Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«РОЗРОБКА ЗАХИЩЕНОГО ВЕБ-РЕСУРСУ ДЛЯ ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ НАСКРІЗНОГО ШИФРУВАННЯ END-TO-END, ДВОФАКТОРНОЇ АУТЕНТИФІКАЦІЇ ТА SIEM-СИСТЕМИ ДЛЯ МОНІТОРИНГУ АКТИВНОСТІ КОРИСТУВАЧІВ»

Виконала: студентка 4 курсу, гр.2КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Осадчук К. Д.
(прізвище та ініціали)

Керівник: д.ф., доц., доцент каф. МБІС
Салієва О. В.
(прізвище та ініціали)
«4» червня 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Савицька Л. А.
(прізвище та ініціали)
«4» червня 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“4” червня 2025р.

Вінниця ВНТУ – 2025

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

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

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Осадчук Катерині Дмитрівні
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка захищеного веб-ресурсу для обміну конфіденційними даними з використанням технології наскрізного шифрування End-to-End, двофакторної аутентифікації та SIEM-системи для моніторингу активності користувачів.

Керівник роботи Салієва Ольга Володимирівна д.ф., доц. каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи
3. Вихідні дані до роботи Електроні джерела, наукові статті, книги, документи, які пов'язані з темою роботи. Доступне програмне забезпечення, технічна документація, вимоги та обмеження.
4. Зміст текстової частини: Для досягнення мети було поставлено задачі, а саме: дослідження обраної теми; дослідження існуючих SIEM-систем, здійснити реалізацію веб-ресурсу, наскрізного шифрування, двофакторної аутентифікації; обрати та інтегрувати систему керування захистом.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень) Алгоритм виконання наскрізного шифрування, алгоритм аутентифікації, зображення інтерфейсу веб-ресурсу (основні сторінки та функціональні елементи), результатати інтеграції SIEM-системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Салієва О. В. д.ф., доц. каф. МБІС		
Розділ II	Салієва О. В. д.ф., доц. каф. МБІС		
Розділ III	Салієва О. В. д.ф., доц. каф. МБІС		

7. Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Ознайомлення з темою кваліфікаційної роботи, аналіз завдань	20.03.2025	30.03.24	Виконано
2	Аналіз сучасних методів захисту веб-ресурсів	31.032025	4.04.2025	Виконано
3	Визначення вимог до функціональності та безпеки веб-ресурсу	5.04.2025	07.04.2025	Виконано
4	Проєктування архітектури веб-ресурсу	08.04.2025	11.04.2025	Виконано
5	Створення бази даних та користувацького інтерфейсу	12.04.2025	18.04.2025	Виконано
6	Реалізація наскрізного шифрування	19.04.2025	25.04.2025	Виконано
7	Впровадження двофакторної аутентифікації	26.04.2025	27.04.25	Виконано
8	Інтеграція SIEM-системи	28.04.2025	15.05.2025	Виконано
9	Оформлення матеріалів до захисту	16.05.2025	30.05.2025	Виконано

Студент _____ Осадчук К. Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Салієва О.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 70 сторінок формату А4, на яких є 16 рисунків, 1 таблиця, список використаних джерел містить 30 найменувань.

Метою роботи є створення веб-ресурсу з підвищеним рівнем захисту для обміну конфіденційною інформацією, у якому реалізовані сучасні засоби інформаційної безпеки, зокрема наскрізне шифрування, двофакторна аутентифікація та використання SIEM-систем для моніторингу дій користувачів.

У загальній частині роботи здійснено аналіз сучасних методів технічного захисту інформації у веб-середовищі, зокрема розглянуто принципи шифрування, засоби аутентифікації та моніторингу дій користувачів. Особливу увагу приділено технології наскрізного шифрування (end-to-end encryption), двофакторній аутентифікації та SIEM-системам як ефективним інструментам забезпечення інформаційної безпеки.

У проєктній частині розроблено концепцію веб-ресурсу з інтеграцією зазначених засобів захисту, зокрема користувацький інтерфейс, модуль шифрування повідомлень, механізм двофакторної аутентифікації та систему моніторингу активності користувачів.

У розділі програмної реалізації описано вибір інструментів розробки, структуру клієнтської та серверної частини, а також наведено реалізацію ключових елементів захисту.

Ключові слова: веб-ресурс, інформаційна безпека, наскрізне шифрування, двофакторна аутентифікація, SIEM-система, конфіденційні дані, шифрування, моніторинг.

ABSTRACT

The bachelor's qualification work consists of 70 pages of A4 format, which contain 16 figures, 1 table, the list of used sources contains 30 items.

The purpose of the work is to create a web resource with an increased level of protection for the exchange of confidential information, which implements modern information security tools, in particular end-to-end encryption, two-factor authentication and the use of SIEM systems for monitoring user actions.

In the general part of the work, an analysis of modern methods of technical information protection in the web environment is carried out, in particular, the principles of encryption, authentication tools and monitoring user actions are considered. Particular attention is paid to end-to-end encryption technology, two-factor authentication and SIEM systems as effective tools for ensuring information security.

In the project part, a concept of a web resource with the integration of the specified protection tools is developed, in particular, a user interface, a message encryption module, a two-factor authentication mechanism and a user activity monitoring system.

The software implementation section describes the choice of development tools, the structure of the client and server parts, and also provides the implementation of key protection elements.

Keywords: web resource, information security, end-to-end encryption, two-factor authentication, SIEM system, confidential data, encryption, monitoring.

ЗМІСТ

ВСТУП 3
1 АНАЛІЗ МЕТОДІВ ЗАХИСТУ: ШИФРУВАННЯ, АУТЕНТИФІКАЦІЯ ТА МОНІТОРИНГ 5
1.1 Аналіз основних принципів захисту веб-ресурсів 5
1.2 Аналіз технології наскрізного шифрування 9
1.3 Двофакторна аутентифікація, як засіб захисту користувачів 12
1.4 Огляд та аналіз SIEM-систем 15
1.5 Висновки до розділу 1 та постановка задач 21
2 ПРОЄКТУВАННЯ ЗАХИЩЕНОГО ВЕБ-РЕСУРСУ ДЛЯ ОБМІНУ ДАНИМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ НАСКРІЗНОГО ШИФРУВАННЯ END-TO-END, ДВОФАКТОРНОЇ АУТЕНТИФІКАЦІЇ ТА SIEM-СИСТЕМИ ДЛЯ МОНІТОРИНГУ КОРИСТУВАЧІВ 22
2.1 Розробка користувацького інтерфейсу 22
2.2 Розроблення підходу наскрізного шифрування повідомлень 24
2.3 Використання двофакторної аутентифікації у веб-ресурсі 27
2.4 Інтеграція SEIM-системи для моніторингу активності користувачів 29
2.5 Висновки до розділу 2 31
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-РЕСУРСУ 32
3.1 Обґрунтування вибору мов програмування, фреймворків, середовища розробки тощо. 32
3.2 Реалізація клієнтської частини 33
3.3 Реалізація серверної частини 40
3.4 Застосування наскрізного шифрування 48
3.5 Реалізація двофакторної аутентифікації 53
3.6 Інтеграція SIEM-системи 55
3.7 Висновки до розділу 3 60
ВИСНОВКИ 61
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 63
Додаток А. Технічне завдання 68
Додаток Б. Лістинг програми (серверна частина) 72
Додаток В. Ілюстративний матеріал 83
Додаток Г. Протокол перевірки на антиплагіат 90

ВСТУП

Актуальність теми. У сучасному цифровому світі особливої уваги набуває проблеми захисту конфіденційних даних при передачі через мережу. Зокрема, під час передачі конфіденційних даних через веб-ресурси користувачі стикаються з великою кількістю ризиків, серед яких: перехоплення даних, несанкціонований доступ, фішинг, підміна особи тощо. Актуальність проблеми зростає у зв’язку з постійним зростанням обсягів чутливої інформації, що передається онлайн, а також появою нових векторів атак з боку зловмисників. Тому важливо забезпечити засоби, що унеможливлять виконання не законних дій.

Поєднання наскрізного шифрування, двофакторної аутентифікації та SIEM-системи буде ефективним та актуальним підходом до створення стійкої системи для веб-ресурсу для обміну конфіденційною інформацією. Такий підхід дозволяє не лише гарантувати конфіденційність даних завдяки шифруванню "від відправника до отримувача", але й значно підвищує рівень контролю за доступом через механізми аутентифікації, що враховують додаткові фактори. Впровадження SIEM-компонента забезпечує постійний аудит, фіксацію підозрілої активності та можливість швидкого реагування на інциденти, що особливо важливо для сучасних веб-систем, які працюють з персональними чи службовими даними. Сукупне використання цих технологій формує комплексний бар’єр для потенційних загроз.

Мета дослідження. Метою даної роботи є розробка захищеного веб-ресурсу для обміну конфіденційними даними, що використовує сучасні засоби інформаційної безпеки, такі як наскрізне шифрування, використання двофакторної аутентифікації та SIEM-систем для моніторингу активності користувачів.

До основних завдань дослідження належать:

– провести аналіз актуальних загроз безпеці веб-додатків та методів їх нейтралізації;

– вивчити та порівняти сучасні засоби реалізації e2ee, 2fa та siem-систем;

– визначити вимоги до архітектури веб-ресурсу;

– обрати відповідні інструменти та технології для реалізації захисту;

– обрати, та спроектувати базу даних;

– реалізувати користувацький інтерфейс;

– здійснити вибір мови програмування, фреймворків, бібліотек та здійснити вибір середовища розробки;

– здійснити програмну реалізацію спроектованого веб-ресурсу для обміну конфіденційними даними з використанням технологій;

– здійснити тестування на функціональність веб-ресурсу.

Об’єкт дослідження. Процес обміну конфіденційною інформацією в мережевому середовищі.

Предмет дослідження. Засоби та методи забезпечення інформаційної безпеки під час передачі даних, а також технічні рішення щодо аутентифікації та моніторингу активності користувачів ресурсів.

Новизна роботи полягає в реалізації комплексного підходу до інформаційної безпеки веб-додатку з акцентом на інтеграцію засобів проактивного захисту. Такий підхід забезпечує не лише збереження конфіденційності, а й підвищення рівня довіри до веб-ресурсів користувачами.

Практична цінність роботи полягає в розробці комплексного рішення, яке застосовується для захисту інформації з обмеженим доступом в веб-додатках, а також використання розробки як бази для подальших наукових, практичних досліджень у сфері кібербезпеки.

Апробація. Тези доповідей даної тематики представленні на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету НТКП ВНТУ (2025) [1] .

1 АНАЛІЗ МЕТОДІВ ЗАХИСТУ: ШИФРУВАННЯ, АУТЕНТИФІКАЦІЯ ТА МОНІТОРИНГ

У даному розділі буде проаналізовано методи захисту веб-ресурсів, зокрема, увага буде зосереджена на розгляд основних принципів захисту, технологій шифрування, аутентифікації, а також огляд систем для моніторингу активності. Це дозволить зрозуміти важливість цих методів у забезпеченні надійного захисту конфіденційних даних.

1.1 Аналіз основних принципів захисту веб-ресурсів

Веб-ресурси сьогодні є основною частиною життя, вони забезпечують доступ до інформації, послуг для більшості користувачів. Проте, якщо подивитися з іншої сторони зростає кількість кібератак, тому захист веб-додатків став надзвичайно актуальним завданням. Ефективні методи забезпечення безпеки мають на меті захистити дані користувачів, гарантувати стабільну роботу сервісів, запобігати спробам несанкціонованого доступу та забезпечувати конфіденційність, цілісність і доступність до інформації.

У цьому розділі розглянуто основні принципи захисту веб-ресурсів, що включають аутентифікацію, криптографічний захист, захист від атак, налаштування політ безпеки, розмежування доступу, програмні підходи до захисту, комплексні, які поєднують в собі кілька способів захисту та інші. Використання будь-яких з цих засобів зменшують ризики пошкодження сервісу, що робить їх захищеними та безпечними у використанні. Розглянемо детальніше деякі з них.

Розмежування доступу є одним із найефективніших способів забезпечення інформаційної безпеки. Даний метод полягає в тому, щоб кожному зареєстрованому користувачу надати можливості безперешкодного доступу до інформації в межах його повноважень і виключити можливості перевищення цих повноважень. Для кожного користувача встановлюються його повноваження щодо файлів, каталогів, логічних дисків та інших системних ресурсів [2].

Підсистема розмежування доступу є одним із головних елементів комплексу безпеки web-ресурсу. За допомогою даного засобу виконується захист від несанкціонованого доступу. При цьому ресурс поділяється на чотири функціональні частини [3]:

– демілітаризована зона, в якому розміщуються сервери порталу, доступ до яких можуть отримати будь-які користувачі мережі Інтернет. До таких серверів відносяться кеш-сервери, публічні Web-сервери і DNS-сервери;

– службові сервери, доступ до ресурсів яких можуть отримати тільки адміністратори або службові сервіси Web-ресурсу;

– сегмент управління, в якому розміщуються засоби, необхідні для управління комплексом безпеки Web-ресурсу;

– комунікаційний сегмент, що включає маршрутизатори і комутатори, що забезпечують взаємодію між іншими сегментами порталу.

Реалізація даного способу виконується на трьох рівнях стека TCP/IP. На канальному рівні відбувається ізоляція частин web-ресурсу за допомогою VLAN. Налаштування комутаторів і списки контролю доступу (ACL) регулюють взаємодію між віртуальними мережами, запобігаючи доступу до інших частин у разі доступу сторонньої особи до сервера. Мережевий рівень фільтрує трафік через мережеві екрани. І прикладний рівень відповідає за контроль доступу на рівні додатків. Цей підхід дозволяє значно зменшити ризики зламу всієї системи.

Після того, як суб’єкт ідентифікує себе, його потрібно аутентифікувати, тобто суб’єкт має довести, що він той, ким себе видає (рис. 1.1). Цей доказ ідентичності досягається шляхом надання облікових даних механізму контролю доступу. Далі перевіряється дійсність наданих облікових даних перед тим, як затвердити запит на аутентифікацію. Іншими словами, аутентифікація визначає, що суб’єкт як володіє, так і контролює надані облікові дані (аутентифікатори). Деякими прикладами облікових даних, які можна використовувати для підтвердження особи, є паролі, PIN-коди, цифрові підписи, біометричні дані тощо [4].

При вході у свої облікові записи більшість людей використовують лише один спосіб підтвердження особи. Зазвичай це – введення логіна і пароля. Проте це недостатньо надійно, особливо якщо як пароль ви використовуєте просте слово чи комбінацію, які хакерам легко зламати [5].

Рисунок 1.1 – Процес ідентифікації, аутентифікації та авторизації [4]

Надійним способом захисту інформаційних систем є використання спеціалізованих програмних засобів. Одним із таких рішень є Тайфун-Web – сучасний інструмент для забезпечення безпеки веб-додатків.

Комплекс засобів захисту Web-ресурсів від несанкціонованого доступу «Тайфун-Web» призначений для криптографічного захисту та розмежування доступу до інформації, оброблюваної в ІС, побудованих на базі Web-технологій [3].

При використанні комплексу «Тайфун-Web» забезпечується [6]:

– взаємна сувора автентифікація (підтвердження справжності) клієнта та сервера за протоколом, побудованим з використанням несиметричних криптографічних алгоритмов;

– захист конфіденційності та цілісності інформації, яка передається між клієнтом та сервером;

– розмежування доступу користувачів до інформаційних ресурсів, представлених у вигляді статичних або динамічних Web-сторінок, які зберігаються та обробляються на відповідних Web-серверах і потребують захисту (захищених Web-ресурсів).

Комплекс реалізує такі основні функції [7]:

– ідентифікацію та автентифікацію користувачів комплексу на основі атрибутів, отриманих від ОС, що дозволяє однозначно встановити певного користувача та у подальшому коректно оброблювати його запити на доступ до захищеної інформації або до засобів адміністрування;

– виділення, на підставі результатів виконаної автентифікації, користувачівадміністраторів, яким надані повноваження із керування засобами комплексу;

– сувору взаємну автентифікацію клієнтської та серверної компонентів комплексу та їхніх користувачів з використанням відповідних протоколів, у яких використовується механізм вироблення / перевірки ЕЦП за алгоритмом, установленим ДСТУ 4145, з використанням відповідних атрибутів;

– керування доступом користувачів до захищених Web-ресурсів, представлених відповідними URL-адресами, на основі атрибутів доступу, призначених спеціально вповноваженими адміністраторами;

– зашифрування/розшифрування інформації, що передається між Web-браузером (або Web-застосуванням), який функціонує на робочій станції (РС) користувача, та захищеним Web-сервером (Proxy-сервером), що забезпечує захист конфіденційності інформації на всьому шляху її передачі по каналах мережі Internet;

– контроль цілісності інформації, що передається між Web-браузером (Webзастосуванням), яке функціонує на РС користувача, та захищеним Web-сервером (Proxусервером), що забезпечує захист від її несанкціонованої модифікації на всьому шляху її передачі по мережі Internet;

– контроль цілісності та самотестування програмних засобів комплексу при старті та в процесі функціонування, що дозволяє забезпечити стійке функціонування засобів захисту та не допустити обробки повідомлень у випадку порушення працездатності;

– протоколювання критичних з погляду захищеності оброблюваної інформації подій у захищеному журналі ОС із забезпеченням можливості аналізу зареєстрованих даних аудита вповноваженими адміністраторами;

– можливість використання для збереження особистих (секретних) ключів користувачів, як незахищених (дискета, flash-drive, і т.д.), так і захищених носіїв (пристрій eToken Pro; SecureToken тощо).

1.2 Аналіз технології наскрізного шифрування

У процесі використання програм, що зосереджені на захисті даних варто говорити про шифрування. Шифрування це один із методів захисту даних, що полягає в перетворені інформації за допомогою ключа так, щоб дані могли читати лише авторизовані особи (рис. 1.2). Ключ – це набір даних, що використовується для шифрування і розшифрування.

Двома основними видами шифрування є симетричне і асиметричне шифрування.

У симетричному шифруванні використовується лише один ключ, і всі сторони користуються одним і тим самим секретним ключом [8].

В асиметричному шифруванні використовується декілька ключів: один ключ для шифрування, а інший для дешифрування (рис. 1.3). Ключ шифрування називають «відкритим ключем», а ключ дешифрування – «закритим ключем» [8].

Рисунок 1.2 – Загальна схема процесу передачі повідомлення

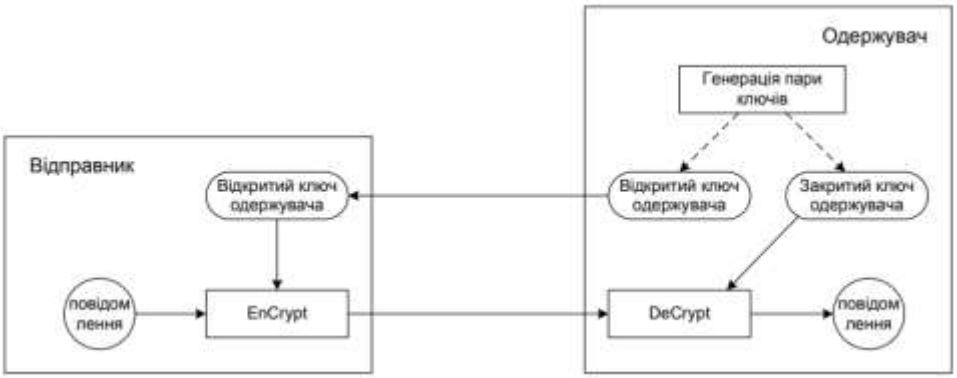


Рисунок 1.3 – Асиметричне шифрування

Сучасною технологією шифрування також є наскрізне шифрування, яке гарантує що дані є захищеними до місця призначення. Тобто тільки користувачі, які спілкуються мають доступ до повідомлень. Даний спосіб захисту вважається найбезпечнішим для обміну повідомлення в інтернеті.

Наскрізне шифрування (E2EE) – це технологія захисту даних, яка гарантує, що інформація залишається доступною лише для відправника та одержувача. Дані шифруються ще на пристрої відправника і розшифровуються лише на пристрої одержувача, без проміжної обробки на серверах. Це забезпечує високий рівень конфіденційності, адже навіть у разі перехоплення доступ до інформації залишається закритим. Жоден провайдер, хакер чи навіть розробники сервісу не зможуть прочитати чи змінити твої повідомлення [9].

