виявлено потребу у створенні програмного забезпечення, яке б забезпечувало фіксацію обміну повідомленнями з гарантованою автентичністю, цілісністю та незмінністю даних.

Для успішної розробки системи необхідно реалізувати такі завдання:

- спроектувати архітектуру програмного забезпечення, що реалізує визначений функціонал;
- розробити модель структури повідомлень, принципи формування блоків та їх зв'язку у ланцюг;
- реалізувати програмні модулі для надсилання, фіксації та перевірки повідомлень;
- забезпечити формування журналу подій та механізмів верифікації даних;
- здійснити тестування працездатності програмного продукту та оцінити його відповідність сформульованим вимогам.

Рисунок Г.8 – Слайд «Постановка задачі на розробку»

ПРОЕКТУВАННЯ 9

ОБҐРУНТУВАННЯ ВИБОРУ АРХІТЕКТУРИ

Для даної розробки було використано клієнт-серверну архітектуру.

Користувацька взаємодія відбувається через браузерний інтерфейс, що динамічно оновлюється залежно від дій користувача. Обробка запитів здійснюється серверною частиною, яка виконує логіку опрацювання повідомлень, керування обліковими записами та роботу з базою даних. Зберігання інформації реалізовано з використанням документно-орієнтованої моделі, що дозволяє гнучко структурувати записи користувачів, повідомлення та пов'язані з ними дії

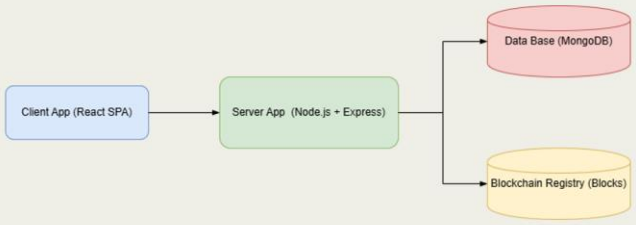
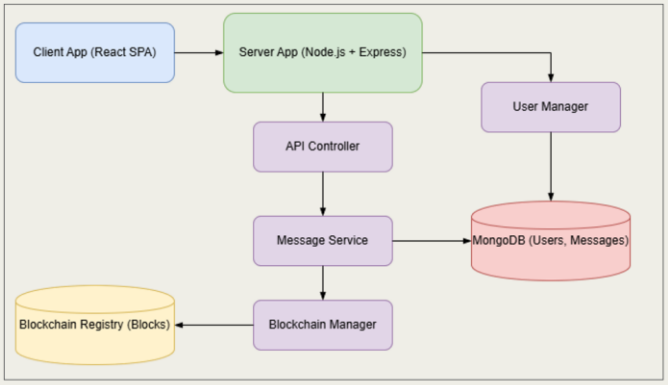


Рисунок Г.9 – Слайд «Проєктування: обґрунтування вибору архітектури»

ПРОЕКТУВАННЯ 10



Структурна UML-діаграма взаємодії компонентів системи

Рисунок Г.10 – Слайд «Проєктування: структурна UML-діаграма»

Рисунок Г.11 – Слайд «Проєктування: структура бази даних»

Рисунок Г.12 – Слайд «Реалізація: вибір інструментів»

ТЕСТУВАННЯ

13

Ручне тестування алгоритмів у Node.js

- Перевірка полягала у запуску скриптів, які хешують повідомлення, генерують та верифікують цифрові підписи, а також перевіряють цілісність ланцюга блоків безпосередньо в консолі. Усі ключові етапи — від зміни навіть одного символу в повідомленні до обробки «підтасованого» блоку — фіксувалися у вигляді текстових повідомлень у виведенні

```
$ node manualTests.js
Original message: "Test message"
Hash: 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
Modified message: "Test message"
Hash: 3f786850e387550fdab836ed7e6dc881de23001b37e91a7715036cbbf873d6a7
Hashes match: false
Key pair generated
Signature: MEUCIQD16Xj4nXc...
Verify signature (correct key): true
Verify signature (wrong key): false
Loading blockchain registry...
Block 0: OK
Block 1: OK
Block 2: OK
Integrity check: false
Error: previousHash mismatch at block 2
```

```
> npm test

> autotests@1.0.0 test
> mocha --reporter spec

  Encryption Module
    ✓ should hash message content correctly (10ms)
  Signature Module
    ✓ should verify valid signatures (8ms)

  MessageController
    ✓ GET /messages returns 200 (12ms)
    ✓ POST /messages creates new message (15ms)

  Blockchain Registry
    ✓ should append new blocks sequentially (20ms)
    ✓ should prevent tampering of previous hash (18ms)

  6 passing (0.12s)
```

Автоматизовані юніт-тести модулів шифрування та фіксації

- Використовуючи фреймворк Mocha/Chai, кожен метод шифрування й генерації підпису перевірявся на очікуваний формат і коректність результату, а тести запускалися автоматично під час складання проєкту

Рисунок Г.13 – Слайд «Тестування»

ВИСНОВКИ

14

Поставлені перед бакалаврською кваліфікаційною роботою задачі виконано в повному обсязі, а саме:

1. Аналіз сучасних засобів корпоративної комунікації та механізмів захисту
   - Проведений детальний огляд популярних платформ (Teams, Slack, Mattermost тощо), виявлені їхні слабкі місця щодо прозорості історії та незмінності даних.
2. Визначення вимог до інформаційної системи
   - Сформульовані функціональні й нефункціональні вимоги: фіксація повідомлень у розподіленому реєстрі, цифровий підпис, відмовостійкість і масштабованість.
3. Проєктування архітектури та моделі даних
   - Створена трирівнева архітектура (React SPA – Node.js – MongoDB + блокчейн-реєстр) і розроблені діаграми прецедентів, розгортання й даних.
4. Реалізація програмних модулів
   - Розроблені сервіси хешування (SHA-256), цифрового підпису (ECDSA), генерації й верифікації блоків, а також REST-API для обміну повідомленнями.
5. Тестування і оцінка працездатності
   - Ручне тестування в консолі та автоматизовані юніт-тести (Mocha/Chai) підтвердили коректність роботи модулів і цілісність блокчейн-ланцюга.

Рисунок Г.14 – Слайд «Висновки 1»

ВИСНОВКИ

15

В результаті виконаної роботи розроблено та досліджено web-систему корпоративних комунікацій з використанням блокчейн-технології, яка відповідає всім функціональним та нефункціональним вимогам. Запропонована архітектура забезпечує прозору незмінність історії повідомлень, а реалізовані модулі хешування, цифрового підпису й блок-реєстру підтвердили свою ефективність під час тестування. Запропоноване рішення дозволяє підвищити безпеку, достовірність і відмовостійкість внутрішньої корпоративної взаємодії та може бути адаптоване для великих організацій із розподіленою інфраструктурою.

Рисунок Г.15 – Слайд «Висновки 2»

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

16

Основні результати роботи було представлено на двох конференціях та опубліковано тези доповіді:

1. 14 Міжнародна науково-практична конференція «Інформаційні технології в освіті та науці»
2. Міжнародна конференція «Молодь в науці 2025».

Впровадження результатів роботи:

Результати роботи планується використати для організації корпоративної комунікації у ФОП «Хомюк Вадим Сергійович».

Рисунок Г.16 – Слайд «Апробація результатів роботи»

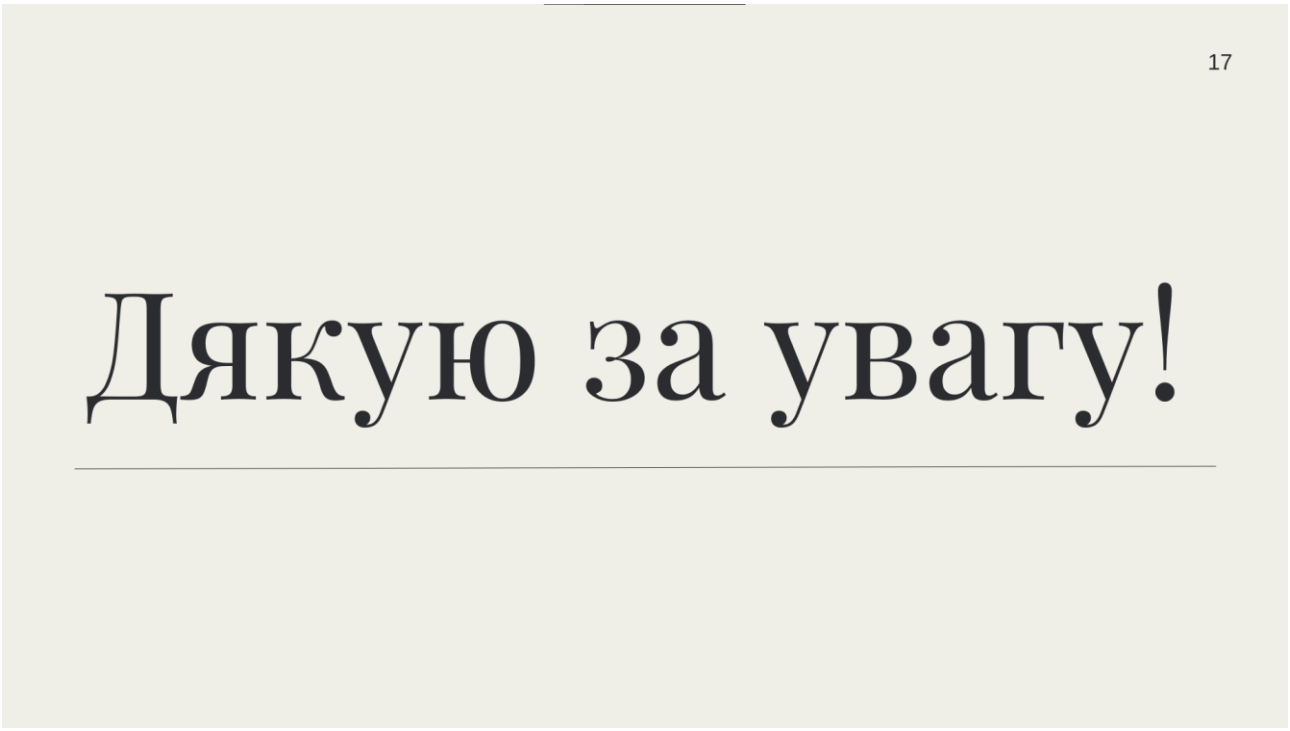


Рисунок Г.17 – Слайд «Подяка за увагу»