Простими словами – повідомлення шифруються, коли відправник їх надсилає, а розшифровуються, коли надходять одержувачу (рис. 1.4). Якщо доставка повідомлення адресату затримується з якихось причин (скажімо, через відсутність зв’язку), воно зберігатиметься на серверах компанії в зашифрованому вигляді, доки не буде доставлене [10].

Рисунок 1.4 – Приклад виконання шифрування

Процес наскрізного шифрування включає кілька кроків[11]:

Крок 1. Шифрування.

E2EE починається з використання алгоритму шифрування для шифрування конфіденційних даних. Цей алгоритм використовує складні математичні функції для перетворення даних у нечитабельний формат, відомий як зашифрований текст. Тільки авторизовані користувачі із секретним ключем, відомим як ключ дешифрування, можуть читати повідомлення.

E2EE може використовувати асиметричну схему шифрування, яка використовує два різні ключі для шифрування та розшифрування даних, або симетричну схему шифрування, яка використовує один спільний ключ для шифрування та розшифрування.

Крок 2. Передача.

Зашифровані дані передаються каналом зв'язку, таким як Інтернет або інші мережі. Повідомлення залишається нечитабельним для серверів додатків, інтернет-провайдерів (ISP), хакерів або інших організацій під час його переміщення до місця призначення. Натомість воно виглядає як випадкові, незрозумілі символи для будь-кого, хто може його перехопити.

Крок 3. Розшифрування.

Після досягнення пристрою одержувача зашифрований текст розшифровується за допомогою закритого ключа одержувача (в асиметричному шифруванні) або спільного ключа (в симетричному шифруванні). Тільки одержувач має закритий ключ, необхідний для розшифрування даних.

Крок 4. Аутентифікація.

Розшифровані дані перевіряються для забезпечення їх цілісності та автентичності. Цей крок може включати перевірку цифрового підпису відправника або інших облікових даних, щоб підтвердити, що ніхто не підробляв дані під час передачі.

Хоча даний метод має недоліки, а саме: складність у впровадженні та управлінні, потенціал зловживань, вразливість для кінцевих точок, але він має

переваги, що роблять потужним інструментом для забезпечення конфіденційності, цілісності та доступності.

Серед переваг використання даного методі шифрування можна виділити такі, як підвищена конфіденційність, захист від витоку даних, зниження ризику шпигунства, стійкість до втручання, покращене управління щодо дотримання вимог, захист від компрометації передавання даних, сервера та центру обробки даних тощо [12].

Роль наскрізного шифрування і його сенс зростають лише тоді, коли він переходить у все більш зв'язаний цифровий світ. Розуміння нюансів, сильних сторін та обмежень E2EE є надзвичайно важливим. Крім того, бажання розробити та впроваджувати інновації в цьому криптографічному методі відіграє ключову роль у розробці безпечного, приватного цифрового майбутнього.

1.3 Двофакторна аутентифікація, як засіб захисту користувачів

Як було описано вище аутентифікація – це процес перевірки користувача чи дійсно він є тим за кого себе видає. Щоб збільшити безпечність використання більшість технологій використовує двофакторну аутентифікацію, оскільки він залучає не лише один фактор для перевірки.

Тобто, двофакторна автентифікація, або скорочено 2FA – це різновид багатофакторної автентифікації, яка грунтується на підтвердженні особистості користувача за допомогою двох, незалежних один від одного, факторів [13].

Типи аутентифікаї поділяються за категоріями, а саме: щось, що ви знаєте (пароль, PIN-код); те, що у вас є (смарт-картка, мобільний пристій); і те, чим ви є (відбитки пальців, розпізнавання обличчя, голосу) [14].

Варіантів методів та способів реалізації двофакторної аутентифікації є досить багато, серед найпоширеніших варто виокремити:

1. Одноразові паролі (OTP) через SMS або email.

OTP - це пароль, дійсний лише для одного сеансу входу до системи або транзакції на комп'ютерній системі або іншому цифровому пристрої. Зазвичай він генерується автоматично сервером автентифікації та надсилається

користувачеві SMS або електронною поштою. Потім користувач вводить OTP на сторінці входу до системи, щоб отримати доступ до системи або завершити транзакцію (рис. 1.5) [15].

Рисунок 1.5 – Приклад SMS коду для підтвердження

Незалежно від виду одноразового паролю мета його залишається та сама, а саме підтвердження користувача, що хоче отримати доступ до сервісу. Це дає можливість зменшити ризик несанкціонованого доступу, оскільки при перевірці зловмисник може не мати доступу до мобільного телефону або електронної пошти особи.

Цей спосіб є достатньо дієвим, але він має недолік, такий як залежність від мобільного пристрою. У разі втрати чи крадіжки процес відновлення доступу до ресурсу може бути ускладненим або взагалі не можливим. Крім того, існує ризик збереження паролів на телефоні, наприклад у нотатках, і можливість використання одного паролю до кількох сервісів, тоді у разі отримання зловмисником доступу до цих даних, він потенційно має можливість авторизації у середовищах, зокрема в електронній пошті.

2. Двофакторна аутентифікація через програмне забезпечення.

Даний метод базується на тому, що необхідно встановити програмне забезпечення для отримання коду. Програма генерує коди для користувача на короткий період часу. Який необхідно ввести для успішного входу користувача. Прикладом таких програм є Google Authenticator, Authy, Microsoft Authenticator.

Наприклад, Google Authenticator – це безкоштовний додаток для вашого смартфона, який генерує новий код кожні 30 секунди (рис. 1.6). Це працює так: після ввімкнення двофакторної аутентифікації (2FA) програма, яку ви захищаєте, генерує QR-код, який користувач може відсканувати камерою телефону, щоб додати профіль до програми Google Authenticator.Потім смартфон вашого

користувача генерує новий код кожні 30 секунд, який використовується для другої фази автентифікації в програмі [16].

Рисунок 1.6 – Приклад відображення кодів у програмі Google Authenticator

Даний спосіб має переваги щодо його використання. Будь-які програмне забезпечення такого роду мають зрозумілий для простого користувача інтерфейс, що надає легкість у використанні. Також коди генеруються автоматично у додатку, що забирають потребу постійно вводити дані для отримання коду. Кросплатформова підтримка – програма працює і на комп'ютері, і на смартфоні. Навіть якщо ви втратили смартфон, ви все одно зможете згенерувати код на комп'ютері та змінити необхідні дані за потреби.

3. Біометричні дані.

Біометрична аутентифікація – це процес організації безпеки, який ґрунтується на технології розпізнавання персони за її біологічними характеристиками. Їхня унікальність дає можливість підтвердити, чи є людина саме тим користувачем, за якого себе видає [17].

Використання біометричних даних є буденністю майже для кожного. Найпоширенішим прикладами є розблокування телефону відбитком пальця або розпізнавання обличчя, авторизація у мобільному банкінгу або створення електронного підпису у додатку ДІЯ.

До переваг даного методу відноситься надійність та швидкість; безпечність, оскільки повністю скопіювати голос, відбитки чи інші ідентифікатори людини не можливо; біометричні дані нереально забути чи

втратити. Однак через вікові зміни, травми тощо потрібно слідкувати за постійним оновленням еталонних моделей.

Отже, двофакторна аутентифікація є ефективним способом захисту від несанкціонованого доступу. Обираючи необхідний фактор аутентифікації, що забезпечить потреби організації чи ресурсу який її використовує, можна забезпечити баланс між безпекою та зручністю.

При аналізі методів 2FA було вирішено обрати додаткову аутентифікацію за допомогою одноразового паролю за SMS або листом на електрону пошту, що забезпечить додатковий рівень захисту, поєднуючи ефективність з мінімальними витратами.

1.4 Огляд та аналіз SIEM-систем

Керування захистом інформації (SIEM) – це рішення, що допомагає організаціям виявляти, аналізувати й усувати загрози безпеці, перш ніж вони зможуть зашкодити бізнес-операціям. SIEM поєднує інструменти для керування інформаційною безпекою, а також засоби для керування подіями безпеки в одне рішення. Воно допомагає краще керувати системою безпеки. Технологія SIEM збирає дані журналу подій із низки джерел, визначає нетипові дії за допомогою аналізу в реальному часі й уживає відповідних заходів [18].

Простими словами ця система у реальному часі на основі визначених правил збирає, аналізує дані, що збираються від програми, сервісу, користувачів, щоб фахівці, які відповідають за безпеку могли передбачити та заблокувати атаки, несанкціоновані дії тощо.

Функціональність [19]:

– агрегація даних: керування журналами даних; дані збираються з різних джерел: мережеві пристрої та послуги, датчики систем безпеки, сервери, бази даних, додатки; забезпечується консолідація даних із метою пошуку критичних подій;

– кореляція: пошук загальних атрибутів, зв'язування подій у значні кластери. Технологія забезпечує застосування різних технічних прийомів для

інтеграції даних із різних джерел перетворення вихідних даних на значну інформацію. Кореляція є типовою функцією підмножини Security Event Management;

– оповіщення: автоматизований аналіз корелюючих подій та генерація сповіщень (тривог) про поточні проблеми. Оповіщення може виводитися на «приладову» панель самої програми, так і бути направлено в інші канали: e-mail, GSM-шлюз і т.п;

– засоби відображення (інформаційні панелі): відображення діаграм, які допомагають ідентифікувати патерни, відмінні від стандартної поведінки;

– сумісність (трансформованість): застосування додатків для автоматизації збору даних, формування звітності для адаптації даних, що агрегуються, до існуючих процесів управління інформаційною безпекою та аудиту;

– зберігання даних: застосування довгострокового сховища даних в історичному порядку для кореляції даних за часом та забезпечення трансформації. Довготривале зберігання даних є критичним для проведення комп'ютерно-технічних експертиз, оскільки розслідування мережного інциденту навряд чи проводитиметься в момент порушення;

– експертний аналіз: можливість пошуку за безліччю журналів на різних вузлах; може виконуватися у межах програмно-технічної експертизи.

Крім базового функціоналу до якого відноситься нормалізація, класифікація та кореляція журналів подій безпеки, централізованого збору SIEM-системи мають можливість підтримки управління ризиками, вразливостями, формування звітів, інтеграція з іншими платформами. Дані можливості дозволяють автоматизовувати дії реагування на загрози без участі людини.

При пошуку SIEM можна знайти безліч різних платформ, які пропонують різні можливості, підтримку на різноманітних операційних системах тощо. Для прикладу розглянемо кілька систем.

a) Wazuh.

Для управління інформацією та подіями безпеки – це централізована платформа для агрегації та аналізу телеметрії в режимі реального часу з метою виявлення загроз та забезпечення відповідності вимогам. Wazuh збирає дані про події з різних джерел, таких як кінцеві точки, мережеві пристрої, хмарні робочі навантаження та програми, для ширшого охоплення безпеки [20].

Тобто це платформа SIEM з відкритим кодом, яка забезпечує збір, зберігання, аналіз подій безпеки для виявлення аномалій, індикаторів компрометації та загроз. Вона використовує контекстуалізацію даних для пришвидшення розслідувань, скорочення часу реагування та надає можливості реального часу для моніторингу інцидентів безпеки. За допомогою вбудованого механізму виявлення вразливостей та агентів, встановлених на кінцевих точках, Wazuh дозволяє пріоритизувати загрози, зменшувати поверхню атак і своєчасно усувати критичні ризики.

Система підтримує відповідність сучасним стандартам безпеки, таким як NIST 800-53, GDPR, HIPAA тощо, використовуючи SCA (Security Configuration Assessment) та сканування на відповідність рекомендаціям CIS. Wazuh забезпечує створення налаштовуваних панелей управління, гнучких звітів та сповіщень, що дозволяє командам ІБ адаптувати систему до власних вимог і демонструвати дотримання нормативних вимог. а тепер те ж саме тільки перефразовано більш своїми словами.

b) Graylog.

Graylog Security – це сучасна SIEM-платформа, створена без компромісів, яка усуває типові обмеження традиційних рішень: від непрозорого ціноутворення до надмірного шуму в сповіщеннях. Вона дозволяє зберігати великі обсяги логів із гнучким доступом лише до потрібної інформації, забезпечуючи контроль над витратами без втрати контексту. Завдяки вбудованому Data Lake, попередньому перегляду логів і механізму Selective Log Retrieval, команда безпеки отримує повну видимість без неочікуваних рахунків [21].

Система зосереджена на прискоренні розслідувань, автоматизації рутинних задач і мінімізації хибнопозитивів. Graylog використовує контекстну оцінку ризиків на базі Adversary Campaign Intelligence, пріоритетизацію подій за критичністю активів і швидке переключення між пов'язаними інцидентами. Завдяки підтримці хмарних, локальних та гібридних середовищ, фіксованому ліцензуванню та готовим до використання сценаріям – це ефективне рішення як для аналітиків, так і для IT-керівництва [21].

c) Splunk

Splunk Enterprise Security – це провідна SIEM-платформа, яка забезпечує централізовану аналітику безпеки, автоматизоване виявлення загроз і ефективне реагування на інциденти. Завдяки здатності збирати, нормалізувати й аналізувати дані з будь-яких джерел у режимі реального часу, Splunk надає повну видимість подій в IT-інфраструктурі, що критично важливо для ефективної роботи центрів кібербезпеки (SOC). Використання інтелектуальних оповіщень на основі ризиків (RBA) дозволяє суттєво зменшити кількість помилкових спрацювань і зосередити увагу на дійсно небезпечних інцидентах [22].

Інтеграція з платформою Splunk SOAR забезпечує автоматизовану обробку подій безпеки, скорочуючи час виявлення (MTTD) та реагування (MTTR). Уніфікована платформа дозволяє аналітикам працювати в єдиному середовищі – від виявлення до розслідування й реагування. Завдяки готовим сценаріям, відповідним MITRE ATT&CK, підтримці вбудованих засобів виявлення загроз і великій екосистемі партнерських рішень, Splunk підходить як для великих організацій, так і для підприємств, які прагнуть масштабованого й надійного рішення у сфері кіберзахисту [22].

d) Logpoint SIEM

Logpoint SIEM – це сучасна система керування інформацією та подіями безпеки (SIEM), яка забезпечує високий рівень спостережуваності та точного аналізу загроз. Вона централізує збір та моніторинг даних з усіх пристроїв, додатків і кінцевих точок, нормалізує події у власну таксономію та збагачує їх контекстом (геолокація, LDAP, розвідка загроз тощо). Це дозволяє аналітикам

швидко виявляти інциденти, розслідувати їх за MITRE ATT&CK та приймати обґрунтовані рішення для реагування [23].

Система постачається з готовими звітами та інструментами для відповідності нормативним вимогам (GDPR, NIS2, GPG13), підтримує автоматизацію TDIR, має гнучке розгортання, а також не потребує складної інтеграції. Рішення Logpoint доповнюється AgentX – вбудованим агентом, що забезпечує глибоку видимість подій на кінцевих точках і виявлення загроз у реальному часі. Завдяки цьому Logpoint перетворює журнали подій на потужний інструмент безпеки, забезпечуючи швидке реагування та зменшення ризиків ще до настання шкоди [23].

e) Securonix

Securonix – це уніфікована хмарна SIEM-платформа нового покоління, яка поєднує аналітику безпеки, автоматизоване реагування та інтелектуальне виявлення загроз в одному рішенні. Завдяки інтеграції компонентів SIEM, UEBA, SOAR і TDIR, Securonix забезпечує повну прозорість, здатність до масштабування та автоматизацію, що дозволяє організаціям ефективно виявляти та нейтралізувати сучасні кібератаки. Архітектура платформи розрахована на обробку понад мільйона подій за секунду без втрати продуктивності, а аналітика на базі загроз із Threat Labs дає змогу значно знизити кількість хибнопозитивних спрацювань [24].

Ключовою перевагою Securonix є її модель «як послуга», що усуває потребу у витратах на інфраструктуру та складному адмініструванні, дозволяючи сконцентруватися на зміцненні захисту. Завдяки готовому контенту, заснованому на реальних сценаріях загроз, компанії можуть прискорити впровадження системи, оптимізувати витрати та значно підвищити рівень кіберзахисту [24].

Для узагальнення всіх описаних SIEM-систем сформуємо таблицю 1.1.

Таблиця 1.1 – Порівняння SIEM-систем

SIEM-система	Тип платформи	Архітектура/ розгортання	Основні можливості	Переваги
Wazuh	Open-source, безкоштовна	Локально, гібридна	Збір логів, IDS, SCA, виявлення вразливостей, відповідність вимогам	Безкоштовна, модульна, підтримує інтеграцію з Elastic Stack
Graylog	Freemium (Community + платна Enterprise)	Локально, хмара, гібридна	Selective Log Retrieval, Data Lake, Adversary Campaign Intelligence, автоматизація	Просте ліцензування, гнучке управління логами, мінімізація шуму
Splunk	Комерційна (безкоштовна пробна)	Локально, хмара, гібридна	RBA, UEBA, інтеграція з SOAR, MITRE ATT&CK, повна уніфікація процесів	Потужна аналітика, масштабованість, велика екосистема
Logpoint	Комерційна (є free trial)	Локально, хмара	MITRE ATT&CK, AgentX, автоматизація TDIR, нормалізація логів	Висока швидкість аналізу, просте розгортання, відповідність стандартам
Securonix	SaaS (хмарна, комерційна, trial доступний)	Повністю хмарна	UEBA, SOAR, TDIR, Threat Labs, масштабованість, понад 1 млн EPS	Мінімальна інфраструктура, готові сценарії, інтелектуальна аналітика
IBM QRadar	Комерційна (trial доступний)	Локально, хмара, гібридна	AI-аналіз, потокова обробка, глибокі кореляції, QNI, інтеграція з XDR	Висока точність кореляцій, перевірене рішення, велика база Threat Intelligence
Microsoft Sentinel	Комерційна (оплата за лог/час, trial)	Хмарна	Native SOAR, ML-аналітика, інтеграція з Microsoft 365/Azure, MITRE ATT&CK, Jupyter Playbooks	Швидке впровадження, глибока інтеграція з Microsoft, зниження TCO

Таким чином, кожна з розглянутих SIEM-систем має свої особливості, переваги й обмеження. Вибір оптимального рішення залежить від потреб організації, її ресурсів, рівня загроз і вимог до відповідності стандартам безпеки. Важливо також враховувати можливість масштабування, підтримку автоматизації та співвідношення ціни й функціональності.

1.5 Висновки до розділу 1 та постановка задач

У результаті аналізу методів захисту було встановлено , що для ефективного створення веб-ресурсу для обміну даними має базуватися на поєднанні кількох ключових складових для досягнення надійності. Наскірзне шифрування забезпечує конфіденційність під час передачі даних, двофакторна аутентифікація дозволить значно зменшити ризики несанкціонованого доступу, а застосування SIEM-системи забезпечить моніторинг подій, що допоможе своєчасному виявленню аномалій та реагування на інциденти.

Опираючись на результати проведеного аналізу та виходячи з мети і актуальності обраної теми, були поставлені наступні задачі для подальшої роботи:

– спроектувати інформаційну структуру, базу даних та інтерфейс користувача;

– забезпечити тестування розробленого веб-ресурсу з метою перевірки його функціональності;

– визначити архітектуру та функціональні вимоги до веб-ресурсу;

– здійснити вибір мови програмування, фреймворків, бібліотек, середовища розробки та відповідних програмних компонентів;

– впровадити механізми E2EE, 2FA та SIEM у веб-додаток відповідно до обраної архітектури;

– провести тестування функціональності ресурсу.

2 ПРОЄКТУВАННЯ ЗАХИЩЕНОГО ВЕБ-РЕСУРСУ ДЛЯ ОБМІНУ ДАНИМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ НАСКРІЗНОГО ШИФРУВАННЯ END-TO-END, ДВОФАКТОРНОЇ АУТЕНТИФІКАЦІЇ ТА SIEM-СИСТЕМИ ДЛЯ МОНІТОРИНГУ КОРИСТУВАЧІВ

У даному розділі роботи розглянуто процес розробки алгоритмів роботи веб-ресурсу, побудовано відповідні блок-схеми. Також здійснено вибір, щодо мов програмування, спроектовано базу даних та користувацький інтерфейс, що буде легким та зручним для використання.

2.1 Розробка користувацького інтерфейсу

Користувацький інтерфейс розроблено з акцентом на функціональність, зручність використання та інтуїтивно зрозумілу навігацію. Одним із головних завдань було створити інтерфейс, який буде максимально простим та ефективним для користувача, без зайвих елементів, що можуть відволікати або ускладнювати взаємодію.

Основна увага була приділена структурі сторінок, розміщенню елементів та їх візуальному оформленню. Всі ці аспекти повинні бути такими, щоб користувач міг легко орієнтуватися у веб-ресурсі та швидко знаходити необхідні функції. У цьому контексті, велике значення має також вибір кольорової гами та шрифтів, що дозволяє підвищити читабельність і зробити ресурс приємним для сприйняття.

Для створення інтерфейсу спершу розроблено макет, що відображає основні елементи. Наприклад, для форм входу та реєстрації була створена спільна структура з невеликою відмінністю в кількості полів, що робить кожну форму зрозумілою для користувача (рис. 2.1).

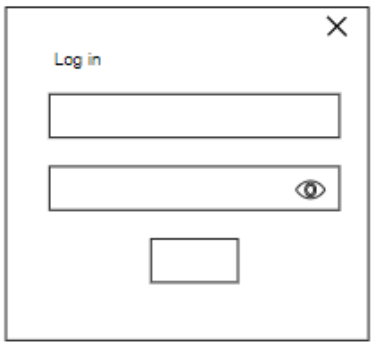


Рисунок 2.1 – Макет форми входу

Також веб-ресурс включає кілька сторінок, а саме: основна, сторінка з чатами та особисті повідомлення. Для кожної сторінки також створюється мінімалістичний макет, що відображає розташування елементів. Наприклад, сторінка з чатами містить кнопку що відображатиме меню, поле пошук та перелік чатів, що дозволяє здійснити перехід до особистих повідомлень (рис. 2.2).

Рисунок 2.2 – Макет сторінки з чатами

Для інших сторінок та елементів ресурсу створюється відповідний макет, який у подальшому буде застосований під час реалізації інтерфейсу. Це дозволяє не лише заздалегідь оцінити дизайн і зручність інтерфейсу, але й забезпечити гармонійне поєднання всіх елементів на різних сторінках ресурсу.

Такий підхід забезпечує цілісність дизайну, спрощує процес верстки та дає можливість вже на етапі макетування врахувати потенційні проблеми з навігацією та взаємодією користувача з ресурсом.

Крім того, розроблені макети дозволяють тестувати різні варіанти розташування елементів та визначати найбільш ефективні варіанти взаємодії

користувача з інтерфейсом ще до початку реалізації на практиці. Це допомагає зберегти час на подальших етапах розробки та забезпечує більш високий рівень зручності для кінцевого користувача.

2.2 Розроблення підходу наскрізного шифрування повідомлень

Після аналізу сучасних засобів захисту текстових повідомлень у мережевих сервісах можна зробити висновок, що для забезпечення конфіденційності та цілісності під час обміну між користувачами можна реалізувати через наскрізне шифрування. Такий підхід гарантує, що повідомлення можуть бути прочитані виключно відправником та отримувачем, незалежно від шляху проходження мережею чи участі серверу, як посередника.

На відміну від класичних підходів, де повідомлення обробляються на стороні сервера, у E2EE архітектурі весь процес шифрування та дешефрування відбувається виключно на клієнтських пристроях. Сервер виступає лише як транспортувальний посередник, який немає доступу до змісту повідомлень.

Запропонований механізм наскрізного шифрування ґрунтується на принципах гібридного криптографічного підходу, що поєднує переваги як асиметричних, так і симетричних алгоритмів. Асиметричне шифрування використовується для захищеного обміну сеансовими симетричними ключами, тоді як основне повідомлення шифрується за допомогою швидкого та ефективного симетричного алгоритму AES.

Така комбінація дозволяє досягти високої продуктивності при передачі великих обсягів даних і водночас гарантувати надійність та конфіденційність за рахунок криптостійкості асиметричних методів. У результаті забезпечується безпечна передача повідомлень, недоступна для читання або модифікації з боку сторонніх осіб чи серверної частини системи.

Для приблизного відображення цього процесу побудуємо схему обміну повідомленнями між користувачами A ти B, що наведена на рисунку 2.3.

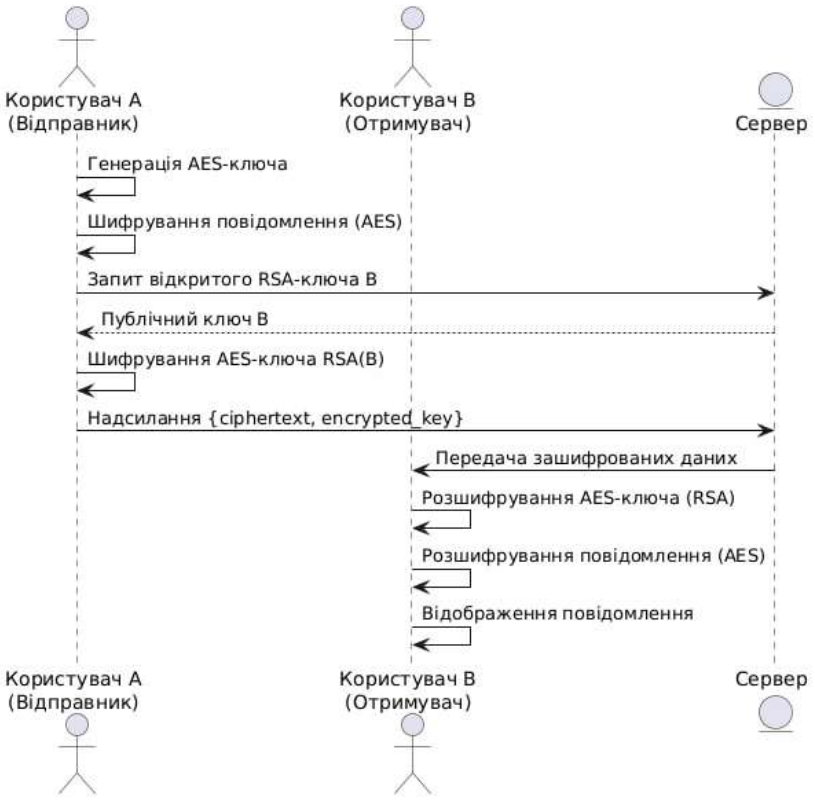


Рисунок 2.3 – Процес обміну повідомленнями

Описати порядок дій можна у такій послідовності.

Крок 1. На етапі реєстраці або під час створення першого входу користувача до системи, на його пристрої генерується криптографічна пара RSA-ключів: відкритий та закритий. Відкритий ключ призначений для вільного поширення та буде використовуватися іншими користувачами для шифрування повідомлень. Закритий ключ використовується виключно для розшифрування даних і не залишатиме межі клієнтського пристрою. Для безпеки зберігання приватного ключа можливе застосування локального шифрування – майстер-пароля або апаратного сховища.

Крок 2. Після генерації ключів, відкритий ключ передається на сервер і прив’язується до облікового запису користувача (індитифікатора). Таким чином, коли інший користувач захоче надіслати повідомлення, він отримає цей відкритий ключ із сервера. Сервер не має доступу до закритого ключа, тому не може розшифрувати жодне повідомлення. Роль сервера полягає у даному випадку тільки у зберіганні відкритих ключів для користувачів і надавати їх за запитом.

Крок 3. Коли користувач хоче надіслати повідомлення іншому користувачу, на його пристрої створюється випадковий сеансовий ключ симетричного шифрування (AES). Повідомлення шифрується цим ключем за допомогою алгоритму AES у режимі GCM. Цей режим забезпечує як конфіденційність, так і автентичність повідомлення. В результаті шифрування створюються зашифрований текст, ініціалізаційний вектор та тег автентифікації, що забезпечує перевірку цілісності.

Крок 4. Сеансовий AES-ключ, яким шифрувалося повідомлення, сам по собі шифрується відкритим RSA-ключем одержувача. Це дозволяє гарантувати, що лише конкретний отримувач зможе отримати доступ до самого ключа і, відповідно, до вмісту повідомлення. Навіть якщо зашифроване повідомлення буде перехоплено, жодна інша особа, окрім власника відповідного приватного ключа, не зможе його розшифрувати.

Крок 5. Далі формується спеціальна структура повідомлення, яка надсилається на сервер. Вона включає ідентифікатори відправника й одержувача, зашифроване повідомлення, зашифрований AES-ключ, ініціалізаційний вектор та тег автентифікації. За потреби до цієї структури також може бути додано цифровий підпис, створений приватним ключем відправника. Це дає змогу підтвердити, що повідомлення справді надійшло саме від того, хто його відправив. Сервер не може прочитати або змінити повідомлення – він просто передає його одержувачу.

Крок 6. Коли одержувач отримує повідомлення, на його клієнтській стороні розпочинається процес дешифрування. Спочатку, використовуючи власний закритий RSA-ключ, він розшифровує зашифрований AES-ключ. Потім за допомогою цього симетричного ключа, IV і тегу автентичності GCM, повідомлення розшифровується. Якщо до повідомлення було додано цифровий підпис, система додатково перевіряє справжність відправника, звіряючи підпис з відкритим ключем відправника.

Усі дії з шифрування та розшифрування відбуваються виключно на клієнтських пристроях, що відповідає принципу наскрізного шифрування. Це

означає, що навіть сервер, який передає повідомлення, не може бачити його зміст. У випадку перехоплення повідомлення третя сторона не зможе його дешифрувати без приватного ключа одержувача.

Завдяки чіткій структурі криптографічних операцій та обмеженню довіри до сервера, система забезпечує надійний захист даних при передачі. Такий підхід ідеально підходить для систем обміну повідомленнями, де важлива приватність.

2.3 Використання двофакторної аутентифікації у веб-ресурсі

Для підвищення рівня захисту облікових записів користувачів у веб-ресурсі впроваджено механізм двофакторної аутентифікації. Даний підхід передбачає використання двох факторів: знання (пароль) та володіння (доступ до поштової скриньки), що дозволяє зменшити ризики несанкціонованого доступу навіть у випадку компрометації основного пароля.

Дану частину можна поділити на такі частини, як реєстрація користувача, вхід у систему з 2FA, захист механізму, поведінка при помилках.

Під час створення облікового запису користувач проходить такі етапи:

– заповнення реєстраційної форми. Тут особа вказує ім’я, електрону пошту та обирає пароль;

– перевірка системою чи не існує такий користувач у базі;

– для реалізації E2EE може бути згенерована пара ключів (відкритий і закритий), де відкритий ключ зберігається на сервері, а приватний – локально на пристрої користувача;

– після успішної реєстрації активується акаунт і користувач матиме можливість входити в систему за цими даними в подальшому.

Процес входу до системи з двофакторною аутентифікацією включає два етапи. Перший етап базова аутентифікація та другий – перевірка другого фактору. Блок-схема на рисунку 2.4 відображає повний процес як це відбувається.

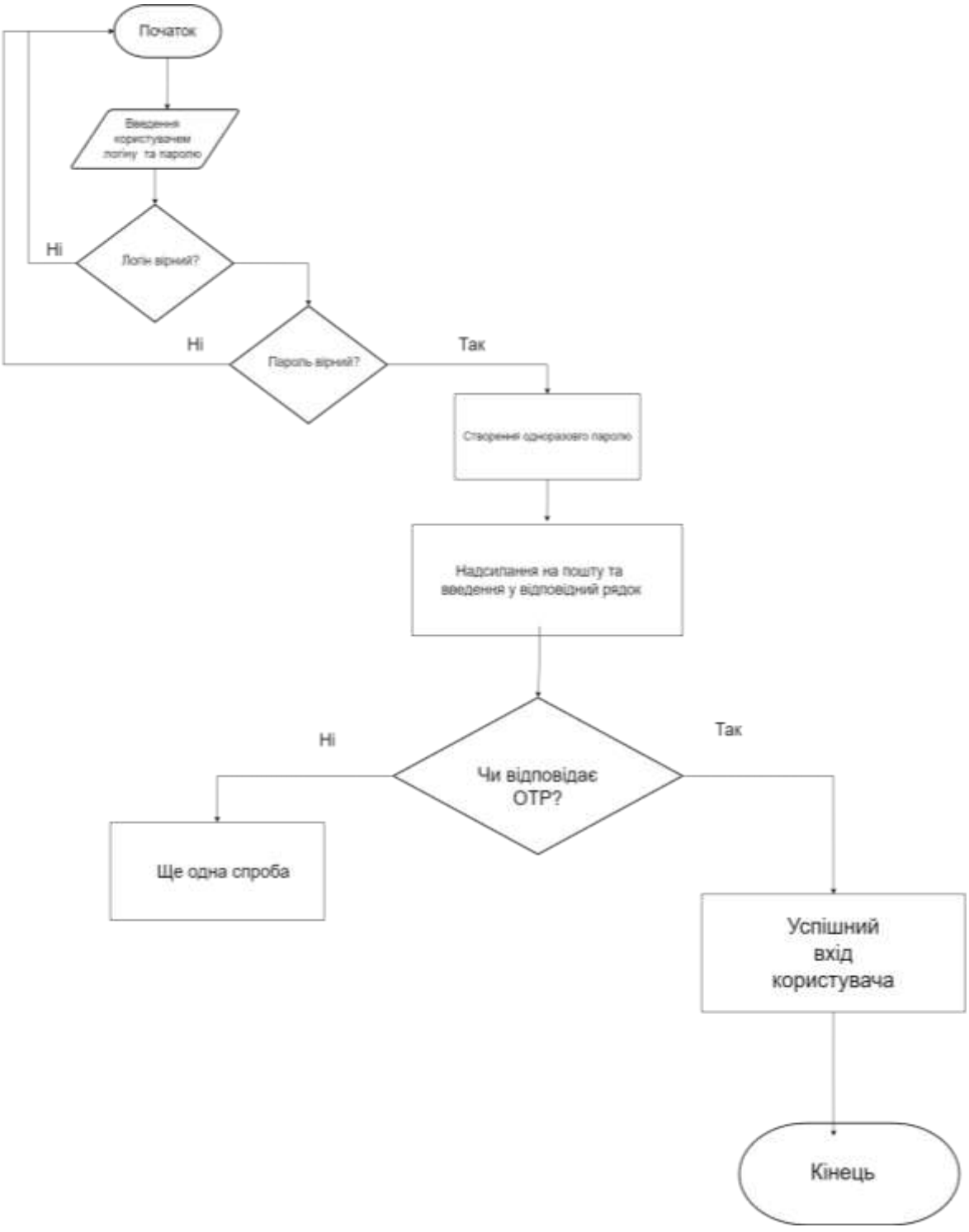


Рисунок 2.4 – Процес аутентифікації користувача у веб-ресурсі

Спочатку користувач вводить логін (email) та пароль, після чого система перевіряє правильність облікових даних. У разі успішної перевірки генерується одноразовий код, який надсилається на електронну пошту користувача. Отриманий код вводиться у відповідне поле і якщо він дійсний – користувач отримує доступ до облікового запису.

Для підвищення безпеки при використанні двофакторної аутентифікації реалізовано низку захисних механізмів. Зокрема, обмежено частоту запитів на отримання одноразового коду, а сам код має суворо обмежений термін дії – після закінчення цього часу він автоматично стає недійсним. У разі введення неправильного OTP користувач бачить повідомлення про помилку та може повторити спробу.

2.4 Інтеграція SEIM-системи для моніторингу активності користувачів

З метою виявлення, реєстрації та аналізу інцидентів безпеки у веб-ресурсі, доцільним є впровадження системи моніторингу подій безпеки – SIEM. У даному проєкті обрано рішення з відкритим кодом – Wazuh, яке дозволяє організувати централізовану обробку подій безпеки, журналювання дій користувачів та автоматичне оповіщення про потенційні загрози.

Система Wazuh вирізняється серед багатьох SIEM-рішень завдяки своїй відкритій архітектурі та повністю безкоштовному ліцензуванню. Це робить її особливо привабливою для організацій, які прагнуть забезпечити високий рівень безпеки без значних фінансових витрат. Завдяки відкритому коду Wazuh прозора у використанні, а її кодова база може бути перевірена, змінена та адаптована відповідно до специфіки проєкту.

Wazuh має гнучку архітектуру, що дозволяє легко інтегрувати її в середовище з різними операційними системами, веб-серверами і базами даних. Така універсальність дозволяє використовувати систему у широкому спектрі проєктів від невеликих веб-ресурсів до складних корпоративних інфраструктур.

Суттєвою перевагою Wazuh є підтримка агентської моделі: на кожен вузол встановлюється агент, який збирає журнали, події та метаінформацію з системи, що дозволяє детально контролювати та аналізувати активність користувачів, спроби вторгнень і зміни конфігурацій. При цьому агенти працюють ефективно та не перевантажують систему.

Користувачі Wazuh отримують змогу кастомізувати правила виявлення загроз, створювати власні політики безпеки, фільтри, шаблони інцидентів та налаштовувати автоматичне реагування на загрози.

Wazuh також відповідає вимогам провідних стандартів інформаційної безпеки, таких як GDPR**,** ISO/IEC та інших, що підтверджує відповідність як технічним, так і нормативним вимогам. Архітектура базується на агентах, що встановлюються на кінцеві точки, які передають дані на центральний сервер для аналізу.

Центральний сервер Wazuh обробляє отримані дані, аналізує їх за допомогою правил та декодерів, і передає результати до індексатора для зберігання та подальшого аналізу. Індексатор, побудований на базі Elasticsearch, забезпечує ефективне зберігання та пошук даних. Для візуалізації та управління використовується веб-інтерфейс на базі Kibana, який надає зручні панелі для перегляду журналів, сповіщень та стану системи.

Wazuh також підтримує безагентні пристрої, такі як брандмауери та маршрутизатори, які можуть надсилати журнали через Syslog або інші протоколи. Крім того, Wazuh дозволяє створювати власні правила, політики безпеки та сценарії реагування, що робить її гнучкою та адаптованою до конкретних потреб організації.

У межах реалізації буде інтегровано систему моніторингу інцидентів Wazuh. Під час її встановлення та діяльності виконуються дії, які описані нижче

Крок 1. Встановлення Wazuh Agent, який виконує функції збору логів та моніторингу системної активності. Налаштування на взаємодію з Wazuh-сервером, що розгорнутий локально. Для збору подій налаштовано моніторинг системних журналів Windows.

Крок 2. Виконання моніторингу подій безпеки за допомогою автоматичного відстеження. Wazuh забезпечує аналіз:

– кількість успішних та невдалих спроб автентифікації до системи чи веб-додатку;

– атипову активность користувачів;

– порушення політики авторизації;
– використання двофакторної автентифікації (2FA).

Крок 3. Застосовується набір правил для автоматичного виявлення інцидентів безпеки. Система класифікує події за рівнем критичності, що дозволяє швидко оцінити загрози. Виявлені інциденти можуть бути автоматично помічені як потенційно шкідливі або потребувати ручного підтвердження.

Крок 4. Для інформування адміністратора передбачено надсилання сповіщень через електронну пошту або інші канали. У перспективі можлива інтеграція з автоматизованими системами реагування, які дозволять реалізувати сценарії типу: блокування IP-адреси, завершення активної сесії тощо.

При інтеграції даної системи забезпечується моніторинг подій безпеки, що дозволяє оперативно виявляти та реагувати на інциденти безпеки, знизуючи ризик на несанкціонований доступ та загальну безпеку веб-ресурсу.

2.5 Висновки до розділу 2

У цьому розділі було розглянуто можливість проєктування захищеного веб-ресурсу для обміну конфіденційними даними, що базується на сучасних підходах до інформаційної безпеки. Здійснено розробку користувацького інтерфейсу з акцентом на зручність та інтуїтивність взаємодії. Ознайомлено з впровадженням механізму наскрізного шифрування із використанням гібридного криптографічного підходу, що гарантує конфіденційність даних навіть у разі компрометації серверної частини.

Для підвищення безпеки доступу досліджено принцип реалізації двофакторної аутентифікації, що зменшує ризик несанкціонованого входу. Також проаналізовано можливості впровадження SIEM-системи Wazuh для моніторингу активності користувачів та виявлення потенційних загроз у реальному часі в межах функціонування веб-ресурсу. Сукупність розглянутих компонентів забезпечує високий рівень захисту даних, надійність функціонування веб-ресурсу та можливість оперативного реагування на інциденти інформаційної безпеки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-РЕСУРСУ

У цьому розділі розглянуто процес програмної реалізації веб-ресурсу, розробленого в межах даної роботи. Детально описано основні функціональні можливості системи, логіку її роботи, вибір інструментів та технологій, а також принципи взаємодії між основними компонентами. Крім того, наведено приклади реалізації ключових елементів інтерфейсу та серверної частини, що забезпечують коректне функціонування ресурсу та зручність для користувача.

3.1 Обґрунтування вибору мов програмування, фреймворків, середовища розробки тощо.

Для початку реалізації безпечного веб-ресурсу важливим етапом є вибір мов програмування, фреймворків та технологій, які забезпечать надійність, зручність у використанні та можливість майбутньої масштабованості.

HTML – це мова розмітки, що складається з низки елементів , що використовуються для обгортання (або огортання) текстового вмісту, щоб визначити його структуру та змусити його поводитися певним чином [25].

CSS (англ. Cascading Style Sheets) – спеціальна мова, яка використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних [26].

JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки [27].

React – це декларативна, ефективна і гнучка JavaScript-бібліотека, призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду – компонентів [28].

Node.js – це безплатне, кросплатформне середовище виконання JavaScript із відкритим кодом, яке дозволяє розробникам створювати сервери, вебзастосунки, інструменти командного рядка та скрипти [29].

Vite — це сучасний, блискавично швидкий інструмент для створення скелетів і групування проєктів, який швидко стає популярним завдяки майже миттєвій компіляції коду та швидкій гарячій заміні модулів. У даній статті ми з вами розберемось, що ж дозволяє йому бути таким продуктивним [30].

Обрана сукупність технологій забезпечує як фронтенд, так і бекенд частини веб-ресурсу, вони враховують сучасні вимоги до продуктивності, безпечності та підтримці коду. на стороні користувача HTML, CSS, JS у поєднані з бібліотекою React дозволяють створити динамічний, адаптивний та зручний інтерфейс. На серверній частині використання Node.js у поєднані з різними бібліотеками та Vite дозволяє швидке розгортання, обробку та інтеграцію різних інструментів.

Таким чином, архітектура проєкту матиме відповідність принципу поділу обов’язків клієнт-сервер, що дозволить ефективно розмежувати обробку даних, їх візуалізацію та загальну реалізацію ресурсу.

3.2 Реалізація клієнтської частини

Клієнтська частина веб-ресурсу реалізована з використанням бібліотеки React, що дозволяє створювати інтерактивний та динамічний інтерфейс користувача. Компонентна структура React забезпечує зручність у розробці, повторне використання коду та спрощене тестування окремих частин інтерфейсу.

Основними компонентами інтерфейсу є:

– сторінка реєстрації та входу користувача, яка реазілує логіку введення облікових даних, перевірку правильності введених значеннь та ініціалізації двофакторної аутентифікації;

– форма введення одноразового паролю для 2FA, яка з’являється після успішного введення логіна та пароля;

– головна сторінка на якій міститься список чатів користувача, яка при натиснення на чат веде до сторінки обміну повідомленнями;

– сторінка обміну повідомленнями, що дозволяє користувачу надсилати повідомлення іншим користувачам.

Розглянемо детальніше деякі частини створення клієнтської сторони, а саме: форми реєстрації, логіну, сторінки обміну повідомленнями.

Форма реєстрації містить три поля: ім’я, електрона пошта та пароль. При заповненні яких є валідація, яка забезпечує введення даних відповідно до правил веб-ресурсу. Це дозволяє запобігти надсиланню некоректної інформації на сервер. Форма має наступний вигляд, як на рисунку 3.1.

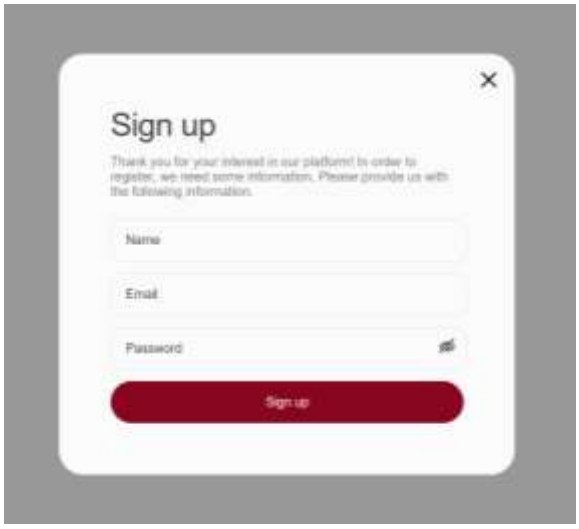


Рисунок 3.1 – Форма реєстрації користувача

Валідація введених даних реалізована за допомогою бібліотеки Yup, яка забезпечує перевірку на клієнтській стороні перед відправкою форми:

```
const schema = Yup.object().shape({
  name: Yup.string().required('Name is required'),
  email: Yup.string().email('Invalid email').required('Email is required'),
  password: Yup.string()
    .min(6, 'Min 6 characters')
    .required('Password is required'),});
```

Це гарантує, що користувач не зможе залишити поля порожніми, а також захищає від введення некоректного email чи занадто короткого пароля. Також тут можна додати ще частини які будуть більш ретельніше аналізувати введені дані та повідомляти користувача, наприклад для паролю можна добавити

обов’язковість комбінації великих та малих букв, використання спеціальних символів тощо.

Для управління станом форми використовується хук useForm з бібліотеки react-hook-form.

```
const {
  register,
  handleSubmit,
  reset,
  formState: { errors },
} = useForm({
  mode: 'onTouched',
  defaultValues: {
    name: '',
    email: '',
    password: '',
  },
  resolver: yupResolver(schema),});
```

У конфігурації хука вказано mode: 'onTouched', що означає відображення помилок лише після того, як користувач взаємодіяв із полем, але ще не натиснув кнопку відправки, що покращує користувацький досвід. Опція defaultValues задає початкові значення для всіх полів (у цьому випадку — порожні), що дозволяє ініціалізувати або скидати форму за допомогою функції reset. Сам хук повертає низку корисних функцій і об’єктів: register використовується для підключення інпутів до системи форм, handleSubmit обробляє подію відправки з автоматичною перевіркою даних, reset скидає значення до початкових, а formState.errors містить усі помилки валідації. Завдяки цьому кожне поле автоматично перевіряється згідно зі схемою, а помилки відображаються у зручний для користувача спосіб без зайвого коду.

Після підтвердження форми, дані надсилаються через Redux-операцію singUp, яка виконує запит до серверу на створення нового користувача. У разі успішної реєстрації користувача автоматично перенаправляє на сторінку логіну.

Опис форми логіну є логічним продовженням функціоналу реєстрації, проте має свої особливості. Форма логіну призначена для автентифікації користувача в системі (рис. 3.2). Вона містить два основні поля: електронна

пошта та пароль, а також додаткова обробка при виконання двофакторної аутентифікації.

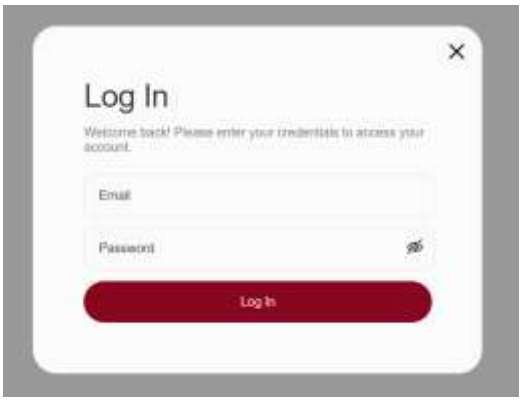


Рисунок 3.2 – Форма логіну

Основна логіка обробки логіну після натиснення кнопки Log In, активується обробник події onSubmit, який викликає Redux-операцію login. Ця операція надсилає дані користувача на сервер для перевірки автентичності.

```
const onSubmit = async data => {
  try {
    const result = await dispatch(login(data)).unwrap();
    if (result?.accessToken) {
      await setupKeys(result.accessToken);
      navigate('/mainPage');
      reset();
    } else if (result?.requires2FA && result?.email) {
      setTempEmail(result.email);
      setShow2FA(true);
    }} catch (error) {
    console.error('Login failed:', error);}};
```

Після отримання відповіді від серверу результат перевіряється. Якщо сервер повертає accessToken – це означає, що автентифікація пройшла успішно. У цьому випадку викликається додаткова функція setupKeys, яка відповідає за створення пари RSA-ключів (відкритого та закритого). Відкритий ключ передається на сервер, щоб забезпечити подальше зашифроване спілкування між клієнтом і сервером.

Після генерації ключів користувач автоматично перенаправляється на головну сторінку додатку за допомогою функції navigate('/mainPage'), а форма скидається до початкового стану.

Якщо сервер повертає requires2FA: true – це означає, що акаунт користувача захищений двофакторною автентифікацією. У такому випадку звичайного логіну недостатньо, і на наступному кроці користувач повинен ввести 6-значний код підтвердження, який було надіслано на вказану електронну пошту.

Якщо ввімкнено 2FA, користувачу відображається нова форма, в якій потрібно ввести код підтвердження з email (рис. 3.3). Цей код перевіряється через Redux-операцію verify2FA, яка надсилає код разом з email на сервер для верифікації.

```
const handle2FASubmit = async e => {
  e.preventDefault();
  try {
    const result = await dispatch(
      verify2FA({ code, email: tempEmail })).unwrap();
    ...
  } catch (error) {
    console.error('2FA verification failed:', error);}};
```

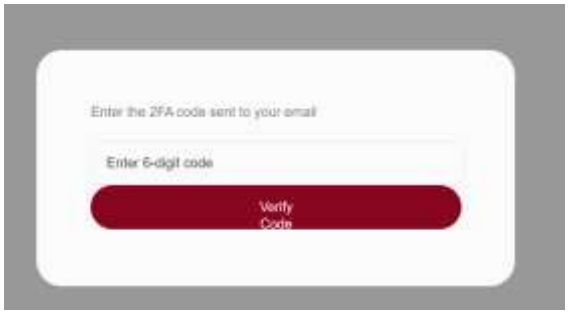


Рисунок 3.3 – Форма введення двофакторної аутентифікації

Подібно до звичайної авторизації, якщо перевірка проходить успішно й сервер повертає accessToken. У разі помилки або невірного коду, у консоль виводиться повідомлення про невдалу верифікацію. Також можливо вивести відповідне повідомлення на інтерфейсі для зворотного зв’язку користувачу.

Сторінка чату (рис. 3.4) є важливою частиною функціоналу застосунку, що забезпечує захищене спілкування між користувачами. Вона реалізує відображення повідомлень, динамічне оновлення інтерфейсу, шифрування та дешифрування повідомлень, а також відправлення нових повідомлень через сокет-з’єднання. У компоненті використовуються сучасні React-хуки,

асинхронна логіка, Redux для отримання даних користувачів і кастомний хук для роботи з сокетом. Далі розглянемо основні частини реалізації цієї сторінки.

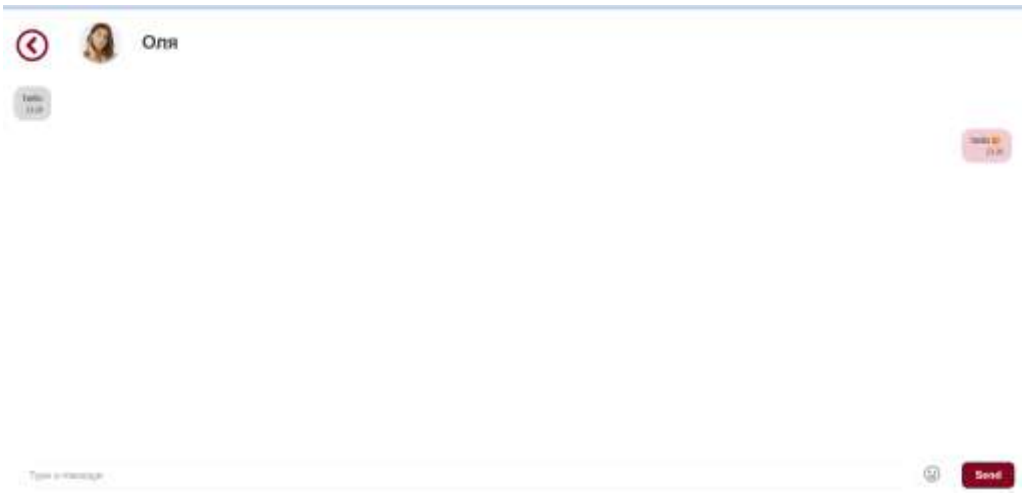


Рисунок 3.4 – основна сторінка обіну повідмленнями

Ініціалізація та отримання даних виконується через передачу на сторінку об’єкт currentUser та chat, які використовуються для визначення активного чату та користувачів, що беруть в ньому участь. Далі за допомогою Redux-операції fetchUserById отримуються дані співрозмовника.

```
const userId = chat?.members?.find(id => id !== currentUser);
const actionResult = await dispatch(fetchUserById(userId));
setUserData(actionResult.payload.data);
```

Після завантаження сторінки, відбувається запит до API для отримання всіх повідомлень з цього чату. Це виконується для забезпечення перегляду попередніх повідомлень. Повідомлення зберігаються в зашифрованому вигляді, тому кожне з них потрібно розшифрувати перед відображенням. Це виконується через функцію decryptMessageForUser.

```
const decryptedMessages = await Promise.all(
  data.map(async msg => {
    const text = await decryptMessageForUser(
      msg.encryptedMessage,
      msg.encryptedAESKey,
      msg.iv,
      privateKey
    );
    return { ...msg, text };}));
setMessages(decryptedMessages);
```

Якщо будь-яке з полів шифрування відсутнє, повідомлення позначається як некоректне ([Invalid message data] або [Unable to decrypt].

За допомогою кастомного хука useSocket, повідомлення, що приходять у реальному часі також розшифровуються.

```
if (receiveMessage?.chatId === chat._id) {
  const text = await decryptMessageForUser(
    receiveMessage.encryptedMessage,
    receiveMessage.encryptedAESKey,
    receiveMessage.iv,
    privateKey
  );
  setMessages(prev => [...prev, { ...receiveMessage, text }]);
}
```

Дана частина коду відповідає за відображення нових повідомлень без оновлення сторінки.

Після введення повідомлення користувачем та натискання кнопки "Send", виконується обробка події, яка починається з запобігання стандартній поведінці форми, щоб уникнути перезавантаження сторінки. Далі визначається ID отримувача, і з сервера запитується його публічний ключ для подальшого шифрування повідомлення. Текст повідомлення шифрується з використанням алгоритму AES, а отриманий AES-ключ додатково шифрується публічним ключем співрозмовника. Далі створюється об’єкт повідомлення.

```
const message = {
  senderId: currentUser,
  chatId: chat._id,
  encryptedMessage,
  encryptedAESKey,
  iv,
};
```

Цей об’єкт надсилається на сервер за допомогою addMessage(message), після чого додається до локального стану компоненту, щоби миттєво відобразити його в інтерфейсі користувача. Окрім цього, повідомлення передається через сокет для негайної доставки адресату, що дозволяє забезпечити обмін повідомленнями в режимі реального часу при збереженні конфіденційності.

3.3 Реалізація серверної частини

У цьому підрозділі розглянуто ключові аспекти реалізації серверної логіки, включаючи основні маршрути, контролери та використані технології. Серверна частина застосунку реалізована за допомогою середовища виконання Node.js із використанням фреймворку Express. Вона забезпечує обробку запитів від клієнта, збереження й обробку даних у базі даних MongoDB, а також реалізує логіку авторизації користувачів, обмін повідомленнями в чаті та шифрування повідомлень. Архітектура побудована за принципом поділу на маршрути, контролери, моделі та допоміжні утиліти, що спрощує підтримку й масштабування проєкту. Сервер також інтегрований із WebSocket для забезпечення реального часу при надсиланні та отриманні повідомлень між користувачами.

Ініціалізація серверної частини починається з файлу bootstrap.js, який виконує підготовчі дії перед запуском сервера. Зокрема, тут відбувається ініціалізація підключення до бази даних MongoDB, а також створення необхідних директорій для збереження тимчасових та постійних завантажень (наприклад, зображень або файлів). Лише після успішного виконання цих дій запускається сервер.

```
const bootstrap = async () => {
  await initMongoDB();
  await createDirIfNotExists(TEMP_UPLOAD_DIR);
  await createDirIfNotExists(UPLOAD_DIR);
  startServer();
};
void bootstrap();
```

Основна конфігурація та запуск сервера реалізовані у файлі server.js. У цьому файлі створюється екземпляр сервера за допомогою фреймворку Express. Для обробки запитів використано такі проміжні обробники (middleware), як express.json() для роботи з JSON-даними, cookieParser() для обробки cookie, cors() для налаштування політики доступу з інших доменів та pino — для логування запитів. Далі підключається головний маршрутизатор router, який об’єднує всі доступні маршрути. Також налаштовано статичну видачу файлів із директорії

uploads, і підключено обробники помилок: для некоректних маршрутів (notFoundHandler) та глобального оброблення винятків (errorHandler).

```
export const startServer = () => {
  const app = express();
  app.use(express.json());
  app.use(cors(corsConfigs));
  app.use(cookieParser());
  app.use(
    pino({
      transport: {
        target: 'pino-pretty',
      },}));
  app.use(router);
  app.use('/uploads', express.static(UPLOAD_DIR));
  app.use(errorHandler);
  app.use(notFoundHandler);
  app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
  });};
```

Ці два файли відіграють ключову роль у забезпеченні стабільної та організованої роботи серверної частини застосунку.

Для організації обробки HTTP-запитів у серверній частині застосунку використовується модульна маршрутизація з Express. Уся логіка маршрутизації зібрана у відповідні роутери, кожен з яких відповідає за окрему область функціоналу: автентифікацію, роботу з користувачами, чатами та повідомленнями. Це дозволяє підтримувати структуру коду чистою, зрозумілою та масштабованою.

У файлі index.js (головному маршрутизаторі) відбувається підключення окремих маршрутів до загального об'єкта router:

```
router.use('/users', UsersRouter);
router.use('/auth', authRouter);
router.use('/chat', ChatRoute);
router.use('/message', MessageRoute);
```

Маршрути для автентифікації обробляють реєстрацію, вхід користувача, верифікацію двофакторної автентифікації (2FA), вихід із системи, оновлення сесії та зміну пароля. Для кожного маршруту передбачено валідацію вхідних

даних за допомогою middleware, а також обгортку ctrlWrapper, яка дозволяє зручно обробляти помилки в асинхронних контролерах.

```
router.post('/register',validateBody(registerUserSchema),
ctrlWrapper(registerUserController));
router.post('/login', validateBody(loginUserSchema), ctrlWrapper(loginUserController));
router.post('/verify-2fa', ctrlWrapper(verify2FAController));
```

У users.js реалізовано маршрути для взаємодії з даними користувача. Сюди входить створення, редагування, видалення користувача, а також завантаження фотографій і відкритих ключів для шифрування. Всі маршрути захищені middleware authenticate, що забезпечує доступ лише для авторизованих користувачів.

```
router.get('/', ctrlWrapper(getUsersController));
router.post('/', upload.single('photo'), validateBody(createUserSchema),
ctrlWrapper(createUserController));
router.put('/:userId', upload.single('photo'), validateBody(createUserSchema),
ctrlWrapper(upsertUserController));
```

Маршрути для чатів обробляють створення нового чату, пошук чату між двома користувачами та отримання чатів конкретного користувача. Повідомлення мають два основні запити – додавання повідомлення до чату та отримання історії повідомлень за chatId.

Завдяки такій структурі кожна частина функціоналу логічно відокремлена, що суттєво полегшує підтримку, тестування та розвиток застосунку.

Кожен роут викликає певний контролер, що відповідає за обробку вхідного запиту, виконання необхідної логіки, формування та відправку відповіді клієнту.

Контролери дозволяють розділити код за функціоналом, зробити його більш структурованим та зручним для підтримки. Кожен контролер зазвичай відповідає за певну бізнес-операцію або групу операцій.

Розглянемо для прикладу кілька контролерів, за що відповідають та що виконують.

Контролер для авторизації користувача. Він приймає логін та пароль, перевіряє їх, та, якщо користувач має ввімкнену двофакторну автентифікацію (2FA), генерує код і надсилає його на електронну пошту.

```
export const loginUserController = async (req, res) => {
  try {
    const user = await loginUser(req.body);
    if (user.isTwoFactorAuthEnabled) {
      const code = generate2FACode();
      await send2FACodeByEmail(user.email, code);
      await save2FACodeForUser(user.id, code);
      return res.status(200).json({ message: '2FA code sent', userId: user.id });
    }
    const { accessToken, refreshToken, sessionId } = await createSession(user.id);
    res.cookie('accessToken', accessToken, { httpOnly: true, secure: true });
    res.cookie('refreshToken', refreshToken, { httpOnly: true, secure: true });
    res.cookie('sessionId', sessionId, { httpOnly: true, secure: true });
    return res.status(200).json({ message: 'Login successful', userId: user.id });
  } catch (error) {
    return res.status(401).json({ message: 'Invalid credentials' });}};
```

Основні кроки які виконує даний контролер перевірку облікових даних, відправку коду для підтвердження аутентифікації, створення сесії і відправки токенів у cookie.

Контролер refreshUsersSessionController відповідає за оновлення сесії користувача у випадку, коли термін дії основного токена доступу (access token) минув, але користувач все ще має дійсний токен оновлення (refresh token). Це дозволяє забезпечити безперервний доступ без повторного введення логіну і пароля.

Спершу контролер отримує з cookie клієнта два ключові значення: refreshToken та sessionId. Якщо будь-який з цих токенів відсутній, сервер повертає помилку авторизації, оскільки без них оновити сесію неможливо. Далі відбувається валідація сесії: контролер перевіряє, чи існує сесія з таким sessionId і чи відповідає збережений refresh token тому, що надійшов. Якщо валідація не проходить, сервер повертає статус 403 – доступ заборонено.

```
export const refreshUsersSessionController = async (req, res) => {
  try {
    const { refreshToken, sessionId } = req.cookies;
    if (!refreshToken || !sessionId) {
      return res.status(401).json({ message: 'No refresh token or session ID provided' });}
    const session = await verifySession(sessionId, refreshToken);
    if (!session) {
      return res.status(403).json({ message: 'Invalid session or refresh token' });}
    const { newAccessToken, newRefreshToken } = await renewTokens(session.userId);
    res.cookie('accessToken', newAccessToken, {
```

```
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'Strict',
   });
   res.cookie('refreshToken', newRefreshToken, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'Strict',
   });
   res.cookie('sessionId', sessionId, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'Strict',
   });
   return res.status(200).json({ message: 'Session successfully refreshed' });
  } catch (error) {
   console.error('Error refreshing session:', error);
   return res.status(500).json({ message: 'Internal server error' });
  }};
```

У разі успішної перевірки, система генерує нові токени — новий access token і оновлений refresh token. Ці токени записуються назад у cookie з параметрами, що забезпечують безпеку: HTTP-only, secure, sameSite. Це захищає їх від доступу через JavaScript і атаки типу CSRF. Нарешті, контролер відправляє клієнту відповідь зі статусом 200, підтверджуючи успішне оновлення сесії.

Таким чином, refreshUsersSessionController забезпечує безпечне і зручне продовження авторизації користувача, зменшуючи ризики, пов’язані з викраденням токенів, і дозволяючи керувати активними сесіями. Це важливий елемент сучасних систем аутентифікації, що базуються на JWT і cookie.

Контролер addMessage відповідає за додавання нового повідомлення до чату. Він приймає з req.body ідентифікатори чату, відправника та текст повідомлення. На основі цих даних створюється новий об’єкт повідомлення, який зберігається в базі даних.

```
export const addMessage = async (req, res) => {
  const { chatId, senderId, text } = req.body;
  const message = new MessageModel({
    chatId,
    senderId,
    text,});
  try {
    const result = await message.save();
    res.status(200).json(result);
```

```
  } catch (error) {
    res.status(500).json(error);}};
```

Якщо збереження проходить успішно, контролер повертає клієнту статус 200 та об’єкт нового повідомлення. У разі помилки – статус 500 і інформацію про помилку. Цей контролер забезпечує збереження повідомлень і їх доступність для подальшої роботи в чаті.

Важливу роль відіграє налаштування підключення до бази даних MongoDB та створення відповідних моделей даних для користувачів, сесій, чатів і повідомлень. Ці моделі визначають структуру збережених документів і забезпечують коректну роботу з інформацією в базі даних. Далі наведено детальний опис ключових частин.

Функція initMongoDB відповідає за ініціалізацію підключення до бази даних. Вона зчитує параметри підключення (користувача, пароль, URL, назву бази) із захищених змінних середовища за допомогою функції getEnvVar. Це гарантує, що конфіденційні дані не будуть жорстко закодовані у вихідному коді.

```
export const initMongoDB = async () => {
  try {
    const user = getEnvVar('MONGODB_USER');
    const pwd = getEnvVar('MONGODB_PASSWORD');
    const url = getEnvVar('MONGODB_URL');
    const db = getEnvVar('MONGODB_DB');
    await mongoose.connect(
      `mongodb+srv://${user}:${pwd}@${url}/${db}?retryWrites=true&w=majority`
    );
    console.log('Mongo connection successfully established!');
  } catch (e) {
    console.log('Error while setting up mongo connection', e);
    throw e;}};
```

Потім виконується асинхронне підключення до MongoDB за допомогою mongoose.connect, де використовується спеціальний URI, що передбачає підключення до кластерної бази даних MongoDB Atlas. Якщо підключення відбувається успішно – в консолі з’являється повідомлення про успіх, у випадку помилки – помилка логуються, а виклик викидає виключення для обробки зверху.

Схема usersSchema описує структуру користувача в базі, включаючи обов’язкові поля — ім’я, email та пароль — та додаткові атрибути, наприклад, фото, дату народження чи двофакторну аутентифікацію. Метод toJSON прибирає поле пароля при серіалізації об’єкта користувача, щоб не віддавати його клієнту.

Схема сесій зберігає інформацію про токени авторизації користувача, а також терміни їх дії. Це необхідно для безпечного контролю доступу та оновлення сесій без повторного введення логіна і пароля.

Схеми чатів і повідомлень забезпечують роботу внутрішньої комунікації. Чат містить список учасників, а повідомлення зберігають інформацію про відправника, чат і текст повідомлення. Обидві схеми підтримують автоматичне додавання часових позначок.

Для реалізації живої комунікації між користувачами широко використовується WebSocket, яка дозволяє встановити постійне двонаправлене з’єднання між клієнтом і сервером. Даний код ілюструє базову серверну логіку на основі бібліотеки Socket.IO, яка значно спрощує роботу з WebSocket, забезпечуючи надійний та ефективний механізм обробки подій у реальному часі.

Спочатку виконується запуск серверної частини з дозволом підключення лише з певного клієнтського домену. Далі описано основні функції, які виконуються.

1. Підключення користувачів та відстеження активних сесій.

Після встановлення WebSocket-з’єднання кожен користувач надсилає подію 'new-user-add' зі своїм унікальним ідентифікатором. Сервер перевіряє, чи такого користувача ще немає у списку активних користувачів activeUsers. Якщо ні – додає його з відповідним socketId, який ідентифікує конкретне WebSocket-з’єднання.

```
socket.on('new-user-add', newUserId => {
  if (!activeUsers.some(user => user.userId === newUserId)) {
    activeUsers.push({
      userId: newUserId,
      socketId: socket.id,
    });}
  io.emit('get-users', activeUsers);});
```

Це дозволяє серверу знати, хто зараз онлайн, і підтримувати актуальний список підключених користувачів.

Після додавання користувача сервер транслює оновлений список активних користувачів усім підключеним клієнтам за допомогою події 'get-users'.

2. Відключення користувачів.

Коли користувач відключається (подія 'disconnect'), його інформація видаляється зі списку активних користувачів, щоб не зберігати зайві записи. Потім сервер оновлює всі клієнти, надсилаючи новий список користувачів без цього відключеного.

```
socket.on('disconnect', () => {
  activeUsers = activeUsers.filter(user => user.socketId !== socket.id);
  io.emit('get-users', activeUsers);
});
```

3. Обмін повідомленнями між користувачами.

Коли користувач надсилає повідомлення (подія 'send-message'), сервер отримує об’єкт data, який повинен містити, зокрема, receiverId – ідентифікатор користувача, що має отримати повідомлення. Сервер шукає у списку активних користувачів отримувача за receiverId. Якщо одержувач онлайн, сервер надсилає йому це повідомлення через його унікальний socketId подією 'receive-message'.

```
socket.on('send-message', data => {
  const { receiverId } = data;
  const user = activeUsers.find(user => user.userId === receiverId);
  if (user) {
    io.to(user.socketId).emit('receive-message', data);
  }});
```

Цей код створює простий, але ефективний WebSocket-сервер для реалізації функціоналу чату в реальному часі. Він відстежує активних користувачів, оновлює інформацію про підключення/відключення, а також забезпечує доставку повідомлень між користувачами безпосередньо через сокети. Завдяки використанню Socket.IO, сервер підтримує масштабованість та надійність передачі даних.

У цьому підрозділі було описано серверну частину проєкту. Вона побудована так, щоб було зручно додавати нові функції і підтримувати код. Використання Express.js допомагає легко обробляти запити від користувачів. Розподіл коду на різні частини робить його більш зрозумілим і організованим. Також враховано безпеку та обробку помилок, що робить сервер стабільним і надійним. Такий підхід підходить для створення сучасних веб-додатків.

3.4 Застосування наскрізного шифрування

Наскрізне шифрування є одним із ключових механізмів забезпечення конфіденційності даних. Його мета є реалізувати можливість обміном повідомленнями між відправником і отримувачем, залишаючи повідомлення зашифровані протягом усього шляху передачі.

Першим кроком у реалізації наскрізного шифрування є створення криптографічної пари ключів: відкритого та закритого. Відкритий ключ буде використовуватися для шифрування даних, а закритий – виключно для їх дешифрування. Цей процес можна реалізувати за допомогою Web Crypto API, який доступний у сучасних браузерах. У наведеному нижче коді представлена функція generateRSAKeyPair, яка асинхронно генерує RSA-ключі з використанням алгоритму RSA-OAEP.

```
export const generateRSAKeyPair = async () => {
  const keyPair = await window.crypto.subtle.generateKey(
    {
      name: 'RSA-OAEP',
      modulusLength: 2048,
      publicExponent: new Uint8Array([1, 0, 1]),
      hash: 'SHA-256',
    },
    true,
    ['encrypt', 'decrypt']);
```

У цьому фрагменті відбувається виклик функції crypto.subtle.generateKey, яка створює пару ключів. Алгоритм RSA-OAEP (Optimal Asymmetric Encryption Padding) забезпечує безпечне шифрування. modulusLength встановлений на 2048 біт, що є сучасним стандартом безпеки. Поле publicExponent містить значення

0x010001, яке широко використовується як загальноприйняте значення для відкритого експонента. Алгоритм хешування SHA-256 визначає криптографічну стійкість операцій. Параметр true вказує, що ключі можуть бути експортовані, а масив ['encrypt', 'decrypt'] визначає дозволені операції для цих ключів.

Далі, відкритий і закритий ключі експортуються в форматах `spki` і `pkcs8` відповідно.

```
const publicKey = await window.crypto.subtle.exportKey(
  'spki',
  keyPair.publicKey);
const privateKey = await window.crypto.subtle.exportKey(
  'pkcs8',
  keyPair.privateKey);
```

На завершення, обидва ключі конвертуються з масиву байтів у Base64-рядки для зручного збереження або передачі, наприклад через мережу або API.

Далі у процесі реалізації наскрізного шифрування поєднується симетричні і асиметричні алгоритми. Нижче розглянуто фрагменти коду, які демонструють, як реалізувати такий підхід за допомогою Web Crypto API в браузерному середовищі.

```
export function base64ToArrayBuffer(base64) {
  if (!base64 || typeof base64 !== 'string') {
    throw new Error('Invalid Base64 input');
  }
  try {
    const binary = atob(base64);
    const len = binary.length;
    const buffer = new Uint8Array(len);
    for (let i = 0; i < len; i++) {
      buffer[i] = binary.charCodeAt(i);
    }
    return buffer;
  } catch (e) {
    console.error('base64ToArrayBuffer failed. Input:', base64);
    throw e;}}
```

Ця функція приймає Base64-строку, декодує її у двійкові дані, і повертає Uint8Array – буфер байтів. Вона перевіряє, чи вхідні дані коректні, а потім через atob() декодує Base64. Кожен символ перетворюється у байт за допомогою charCodeAt(). Якщо щось піде не так – викидається помилка з виводом в консоль.

```
export async function importRSAPublicKey(base64Key) {
  const binaryDer = base64ToArrayBuffer(base64Key);
  return crypto.subtle.importKey(
    'spki',
    binaryDer,
    {
      name: 'RSA-OAEP',
      hash: 'SHA-256',
    },
    true,
    ['encrypt']
  );}
```

Ця функція імпортує відкритий ключ RSA, який поданий у форматі Base64. Спочатку він декодується через base64ToArrayBuffer(), потім використовується crypto.subtle.importKey для імпорту ключа у форматі spki (SubjectPublicKeyInfo). Ключ призначений для шифрування і використовується з алгоритмом RSA-OAEP та хеш-функцією SHA-256.

```
export async function encryptMessageForUser(message, recipientPublicKeyBase64) {
  const encoder = new TextEncoder();
  const aesKey = await crypto.subtle.generateKey(
    { name: 'AES-GCM', length: 256 },
    true,
    ['encrypt', 'decrypt']
  );
  const iv = crypto.getRandomValues(new Uint8Array(12));
  const encrypted = await crypto.subtle.encrypt(
    { name: 'AES-GCM', iv },
    aesKey,
    encoder.encode(message)
  );
  const rawAesKey = await crypto.subtle.exportKey('raw', aesKey);
  const recipientPublicKey = await importRSAPublicKey(recipientPublicKeyBase64);
  const encryptedAESKey = await crypto.subtle.encrypt(
    { name: 'RSA-OAEP' },
    recipientPublicKey,
    rawAesKey
  );
  return {
    encryptedMessage: btoa(String.fromCharCode(...new Uint8Array(encrypted))),
    encryptedAESKey: btoa(
      String.fromCharCode(...new Uint8Array(encryptedAESKey))
    ),
    iv: btoa(String.fromCharCode(...iv)),
  };}
```

Ця функція виконує гібридне шифрування повідомлення виконуючи послідовність дій, а саме: створення симетричного AES-ключ, генерування випадкового ініціалізаційного вектору, кодування повідомлення у байти і шифрування через AES, експорт AES-ключа для шифрування, імпорт відкритого ключа користувача для шифрування AES-ключа через RSA. Об’єкт, що повертається у результаті виконання містить зашифроване повідомлення (encryptedMessage), AES-ключ, зашифрований RSA (encryptedAESKey) і ініціалізаційний вектор у Base64.

Також є ще функція getStoredPrivateKey та decryptMessageForUser. Перша функція зчитує закритий ключ RSA із localStorage. Після декодування через base64ToArrayBuffer імпортується через crypto.subtle.importKey як pkcs8 (Private Key Info формат). Цей ключ використовуватиметься для розшифрування повідомлень.

Друга функція використовується для розшифрування повідомлення. Спочатку зашифрований AES-ключ спочатку розшифровується за допомогою закритого ключа RSA і алгоритму RSA-OAEP. Потім цей AES-ключ імпортується як ключ для AES-GCM. Саме повідомлення розшифровується з використанням AES та наданого iv. Результат виконання з буфера в рядок і повертається.

У даній реалізації кожен користувач зберігає свій відкритий ключ на сервері. Це дозволяє іншим користувачам отримувати його для шифрування повідомлень, які може розшифрувати лише власник приватного ключа. Нижче наведено серверну частину для збереження (POST) і отримання (GET) відкритого ключа користувача.

Функція uploadPublicKey обробляє HTTP POST-запит, який надсилає клієнт, щоб зберегти свій відкритий ключ на сервері.

```
export const uploadPublicKey = async (req, res) => {
  try {
    const { publicKey } = req.body;
    const userId = req.user.id;
    if (!publicKey) {
      return res.status(400).json({ message: 'Public key is required' });
```

```
    }
    await UsersCollection.findByIdAndUpdate(userId, { publicKey });
    return res.status(200).json({ message: 'Public key saved successfully' });
  } catch (error) {
    console.error('Error saving public key:', error);
    return res.status(500).json({ message: 'Server error' });
  }};
```

У тілі запиту очікується об'єкт із відкритим ключем. Ідентифікатор користувача витягується з об'єкта req.user, що зазвичай формується після проходження користувачем аутентифікації. Якщо ключ не передано – сервер повертає помилку 400. У разі успіху ключ зберігається у базі даних MongoDB через метод findByIdAndUpdate, і користувач отримує повідомлення про успішне збереження.

Щоб один користувач міг надіслати зашифроване повідомлення іншому, йому потрібно отримати відкритий ключ одержувача. Для цього призначена функція getUserPublicKey, яка обробляє HTTP GET-запит.

```
export const getUserPublicKey = async (req, res) => {
  try {
    const user = await UsersCollection.findById(req.params.id).select(
      'publicKey'
    );
    if (!user || !user.publicKey) {
      return res.status(404).json({ message: 'Public key not found' });
    }
    return res.status(200).json({ publicKey: user.publicKey });
  } catch (error) {
    console.error('Error fetching public key:', error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

Ця функція здійснює пошук користувача за ID, отриманим із параметра маршруту. Метод select('publicKey') обмежує результат лише полем відкритого ключа. Якщо такий користувач не знайдений або не має збереженого ключа, повертається помилка 404. У протилежному випадку відкритий ключ повертається у форматі JSON.

Для інтеграції описаних функцій у сервер додано відповідні маршрути Express. Маршрут POST /public-key дозволяє автентифікованому користувачу зберігати свій відкритий ключ. Маршрут GET /:id/public-key дозволяє будь-

якому користувачу отримати відкритий ключ іншого користувача за його ID. Це створює основу для реалізації наскрізного шифрування в межах клієнт-серверної архітектури, де всі повідомлення шифруються локально на клієнті за допомогою відкритого ключа адресата.

Таким чином, реалізація наскрізного шифрування забезпечує високий рівень конфіденційності обміну даними між користувачами. Поєднання симетричного та асиметричного шифрування дозволяє ефективно захищати як самі повідомлення, так і ключі, якими вони шифруються. Збереження відкритих ключів на сервері і обмежений доступ до приватних ключів гарантують, що лише призначений одержувач може розшифрувати отриману інформацію. Такий підхід є надійним фундаментом для створення сучасних безпечних комунікаційних систем.

3.5 Реалізація двофакторної аутентифікації

Для підвищення рівня безпеки користувацьких облікових записів у системі реалізовано двофакторну аутентифікацію (2FA). Її мета – захистити акаунти навіть у випадку, якщо зловмисник дізнається пароль.

На клієнті реалізовано логіку, яка перевіряє, чи після введення логіну та паролю користувачу потрібно пройти додаткову перевірку. Це реалізовано через Redux-операцію login, яка обробляє відповідь сервера. Якщо у відповіді відсутній токен, але зазначено, що потрібна 2FA, то відображається форма введення коду:

```
if (result?.requires2FA && result?.email) {
  setTempEmail(result.email);
  setShow2FA(true);
}
```

Після введення коду формується запит на перевірку через Redux-операцію verify2FA, яка надсилає код та email.

```
dispatch(verify2FA({ code, email: tempEmail }))
```

У випадку успішної перевірки код очищується, а користувача перенаправляють на головну сторінку. Також зберігаються криптографічні ключі в локальному сховищі, якщо їх ще не існує.

На сервері реалізовано контролер verify2FAController, який:

– Шукає користувача за email;

– Перевіряє правильність і актуальність одноразового коду (twoFactorCode) та терміну його дії (twoFactorExpires);

– Якщо код вірний, очищує ці поля в базі даних і встановлює прапорець isTwoFactorVerified: true;

– Створює сесію та видає токени (access та refresh);

– Повертає відповідь з токеном і даними користувача.

```
if (
  !user.twoFactorCode ||
  !user.twoFactorExpires ||
  user.twoFactorCode !== code ||
  user.twoFactorExpires < new Date()
) {
  return res.status(400).json({ message: 'Invalid or expired verification code' });
}
```

Стан користувача при логіні враховує, чи користувач уже пройшов двофакторну аутентифікацію. Якщо так – одразу надається токен доступу, і користувач вважається авторизованим. Інакше система очікує завершення другого етапу перевірки.

Для зручності перегляду відправленого коду піж час реалізації всі налаштування додатково виводимо в консоль (рис. 3.5).

Рисунок 3.5 – Ввиведені дані для повідомлення в консолі

Таким чином, реалізована система двофакторної аутентифікації підвищує рівень захищеності користувачів шляхом додаткової перевірки

особистості за допомогою одноразового коду. Такий підхід забезпечує ефективний захист облікових записів навіть у випадку компрометації пароля. Інтеграція 2FA була виконана з урахуванням зручності користувача, а також збереженням безпечного зберігання ключів, що робить систему не лише надійною, а й сучасною з точки зору кібербезпеки.

3.6 Інтеграція SIEM-системи

Використання SIEM-систем є важливим етапом, який передбачає в роботі можливість об’єднання різнорідних джерел, таких як журнали подій, мережевий трафік, системи аутентифікації та інші засоби моніторингу, з метою збору, аналізу та кореляції інформації. У цьому підрозділі розглядається частина дій, які виконуються для налаштування.

Для виконання поставленого завдання обрано SIEM-систему Wazuh, яка забезпечує ефективний моніторинг та захист. Має досить велику кількість способів реалізації детальніше з якими можна ознайомитися у офіційній документації.

Першим етапом є встановлення центральних компонентів. У даному проєкті було використано альтернативний спосіб встановлення, а саме через віртуальну машину. Wazuh надає попередньо створений образ віртуальної машини (OVA), який можна безпосередньо імпортувати за допомогою VirtualBox або інших систем віртуалізації, сумісних з OVA.

Система відкривається у вигляді вікна консолі (рис. 3.6), що виконує роль сервера. Тут можна виконувати дії які пов’язані з налаштуванням. У результаті запуску ми отримуємо посилання https:/<ip-adress>, перейшовши за якої отримуємо розгорнуте систему. Також для того, щоб виконувалось збір необхідно встановити агента безпосередньо на кінцеву точку з якою виконуються дії.

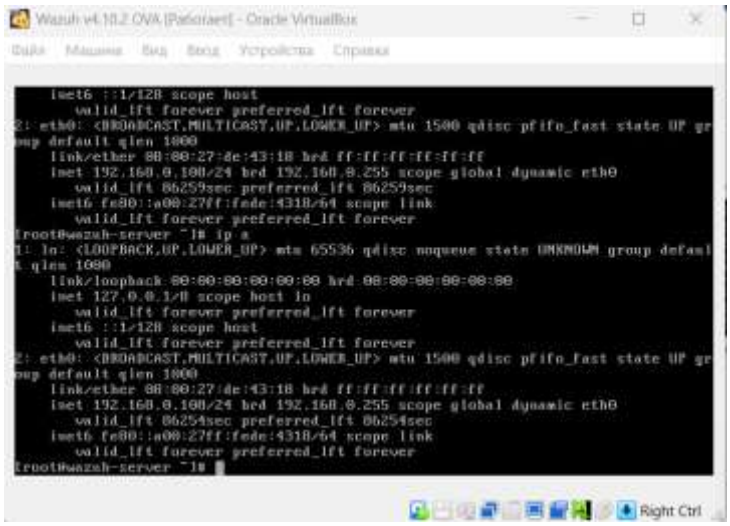


Рисунок 3.6 – Wazuh на віртуальній машині

Для отримання даних з проєкту створено функцію logActivity, що є універсальним механізмом для ведення журналу подій у системі. Її основна мета – зберігати інформацію про різні дії користувачів або системні події, фіксуючи час, користувача, IP-адресу, а також додаткові контекстні дані.

```
export const logActivity = async ({ req, userId, action, metadata = {} }) => {
 try {
  const logEntry = {
   userId,
   action,
   ip: req.ip,
   userAgent: req.headers['user-agent'],
   metadata,
   timestamp: new Date().toISOString(),
  };
  await LogsCollection.create(logEntry);
  logger.info(logEntry);
 } catch (error) {
  console.error('Logging error:', error);}};
```

Функція оголошена як асинхронна і приймає об'єкт із кількома параметрами: об’єкт HTTP-запиту req, ідентифікатор користувача userId, тип дії action (наприклад, 'LOGIN_ATTEMPT' чи 'LOGOUT') та необов’язковий об’єкт metadata для додаткової інформації.

Всередині функції створюється об'єкт logEntry, який агрегує всі необхідні для запису дані: userId, текст дії action, IP-адресу користувача, отриману із

запиту, а також інформацію про браузер чи клієнт, що відправив запит, через HTTP-заголовок user-agent.

Після підготовки цього об’єкта відбувається асинхронне збереження запису у базу даних через виклик LogsCollection.create(logEntry). Одночасно цей же об’єкт передається у logger.info(), що забезпечує збереження логу у спеціальний файл. Це необхідно для подальшого аналізу подій системою моніторингу безпеки.

Для забезпечення надійного та зручного ведення журналу подій у додатку був створений централізований логер за допомогою бібліотеки Winston. Його основною метою є збір і збереження важливої інформації про дії користувачів, системні події та помилки у структурованому форматі.

Логер допомагає фіксувати важливі повідомлення, наприклад, дії користувачів, помилки або інші інформаційні події, що необхідно зберігати для аналізу або аудиту.

Спочатку визначається шлях до файлу журналу app.log (рис. 3.7), у який будуть записуватись логи. Шлях формується відносно кореневого каталогу проекту, щоб логи завжди зберігались у папці logs всередині проекту.

Рисунок 3.7 – Приклад отриманих даних

```
const logFilePath = path.join(process.cwd(), 'logs', 'app.log');
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [new winston.transports.File({ filename: logFilePath })],
});
```

Далі створюється сам логер, де властивість level: 'info' вказує, що у файл будуть записуватись всі повідомлення рівня info і вищого, format: winston.format.json() означає, що всі записи логів будуть зберігатись у форматі JSON, transports визначає, куди логер буде записувати повідомлення.

Для того, щоб дані відправлялись для обробки на сервер Wazuh необхідно в нести зміни до конфігураційних файлів.

```
<localfile>
        <log_format>json</log_format>
        <location>C:\Users\User\Desktop\курси\Bachelor-s-app\backend\logs\app.log</location>
</localfile>
```

Цей блок конфігурації описує локальний файл журналу, куди система записує логи. Визначає точний шлях до файлу, у якому зберігаються записи та дозволяє легко знаходити та обробляти журнали подій додатку.

Одним з фрагментів конфігурації є правило для Wazuh, що описує створену групу правил, яка використовується для аналізу логів. Завдяки цим правилам, Wazuh може розпізнавати важливі події у вашому додатку такі як спроби входу, проходження або невдача 2FA, надсилання кодів тощо.

Кожне правило в межах цієї групи шукає конкретне значення в полі message.action логів, яке відповідає певній дії користувача. Наприклад, наступне правило описує ситуацію, коли користувач проходить двофакторну аутентифікацію.

```
<rule id="100101" level="3">
  <field name="message.action">2FA_VERIFIED</field>
  <description>Двофакторна аутентифікація пройдена</description>
</rule>
```

Використання таких правил дозволяє інтегрувати вашу систему Node.js з Wazuh, забезпечуючи автоматизоване виявлення й реагування на підозрілі або критично важливі події.

Перейшовши за посиланням https://<ip-address>, користувач отримує доступ до веб-інтерфейсу менеджера Wazuh (рис. 3.8). У цьому інтерфейсі відображається загальний стан системи, події безпеки, активні агенти та інша аналітична інформація. На головній сторінці необхідно перейти до розділу “Agents” і знайти агента, якого було попередньо встановлено на цільовій системі. Важливо, щоб статус агента був "Active", що означає його успішне підключення до менеджера та коректну передачу логів (рис. 3.9)

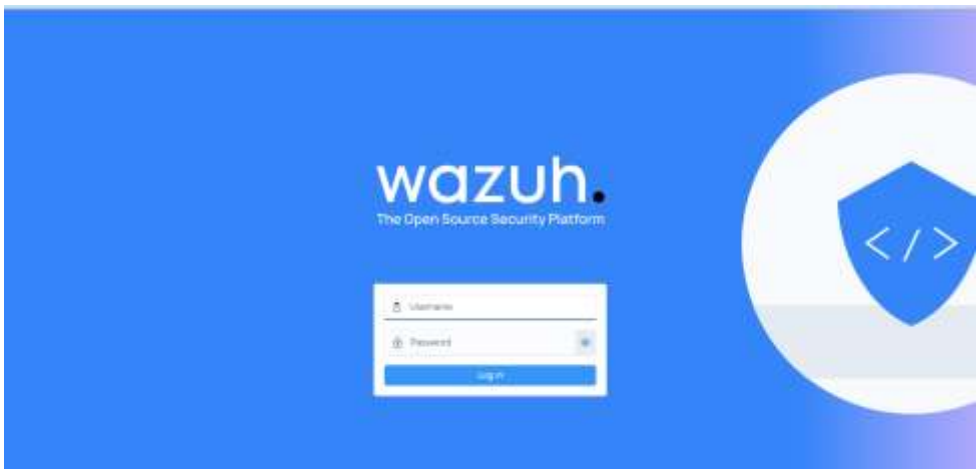


Рисунок 3.8 – Сторінка входу до сервісу

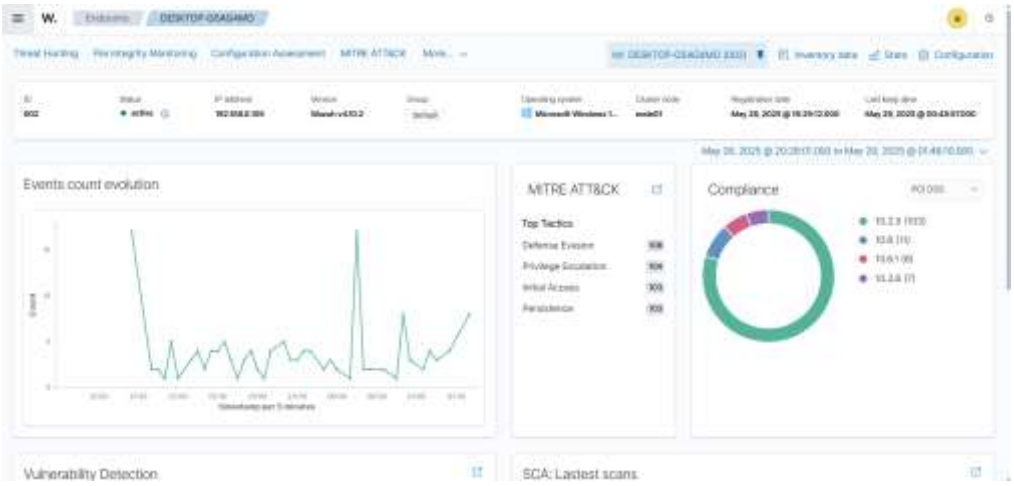


Рисунок 3.9 – Сторінка активного агента

Даний сервіс надає багато можливостей для перегляду даних, наприклад, перегляд правил що встановленні на кінцевому пристрої, але оскільки проєкт працює локально враховуються всі правила (рис. 3.10).

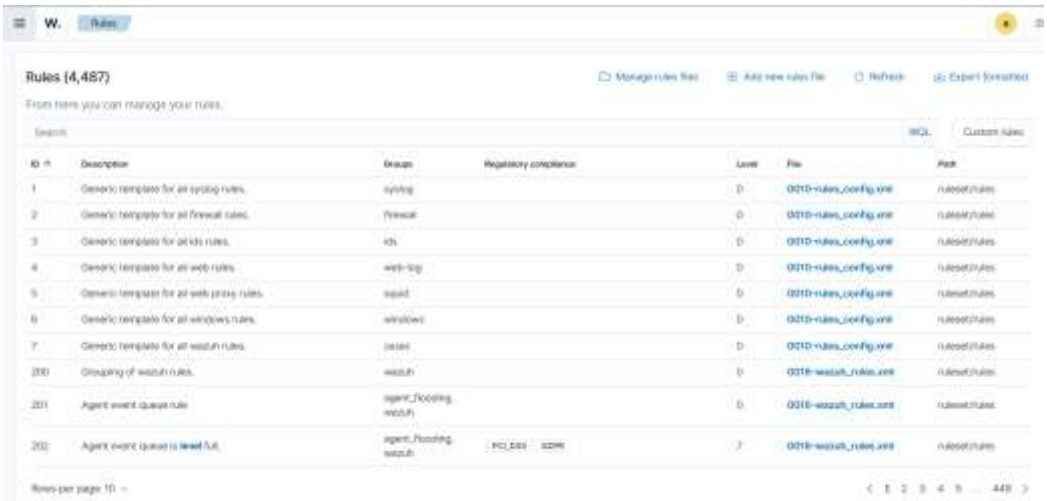


Рисунок 3.10 – Правила

Таким чином, впровадження SIEM-системи Wazuh дозволяє забезпечити централізований збір, зберігання та аналіз журналів подій з різних джерел. Завдяки гнучким механізмам конфігурації, підтримці користувацьких правил та широким можливостям візуалізації, Wazuh надає всі необхідні інструменти для своєчасного виявлення інцидентів безпеки, моніторингу активності користувачів та проведення глибокого аналізу зібраних даних.

3.7 Висновки до розділу 3

У цьому розділі було детально розглянуто основні компоненти програмної реалізації, що забезпечують функціонування розробленого веб-ресурсу. Описано ключові частини системи, що дають уялення про процеси що виконують і про загальну роботу ресурсу.

Особлива увага була приділена логіці обробки даних, механізмам взаємодії з користувачем та реалізації алгоритмів, що лежать в основі роботи системи.

Реалізація програмного продукту була здійснена із використанням сучасних інструментів та технологій, що дозволило забезпечити стабільну та ефективну роботу системи. Кожен компонент розроблявся з урахуванням вимог до масштабованості, модульності та повторного використання коду. Це дозволяє у подальшому легко адаптувати систему до нових вимог або інтегрувати з іншими рішеннями.

Під час реалізації програмного коду також було проведено тестування системи на працездатність. Було продемонстровано роботу основних алгоритмів, а саме: двофакторної аутентифікації, наскрізного шифрування та інтеграції SIEM- системи. Таким чином, реалізований веб-ресурс демонструє високу функціональність, безпечність та зручність у використанні. Застосування сучасних підходів до обробки даних, захисту інформації та взаємодії з користувачем забезпечує надійну роботу системи в умовах реального використання.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему «Розробка захищеного веб-ресурсу для обміну конфіденційними даними з використанням технології наскрізного шифрування End-to-End, двофакторної аутентифікації та SIEM-системи для моніторингу активності користувачів» було досягнуто поставленої мети – створено програмний продукт, який поєднує сучасні засоби інформаційної безпеки, забезпечуючи високий рівень захисту конфіденційної інформації при її передачі через мережу.

У процесі роботи проведено всебічний аналіз сучасних загроз, які стосуються веб-додатків, зокрема таких як перехоплення даних, фішинг, несанкціонований доступ та підміна особи. На основі аналізу сформульовано технічні та функціональні вимоги до системи, що дозволило чітко визначити підхід до її реалізації. Встановлено, що найбільш ефективним шляхом захисту є поєднання кількох технологій: наскрізного шифрування, багаторівневої аутентифікації та постійного моніторингу подій безпеки.

Реалізоване наскрізне шифрування (E2EE) забезпечує максимальну конфіденційність переданої інформації. Повідомлення шифруються на пристрої відправника і розшифровуються лише на пристрої отримувача. Таким чином, навіть у разі перехоплення переданих даних, жодна третя сторона, включно з сервером, не має змоги їх прочитати. Для цього було використано гібридну криптографічну модель, поєднання асиметричного та симетричного шифрування, що дозволяє зберегти як безпеку, так і продуктивність.

Впровадження двофакторної аутентифікації дозволило значно знизити ризик компрометації облікових записів користувачів. У роботі реалізовано схему підтвердження особистості шляхом надсилання одноразового коду (OTP) на електронну пошту, що стало додатковим бар’єром для несанкціонованого доступу навіть у випадку втрати чи викрадення основного пароля.

Особливу увагу приділено інтеграції SIEM-системи, зокрема, обрано платформу з відкритим кодом Wazuh, яка забезпечує гнучке логування, виявлення інцидентів та автоматичне повідомлення про загрози. Завдяки

інтеграції цієї системи було реалізовано моніторинг активності користувачів у режимі реального часу, що дозволяє оперативно реагувати на потенційні порушення безпеки.

Практичне значення роботи полягає в створенні комплексного рішення, яке може бути використане як основа для побудови безпечних веб-додатків, що працюють з чутливою інформацією. Застосовані підходи відповідають сучасним вимогам до безпеки та можуть бути масштабовані залежно від потреб конкретного проєкту чи організації.

Таким чином, виконана робота підтвердила ефективність комплексного підходу до забезпечення інформаційної безпеки та продемонструвала практичну можливість поєднання в одному веб-ресурсі. Отримані результати мають як теоретичну, так і прикладну цінність та можуть бути основою для подальших досліджень і вдосконалень у галузі захисту інформації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Осадчук, К., Салієва, О. (2025). Конференції ВНТУ електронні наукові видання. ВНТКП ВНТУ. Факультет менеджменту та інформаційної безпеки. URL: https://conferences.vntu.edu.ua/index.php/all-fm/all-fm-2025/paper/view/23117

2. Засоби захисту інформації. Stud. URL: https://stud.com.ua/94403/informatika/zasobi_zahistu_informatsiyi (дата звернення: 22.03.2025)

3. Інститут докторантури та аспірантури – Вінницький Національний Технічний Університет. URL: https://inmad.vntu.edu.ua/portal/static/E5C91F7E-6AD6-4B55-AC86-A49C2142ABB0.pdf (дата звернення: 22.05.2025)

4. Аутентифікація, авторизація та ідентифікація. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/ (дата звернення: 22.03.2025)

5. Державна служба спеціального зв’язку та захисту інформації України. Access Denied. URL: https://cip.gov.ua/ua/faqs/sho-take-dvofaktorna-avtentifikaciya-i-yak-vona-pracyuye (дата звернення: 22.03.2025)

6. Інститут комп'ютерних технологій / Захист Web-ресурсів. Інститут комп'ютерних технологій / Новини. URL: https://www.ict.com.ua/?lng=1&sec=21 (дата звернення: 22.03.2025)

7. All Lec (kszi) - Стр 16. StudFiles. URL: https://studfile.net/preview/9649925/page:16/ (дата звернення: 22.03.2025)

8. Що таке шифрування та як воно працює? - Kingston Technology. Kingston Technology Company. URL: https://www.kingston.com/ua/blog/data-security/what-is-encryption (дата звернення: 23.03.2025)

9. Наскрізне шифрування (E2EE): детальний огляд. IT Education Center Blog. URL: https://itedu.center/ua/blog/review/naskrizne-shyfruvannia-prostymy-slovamy-pro-e2ee/?srsltid=AfmBOopYuaQ7UaSIczJG-Nw1tNV1CnKo9ds-zdvdJB1XhY__TBN4BITy (дата звернення: 23.03.2025)

10. Державна служба спеціального зв’язку та захисту інформації України. Access Denied. URL: https://cip.gov.ua/ua/faqs/sho-take-end-to-end-shifruvannya-yake-chasto-zgaduyut-govoryachi-pro-mesendzheri (дата звернення: 23.03.2025)

11. IBM. What Is End-to-End Encryption? | IBM. IBM - United States. URL: https://www.ibm.com/think/topics/end-to-end-encryption (дата звернення: 23.03.2025)

12. Huber M. The 8 benefits you need to know about end-to-end encryption. Seald. URL: https://www.seald.io/blog/the-8-benefits-you-need-to-know-about-end-to-end-encryption (дата звернення: 23.03.2025)

13. Двофакторна автентифікація | PeopleForce База знань. PeopleForce help center. URL: https://help.peopleforce.io/uk/articles/6628189-двофакторна-автентифікація (дата звернення: 25.03.2025)

14. Oteir N. У чому різниця між аутентифікацією та авторизацією?. INTROSERV. URL: https://introserv.com/ua/blog/u-chomu-rizniczya-mizh-autentifikaczieyu-ta-avtorizaczieyu/ (дата звернення: 25.03.2025)

15. Oteir N. Одноразовий пароль та інші алгоритми аутентифікації: як вони використовуються в цифровій безпеці. INTROSERV. URL: https://introserv.com/ua/blog/digital-security-and-one-time-password-algorithms/ (дата звернення: 26.03.2025)

16. Barrett R. Easy Two Factor Authentication (2FA) with Google Authenticator & PHP. Medium. URL: https://medium.com/@richb_/easy-two-factor-authentication-2fa-with-google-authenticator-php-108388a1ea23 (дата звернення: 26.06.2025)

17. Види біометричної аутентифікації. Nixj. URL: https://nixj.ua/vidi-bometrichno-autentifkac (дата звернення: 27.03.2025)

18. Що таке SIEM? | Захисний комплекс Microsoft. Your request has been blocked. This could be due to several reasons. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-siem (дата звернення: 27.03.2025)

19. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Книшук А.В. Вступ до кібербезпеки: навч. посіб. – Кропивницький: ЦНТУ, 2022. – 967 с.

20. Security Information and Event Management (SIEM). Real Time Monitoring | Wazuh. Wazuh. URL: https://wazuh.com/platform/siem/ (дата звернення: 28.03.2025)

21. Graylog Security. Graylog. URL: https://graylog.org/products/security/ (дата звернення: 28.03.2025)

22. Splunk Enterprise Security | Splunk. Splunk. URL: https://www.splunk.com/en_us/products/enterprise-security.html (дата звернення: 28.03.2025)

23. SIEM. Logpoint. URL: https://www.logpoint.com/en/product/siem/ (дата звернення: 29.03.2025)

24. Why Securonix. Securonix. URL: https://www.securonix.com/why-securonix/ (дата звернення: 30.03.2025)

25. HTML: Creating the content - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Your_first_website/Creating_the_content (дата звернення: 14.04.2025)

26. Ukrainian W. Уроки від W3Schools українською онлайн. W3Schools українською. Безплатні уроки онлайн для початківців, школярів та студентів. URL: https://w3schoolsua.github.io/css/index.html#gsc.tab=0 (дата звернення: 14.04.2025)

27. JavaScript | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 15.04.2025)

28. Посібник: знайомство з React – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/tutorial/tutorial.html#what-is-react (дата звернення: 15.04.2025)

29. Node.js – Запускайте JavaScript будь-де. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/uk (дата звернення: 16.04.2025)

30. Getting Started. vitejs. URL: https://vite.dev/guide/ (дата звернення: 16.04.2025)

ДОДАТКИ

Додаток А. Технічне завдання

68

Додаток А. Технічне завдання

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

ЗАТВЕРДЖУЮ
Голова секції "Управління
інформаційною
безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:

«РОЗРОБКА ЗАХИЩЕНОГО ВЕБ-РЕСУРСУ ДЛЯ ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ НАСКРІЗНОГО ШИФРУВАННЯ END-TO-END, ДВОФАКТОРНОЇ АУТЕНТИФІКАЦІЇ ТА SIEM-СИСТЕМИ ДЛЯ МОНІТОРИНГУ АКТИВНОСТІ КОРИСТУВАЧІВ

08-72.БКР.013.00.070.ТЗ

Керівник бакалаврської кваліфікаційної роботи
д.ф., доц. каф. МБІС
Салієва О. В.
"20" березня

Вінниця – 2025 р.

1. Найменування та область застосування

Програмна реалізація захищеного веб-ресурсу для обміну конфіденційними даними з використанням технології наскрізного шифрування End-to-End, двофакторної аутентифікації та SIEM-системи для моніторингу активності. Область застосування: організації, що потребують захищеного обміну конфіденційною інформацією

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: метою розробки є створення безпечного веб-ресурсу, який забезпечує конфіденційний обмін даними між користувачами шляхом впровадження сучасних засобів інформаційної безпеки.

3.2 Призначення: полягає у забезпеченні захищеного середовища для передачі чутливої інформації в мережі Інтернет, що дозволяє попереджати несанкціонований доступ, відстежувати підозрілу активність та гарантувати цілісність і конфіденційність.

4. Джерела розробки

4.1. Аутентифікація, авторизація та ідентифікація. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/ (дата звернення: 22.03.2025)

4.2. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Книшук А.В. Вступ до кібербезпеки: навч. посіб. – Кропивницький: ЦНТУ, 2022. – 967 с.

4.3. 20. Security Information and Event Management (SIEM). Real Time Monitoring | Wazuh. Wazuh. URL: https://wazuh.com/platform/siem/ (дата звернення: 28.03.2025)

4.4. 9. Наскрізне шифрування (E2EE): детальний огляд. IT Education Center Blog. URL: https://itedu.center/ua/blog/review/naskrizne-shyfruvannia-prostymy-

slovamy-pro-e2ee/?srsltid=AfmBOopYuaQ7UaSIczJG-Nw1tNV1CnKo9ds-zdvdJB1XhY__TBN4BITy (дата звернення: 23.03.2025)

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

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

5.1.2 Система має підтримувати захищений обмін повідомленнями між зареєстрованими користувачами з використанням наскрізного шифрування;

5.1.3 Веб-ресурс повинен забезпечувати реєстрацію та автентифікацію користувачів із використанням двофакторної аутентифікації.

5.1.4 Система має фіксувати дії користувачів та події безпеки

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен забезпечувати безперервну роботу веб-ресурсу

5.2.2 У разі збоїв або втрати з’єднання дані користувача не повинні бути втрачені;

5.2.3 Усі критичні дії в системі (вхід, вихід, зміна пароля, невдалі спроби входу) мають бути зафіксовані.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – AMD Ryzen 5 7535HS і подібні до них;

– оперативна пам’ять – не менше 8 Гб;

– середовище функціонування – операційна система сімейство Windows, віртуальна машина для Linux;

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

6. Вимоги до програмної документації

6.1 Програмна документація до розробленого веб-ресурсу повинна містити повний опис структури, функціональних можливостей та правил використання програмного продукту.

71

7. Вимоги до технічного захисту інформації

7.1 Необхідно впровадження систем аутентифікації для запобігання несанкціонованому доступу до облікових записів користувачів.

7.2 Захист конфіденційної інформації шляхом кодування даних під час збереження та передачі.

8. Техніко-економічні показники

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

8.2 Проект повинен бути реалізований з урахуванням можливості його широкого використання, що забезпечить доступність і практичну цінність.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	21.03.2025	23.03.2025
2.	Аналіз предметної області обраної теми	24.03.2025	20.05.2025
3.	Розробка алгоритму роботи	21.05.2025	15.05.2025
4.	Написання бакалаврської роботи на основі розробленої теми	16.05.2025	25.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	28.05.2025	06.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	11.06.2025	16.05.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____ Осадчук К.Д.

Додаток Б. Лістинг програми (серверна частина)

```
import express from 'express';
import pino from 'pino-http';
import cors from 'cors';
import { getEnvVar } from './utils/getEnvVar.js';
import { notFoundHandler } from './middlewares/notFoundHandler.js';
import { errorHandler } from './middlewares/errorHandler.js';
import router from './routers/index.js';
import cookieParser from 'cookie-parser';
import { corsConfigs } from './configs/corsConfigs.js';
const PORT = Number(getEnvVar('PORT', '3000'));
export const startServer = () => {
  const app = express();
  app.use(express.json());
  app.use(cors(corsConfigs));
  app.use(cookieParser());
  app.use(
    pino({
      transport: {
        target: 'pino-pretty',
      },}));
  app.use(router);
  app.use('/uploads', express.static(UPLOAD_DIR));
  app.use(errorHandler);
  app.use(notFoundHandler);
  app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);});};
import { TEMP_UPLOAD_DIR, UPLOAD_DIR } from './constants/index.js';
import { initMongoDB } from './db/initMongoDB.js';
import { startServer } from './server.js';
import { createDirIfNotExists } from './utils/createDirIfNotExists.js';
const bootstrap = async () => {
  await initMongoDB();
  await createDirIfNotExists(TEMP_UPLOAD_DIR);
  await createDirIfNotExists(UPLOAD_DIR);
  startServer();
};
void bootstrap();
import createHttpError from 'http-errors';
import { UsersCollection } from '../db/models/users.js';
import bcrypt from 'bcrypt';
import { SessionsCollection } from '../db/models/session.js';
import { randomBytes } from 'crypto';
import { FIFTEEN_MINUTES, ONE_DAY, SMTP, TEMPLATES_DIR,} from
'../constants/index.js';
import jwt from 'jsonwebtoken';
import { getEnvVar } from '../utils/getEnvVar.js';
import { sendEmail } from '../utils/sendMail.js';
import handlebars from 'handlebars';
import path from 'node:path';
import fs from 'node:fs/promises';
```

```
export const registerUser = async payload => {
  const user = await UsersCollection.findOne({ email: payload.email });
  if (user) throw createHttpError(409, 'Email in use');
  const encryptedPassword = await bcrypt.hash(payload.password, 10);
  return await UsersCollection.create({
    ...payload,
    password: encryptedPassword,});};
export const loginUser = async payload => {
  const user = await UsersCollection.findOne({ email: payload.email });
  if (!user) {
    throw createHttpError(404, 'User not found');
  }
  const isEqual = await bcrypt.compare(payload.password, user.password);
  if (!isEqual) {
    throw createHttpError(401, 'Unauthorized');
  }
  await SessionsCollection.deleteOne({ userId: user._id });
  const accessToken = randomBytes(30).toString('base64');
  const refreshToken = randomBytes(30).toString('base64');
  return await SessionsCollection.create({
    userId: user._id,
    accessToken,
    refreshToken,
    accessTokenValidUntil: new Date(Date.now() + FIFTEEN_MINUTES),
    refreshTokenValidUntil: new Date(Date.now() + ONE_DAY),
  });};
export const logoutUser = async sessionId => {
  await SessionsCollection.deleteOne({ _id: sessionId });};
export const refreshUsersSession = async ({ sessionId, refreshToken }) => {
  const session = await SessionsCollection.findOne({
    _id: sessionId,
    refreshToken,
  });
  if (!session) {
    throw createHttpError(401, 'Session not found');
  }
  const isSessionTokenExpired =
    new Date() > new Date(session.refreshTokenValidUntil);
  if (isSessionTokenExpired) {
    throw createHttpError(401, 'Session token expired');
  }
  const newSession = createSession();
  await SessionsCollection.deleteOne({ _id: sessionId, refreshToken });
  return await SessionsCollection.create({
    userId: session.userId,
    ...newSession,
  });};
export const requestResetToken = async email => {
  const user = await UsersCollection.findOne({ email });
  if (!user) {throw createHttpError(404, 'User not found');}
  const resetToken = jwt.sign(
    {
      sub: user._id,
```

```
      email,
    },
    getEnvVar('JWT_SECRET'),
    { expiresIn: '15m',});
  const resetPasswordTemplatePath = path.join(
    TEMPLATES_DIR,  'reset-password-email.html'
  );
  const templateSource = (
    await fs.readFile(resetPasswordTemplatePath)
  ).toString();
  const template = handlebars.compile(templateSource);
  const html = template({
    name: user.name,
    link: `${getEnvVar('APP_DOMAIN')}/reset-password?token=${resetToken}`,
  });
  await sendEmail({
    from: getEnvVar(SMTP.SMTP_FROM),
    to: email,
    subject: 'Reset your password',
    html,
  });};
export const resetPassword = async payload => {
  let entries;
  try {
    entries = jwt.verify(payload.token, getEnvVar('JWT_SECRET'));
  } catch (err) {
    if (err instanceof Error) throw createHttpError(401, err.message);
    throw err;
  }
  const user = await UsersCollection.findOne({
    email: entries.email,
    _id: entries.sub,
  });
  if (!user) {
    throw createHttpError(404, 'USer not found');
  }
  const encryptedPassword = await bcrypt.hash(payload.password, 10);
  await UsersCollection.updateOne(
    { _id: user._id },
    { password: encryptedPassword }
  );};
export const createSession = async userId => {
  const accessToken = randomBytes(30).toString('base64');
  const refreshToken = randomBytes(30).toString('base64');
  const session = await SessionsCollection.create({
    userId, accessToken, refreshToken,
    accessTokenValidUntil: new Date(Date.now() + FIFTEEN_MINUTES),
    refreshTokenValidUntil: new Date(Date.now() + ONE_DAY),
  });
  return {
    _id: session._id,
    accessToken: session.accessToken,
    refreshToken: session.refreshToken, };};
```

```
import { SORT_ORDER } from '../constants/index.js';
import { UsersCollection } from '../db/models/users.js';
import { calculatePaginationData } from '../utils/calculatePaginationData.js';
export const getAllUsers = async ({
  page = 1,
  perPage = 10,
  sortOrder = SORT_ORDER.ASC,
  sortBy = '_id',
}) => {
  const limit = perPage;
  const skip = (page - 1) * perPage;
  const usersQuery = UsersCollection.find();
  const userCount = await UsersCollection.find()
    .merge(usersQuery)
    .countDocuments();
  const users = await usersQuery
    .skip(skip)
    .limit(limit)
    .sort({ [sortBy]: sortOrder })
    .exec();
  const paginationData = calculatePaginationData(userCount, perPage, page);
  return {
    data: users,
    ...paginationData,
  };};
export const getUsersById = async usersId => {
  const user = await UsersCollection.findById(usersId);
  return user;
};
export const createUser = async payload => {
  const users = await UsersCollection.create(payload);
  return users;
};
export const deleteUser = async userId => {
  const user = await UsersCollection.findOneAndDelete({
    _id: userId, });
  return user;
};
export const updateUser = async (userId, payload, options = {}) => {
  const rawResult = await UsersCollection.findOneAndUpdate(
    { _id: userId },
    payload,
    {
      new: true,
      includeResultMetadata: true,
      ...options,
    });
  if (!rawResult || !rawResult.value) return null;
  return {
    user: rawResult.value,
    isNew: Boolean(rawResult?.lastErrorObject?.upserted),
  };};
import { Router } from 'express';
```

```
import UsersRouter from './users.js';
import authRouter from './auth.js';
import ChatRoute from './ChatRoute.js';
import MessageRoute from './messageRoute.js';
import logsRouter from './logsRouter.js';
const router = Router();
router.use('/users', UsersRouter);
router.use('/auth', authRouter);
router.use('/chat', ChatRoute);
router.use('/message', MessageRoute);
router.use('/logs', logsRouter);
export default router;
import { Router } from 'express';
import { ctrlWrapper } from '../utils/ctrlWrapper.js';
import {loginUserSchema, registerUserSchema, requestResetEmailSchema, resetPasswordSchema,} from '../validation/users.js';
import {loginUserController, logoutUserController, refreshUsersSessionController,
  registerUserController, requestResetEmailController, resetPasswordController,
  verify2FAController,} from '../controllers/auth.js';
const router = Router();
router.post(
  '/register',validateBody(registerUserSchema), ctrlWrapper(registerUserController));
router.post(
  '/login', validateBody(loginUserSchema), ctrlWrapper(loginUserController));
router.post('/verify-2fa', ctrlWrapper(verify2FAController));
router.post('/logout', ctrlWrapper(logoutUserController));
router.post('/refresh', ctrlWrapper(refreshUsersSessionController));
router.post( '/request-reset-email', validateBody(requestResetEmailSchema),
  ctrlWrapper(requestResetEmailController));
router.post( '/reset-password', validateBody(resetPasswordSchema),
  ctrlWrapper(resetPasswordController));
export default router;
import { Router } from 'express';
import {createUserController, deleteUserController, getUserPublicKey, getUsersByIdController, getUsersController, patchUserController, uploadPublicKey, upsertUserController,} from '../controllers/users.js';
import { ctrlWrapper } from '../utils/ctrlWrapper.js';
import { validateBody } from '../middlewares/validateBody.js';
import { createUserSchema } from '../validation/users.js';
import { isValidId } from '../middlewares/isValidId.js';
import { authenticate } from '../middlewares/authenticate.js';
import { upload } from '../middlewares/multer.js';
const router = Router();
router.use(authenticate);
router.get('/', ctrlWrapper(getUsersController));
router.get('/:userId', isValidId, ctrlWrapper(getUsersByIdController));
router.post('/', upload.single('photo'), validateBody(createUserSchema),
  ctrlWrapper(createUserController));
router.delete('/:userId', isValidId, ctrlWrapper(deleteUserController));
router.put( '/:userId',  isValidId, upload.single('photo'), validateBody(createUserSchema),
  ctrlWrapper(upsertUserController));
router.patch( '/:userId', isValidId, upload.single('photo'), ctrlWrapper(patchUserController));
router.post('/public-key', uploadPublicKey);
```

```
router.get('/:id/public-key', getUserPublicKey);
export default router;
import {  createSession, loginUser, logoutUser, refreshUsersSession, registerUser,
  requestResetToken, resetPassword,} from '../services/auth.js';
import { ONE_DAY, SMTP } from '../constants/index.js';
import { UsersCollection } from '../db/models/users.js';
import { sendEmail } from '../utils/sendMail.js';
import { getEnvVar } from '../utils/getEnvVar.js';
import { logActivity } from '../utils/logActivity.js';
import { LogsCollection } from '../db/models/logs.js';
export const registerUserController = async (req, res) => {
  const user = await registerUser(req.body);
  res.status(201).json({
    status: 201,
    message: 'Successfully registered a user!',
    data: user,
  }); };
export const loginUserController = async (req, res) => {
  try {
    const session = await loginUser(req.body);
    const user = await UsersCollection.findById(session.userId).select('-password');
    user.isTwoFactorVerified = false;
    await logActivity({
      req,
      userId: user._id,
      action: 'LOGIN_ATTEMPT',
      metadata: { email: user.email },});
    if (!user.isTwoFactorVerified) {
      const code = crypto.randomInt(100000, 999999).toString();
      user.twoFactorCode = code;
      user.twoFactorExpires = new Date(Date.now() + 10 * 60 * 1000); // 10 хв
      await user.save();
      await sendEmail({
        to: user.email,
        subject: '2FA Verification Code',
        text: `Your verification code is: ${code}`,
        html: `<p>Your verification code is: <strong>${code}</strong></p>`,
        from: `"Security Team" <${getEnvVar(SMTP.SMTP_USER)}>`,
       });
      await logActivity({
        req,
        userId: user._id,
        action: '2FA_CODE_SENT',
        metadata: { email: user.email },
      });
      return res.status(200).json({
        status: 200,
        message: 'Verification code sent to your email',
        data: {
          requires2FA: true,
          email: user.email,
        },});}
    await logActivity({
```

```
      req,
      userId: user._id,
      action: 'LOGIN_SUCCESS',
      metadata: { email: user.email },
    });
    res.cookie('refreshToken', session.refreshToken, {
      httpOnly: true,
      expires: new Date(Date.now() + ONE_DAY),
    });
    res.cookie('sessionId', session._id, {
      httpOnly: true,
      expires: new Date(Date.now() + ONE_DAY),
    });
    res.json({
      status: 200,
      message: 'Successfully logged in a user',
      data: {
        accessToken: session.accessToken,
        user,
      },});
  } catch (error) {
    await logActivity({
      req,
      userId: null,
      action: 'LOGIN_FAILED',
      metadata: {
        email: req.body?.email || 'unknown',
        reason: error.message,
      },});
    return res.status(401).json({ message: 'Invalid credentials' });
  }};
export const logoutUserController = async (req, res) => {
  if (req.cookies.sessionId) {
    const session = await logoutUser(req.cookies.sessionId);
    await logActivity({
      req,
      userId: session?.userId || null,
      action: 'LOGOUT',
    });}
  res.clearCookie('sessionId');
  res.clearCookie('refreshToken');
  res.status(204).send();
};
const setupSession = (res, session) => {
  res.cookie('refreshToken', session.refreshToken, {
    httpOnly: true,
    expires: new Date(Date.now() + ONE_DAY),
  });
  res.cookie('sessionId', session._id, {
    httpOnly: true,
    expires: new Date(Date.now() + ONE_DAY),
  });};
export const refreshUsersSessionController = async (req, res) => {
```

```
  const session = await refreshUsersSession({
    sessionId: req.cookies.sessionId,
    refreshToken: req.cookies.refreshToken,
  });
  setupSession(res, session);
  const user = await UsersCollection.findById(session.userId).select('-password');
  res.json({
    status: 200,
    message: 'Successfully refreshed a session',
    data: {
      accessToken: session.accessToken,
      user,
    },});};
export const requestResetEmailController = async (req, res) => {
  await requestResetToken(req.body.email);
  res.json({
    status: 200,
    message: 'Reset password email was successfully',
    data: {},
  });};
export const resetPasswordController = async (req, res) => {
  try {
    await resetPassword(req.body);
    await logActivity({
      req,
      userId: null,
      action: 'RESET_PASSWORD_SUCCESS',
      metadata: { email: req.body.email },
    });
    res.json({
      message: 'Password was successfully reset!',
      status: 200,
      data: {},
    });
  } catch (error) {
    await logActivity({
      req,
      userId: null,
      action: 'RESET_PASSWORD_FAILED',
      metadata: {
        email: req.body.email,
        reason: error.message,
      },});
    return res.status(400).json({ message: 'Reset failed' });
  }};
export const verify2FAController = async (req, res) => {
  const { email, code } = req.body;
  const user = await UsersCollection.findOne({ email });
  if (!user) {
    return res.status(400).json({ message: 'User not found' });
  }
  if (
    !user.twoFactorCode ||
```

```
    !user.twoFactorExpires ||
    user.twoFactorCode !== code ||
    user.twoFactorExpires < new Date()
  ) {
    await logActivity({
      req,
      userId: user._id,
      action: '2FA_FAILED',
      metadata: {
        reason: 'Invalid or expired verification code',
        email,
      },});
    return res
      .status(400)
      .json({ message: 'Invalid or expired verification code' });
  }
  user.twoFactorCode = undefined;
  user.twoFactorExpires = undefined;
  user.isTwoFactorVerified = true;
  await user.save();
  await logActivity({
    req,
    userId: user._id,
    action: '2FA_VERIFIED',
  });
  const session = await createSession(user._id);
  res.cookie('refreshToken', session.refreshToken, {
    httpOnly: true,
    expires: new Date(Date.now() + ONE_DAY),
  });
  res.cookie('sessionId', session._id, {
    httpOnly: true,
    expires: new Date(Date.now() + ONE_DAY),
  });
  res.json({
    status: 200,
    message: '2FA verification successful. Logged in.',
    data: {
      accessToken: session.accessToken,
      user: {
        _id: user._id,
        email: user.email,
        name: user.name,
      },},});};
export const getUserLogsController = async (req, res) => {
  const logs = await LogsCollection.find({ userId: req.params.userId }).sort({ createdAt: -1,});
  res.json({
    status: 200,
    message: 'User logs retrieved',
    data: logs,
  });};
import createHttpError from 'http-errors';
import {
```

```
  createUser, deleteUser, getAllUsers, getUsersById, updateUser,
} from '../services/users.js';
import { parsePaginationParams } from '../utils/parsePaginationParams.js';
import { parseSortParams } from '../utils/parseSortParams.js';
import { saveFileToUploadDir } from '../utils/saveFileToUploadDir.js';
import { getEnvVar } from '../utils/getEnvVar.js';
import { saveFileToCloudinary } from '../utils/saveFileToCloudinary.js';
import { UsersCollection } from '../db/models/users.js';
export const getUsersController = async (req, res, next) => {
  try {
    const { page, perPage } = parsePaginationParams(req.query);
    const { sortBy, sortOrder } = parseSortParams(req.query);
    const users = await getAllUsers({
      page, perPage, sortBy, sortOrder,
    });
    res.json({
      status: 200,
      message: 'Successfully found users',
      data: users,
    });
  } catch (err) {
    next(err);
  }};
export const getUsersByIdController = async (req, res, next) => {
  const { userId } = req.params;
  const user = await getUsersById(userId);
  if (!user) {
    throw createHttpError(404, 'User not found');
  }
  res.json({
    status: 200,
    message: `Successfully found user with id ${userId}!`,
    data: user,
  });};
export const createUserController = async (req, res) => {
  const user = await createUser(req.body);
  res.status(201).json({
    status: 201,
    message: 'Successfully created a student',
    data: user,
  });};
export const deleteUserController = async (req, res, next) => {
  const { userId } = req.params;
  const user = await deleteUser(userId);
  if (!user) {
    next(createHttpError(404, 'User not found'));
    return;
  }
  res.status(204).send();
};
export const upsertUserController = async (req, res, next) => {
  const { userId } = req.params;
  const result = await updateUser(userId, req.body, {
```

```
    upsert: true,
  });
  if (!result) {
    next(createHttpError(404, 'User not found'));
  }
  const status = result.isNew ? 201 : 200;
  res.status(status).json({
    status,
    message: 'successfully upserted a user',
    data: result.user,
  });};
export const patchUserController = async (req, res, next) => {
  const { userId } = req.params;
  const photo = req.file;
  let photoUrl;
  if (photo) {
    if (getEnvVar('ENABLE_CLOUDINARY') === 'true') {
      photoUrl = await saveFileToCloudinary(photo);
    } else { photoUrl = await saveFileToUploadDir(photo); }}
  const result = await updateUser(userId, { ...req.body, photo: photoUrl });
  if (!result) {
    next(createHttpError(404, 'User not found'));
    return;
  }
  res.status(200).json({
    status: 200,
    message: 'Successfully patched a user',
    data: result.user,
  });};
export const uploadPublicKey = async (req, res) => {
  try {
    const { publicKey } = req.body;
    const userId = req.user.id;
   if (!publicKey) {
      return res.status(400).json({ message: 'Public key is required' });
    }
    await UsersCollection.findByIdAndUpdate(userId, { publicKey });
    return res.status(200).json({ message: 'Public key saved successfully' });
  } catch (error) {
    console.error('Error saving public key:', error);
    return res.status(500).json({ message: 'Server error' }); }};
export const getUserPublicKey = async (req, res) => {
  try {
    const user = await UsersCollection.findById(req.params.id).select(
      'publicKey'
    );
    if (!user || !user.publicKey) {
      return res.status(404).json({ message: 'Public key not found' });
    }
    return res.status(200).json({ publicKey: user.publicKey });
  } catch (error) {
    console.error('Error fetching public key:', error);
    res.status(500).json({ message: 'Server error' });  }};
```

Додаток В. Ілюстравтиний матеріал

РОЗРОБКА ЗАХИЩЕНОГО ВЕБ-РЕСУРСУ ДЛЯ ОБМІНУ КОНФІДЕНЦІЙНИМИ ДАНИМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ НАСКРІЗНОГО ШИФРУВАННЯ END-TO-END, ДВОФАКТОРНОЇ АУТЕНТИФІКАЦІЇ ТА SIEM-СИСТЕМИ ДЛЯ МОНІТОРИНГУ АКТИВНОСТІ КОРИСТУВАЧІВ

Виконала студентка групи 2КІТС-21б Осадчук Катерина Дмитрівна
Керівник роботи Салієва Ольга Володимирівна д.ф., доц. каф. МБІС

ВІННИЦЯ-2025

Актуальність

У сучасному цифровому світі передача конфіденційної інформації через веб-ресурси супроводжується численними ризиками: перехоплення, несанкціонований доступ, фішинг тощо. Актуальність цієї проблеми зростає разом із обсягом чутливої інформації, що циркулює онлайн. Ефективним підходом до захисту є комбінація кількох засобів захисту. Таке комплексне рішення формує надійний захист для веб-ресурсів, що працюють із персональними або службовими даними.

Новизна

Полягає в реалізації комплексного підходу до інформаційної безпеки веб-додатку з акцентом на інтеграцію засобів проактивного захисту. Такий підхід забезпечує не лише збереження конфіденційності, а й підвищення рівня довіри до веб-ресурсів користувачами.

Мета роботи

Метою даної роботи є розробка захищеного веб-ресурсу для обміну конфіденційними даними, що використовує сучасні засоби інформаційної безпеки, такі як наскрізне шифрування, використання двофакторної аутентифікації та SIEM-систем для моніторингу активності користувачів.

01 Об'єкт дослідження

Процес обміну конфіденційною інформацією в мережевому середовищі.

02 Предмет дослідження

Засоби та методи забезпечення інформаційної безпеки під час передачі даних, а також технічні рішення щодо аутентифікації та моніторингу активності користувачів ресурсів

03 Практична цінність

Полягає в розробці комплексного рішення, яке застосовується для захисту інформації з обмеженим доступом в веб-додатках, а також використання розробки як бази для подальших наукових, практичних досліджень у сфері кібербезпеки

Головні складові розробки

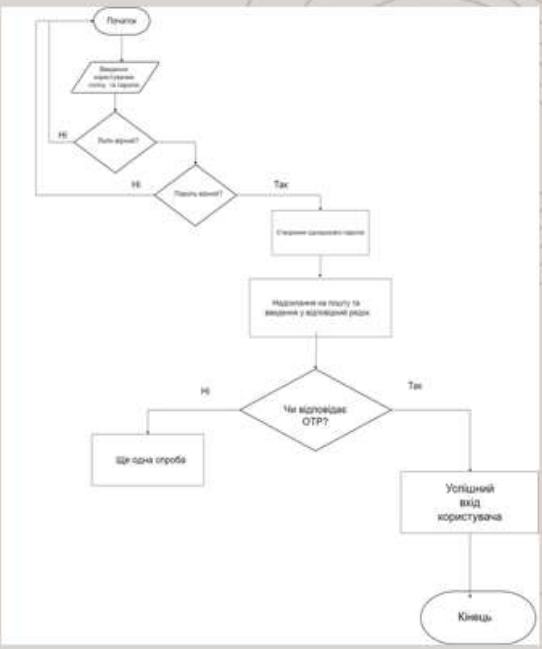
01 Наскрізне шифрування

02 Двофакторна аутентифікація

03 Інтеграція SIEM-системи

Двофакторна аутентифікація

Процес входу до системи з двофакторною аутентифікацією включає два етапи. Перший етап базова аутентифікація та другий – перевірка другого фактору.



Наскрізне шифрування

Під час реєстрації користувача генерується пара RSA-ключів: приватний зберігається на пристрої, а публічний передається на сервер. При надсиланні повідомлення генерується сеансовий AES-ключ, яким шифрується текст. Цей AES-ключ шифрується публічним ключем одержувача. На сервер передається зашифроване повідомлення разом із службовими даними. Одержувач розшифровує AES-ключ своїм приватним ключем і відновлює повідомлення. Усі криптографічні операції відбуваються лише на пристроях користувачів, що забезпечує повну конфіденційність даних.

Обрання SIEM-системи

У якості SIEM-рішення для моніторингу подій безпеки було обрано Wazuh — сучасну платформу з відкритим кодом, яка поєднує функції збору та аналізу логів, виявлення загроз, перевірки цілісності системи та контролю відповідності вимогам безпеки. Вона добре підходить для інтеграції з веб-додатками завдяки гнучкості налаштувань, безкоштовній ліцензії та підтримці реального часу.

wazuh.

Алгоритм роботи програми

01 Головна сторінка

Користувач потрапляє на головну сторінку, де має можливість пройти реєстрацію або авторизуватись.

02 Реєстрація

Під час реєстрації створюється обліковий запис користувача, генерується пара RSA-ключів.

03 Вхід (логін)

Користувач вводить свої облікові дані у форму входу. При правильних даних виконується наступний пукт.

04 Двофакторна аутентифікація (2FA)

Система надсилає одноразовий код (OTP) на email. Користувач вводить код у відповідне поле: якщо код вірний — здійснюється вхід; невірний — виводиться помилка.

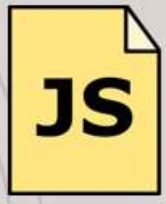
05 Інтерфейс чатів

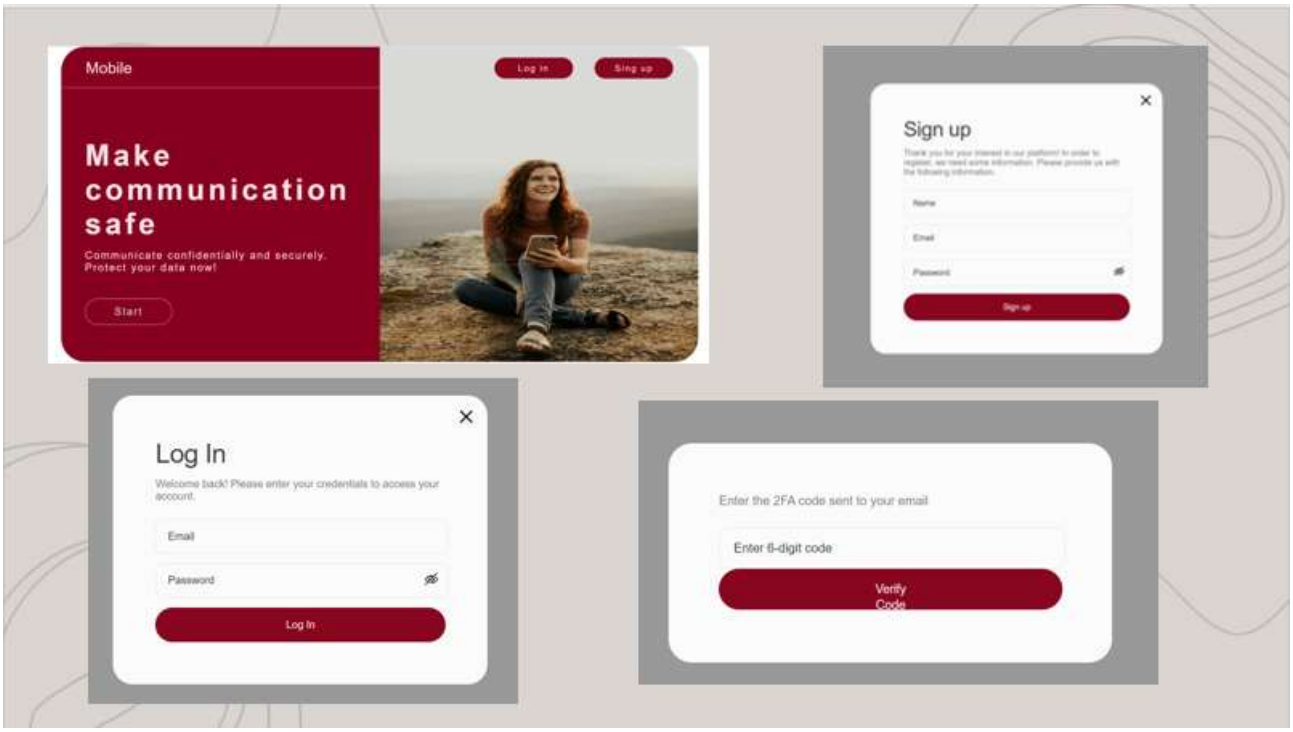
Після успішної аутентифікації користувач потрапляє на основну сторінку з чатами, де може переходити між діалогами, а також отримати доступ до меню з налаштуваннями або вийти з облікового запису.

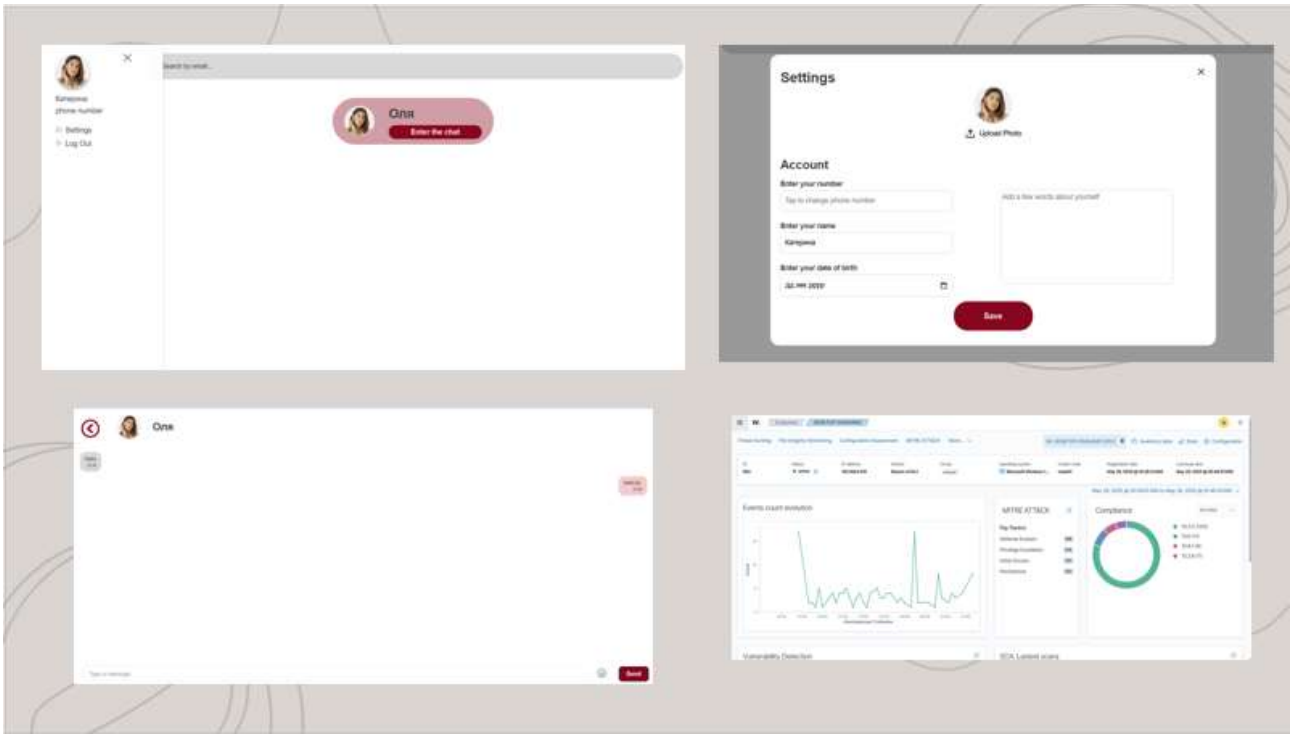
06 Обмін повідомленнями

Під час обміну повідомленнями кожне повідомлення шифрується на пристрої відправника за принципом наскрізного шифрування (E2EE), передається через сервер у зашифрованому вигляді та розшифровується виключно на стороні отримувача. Завдяки цьому сервер не має доступу до вмісту повідомлення, що гарантує повну конфіденційність.

Обрані інструменти розробки




Mobile
Log in
Sing up
Make communication safe
Communicate confidentially and securely. Protect your data now!
Start
Sign up
Name
Email
Password
Sign up
Log In
Welcome back! Please enter your credentials to access your account.
Email
Password
Log In
Enter the 2FA code sent to your email
Enter 6-digit code
Verify Code


Ona
Settings
Upload Photo
Account
Enter your number
Enter your name
Karajeva
Enter your date of birth
Save
Ona
Send

Результати та висновки

У межах бакалаврської роботи було:

- проаналізовано методи захисту;
- сформовано технічні та функціональні вимоги до захищеного веб-ресурсу;
- реалізовано механізм наскрізного шифрування;
- впроваджено двофакторну аутентифікацію з OTP-кодом на електронну пошту;
- обрано та інтегровано SIEM-систему Wazuh;
- здійснено тестування функціональності та захищеності системи.

Результати роботи підтверджують, що поєднання наскрізного шифрування, двофакторної аутентифікації та SIEM-системи є дієвим способом захисту веб-додатків від сучасних загроз. Запропоноване рішення дозволяє забезпечити захищену передачу конфіденційної інформації, контроль доступу до облікових записів та оперативне реагування на інциденти безпеки. Такий підхід може бути впроваджений у веб-системах, які працюють із персональними або службовими даними, та легко адаптується до потреб різних проєктів.

Дякую за увагу

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка захищеного веб-ресурсу для обміну конфіденційними даними з використанням технології наскрізного шифрування End-to-End, двофакторної аутентифікації та SIEM-системи для моніторингу активності користувачів.

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 2КІТС-21б

Керівник доцент Салієва О.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	89,89%
Загальна схожість	10,11%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Осадчук К.Д.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту.**

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

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