Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES»

Виконав: ст. 2-го курсу, групи КІТС-22мз
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Катричко Ю.В.
(прізвище та ініціали)

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

Опонент: к.т.н., доцент, каф. ОТ
Колесник І.С.
(прізвище та ініціали)
«____» _____ 2024 р.

Допущено до захисту
Голова секції УБ кафедри МБІС

Юрій ЯРЕМЧУК
“_____” _____2024 р.

Вінниця ВНТУ – 2024 рік

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

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

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
_____ **Юрій ЯРЕМЧУК**
“ ” _____ 2024 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Катричко Юрій Вікторович
(прізвище, ім`я, по-батькові)

1. Тема роботи:

«Підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES»
Керівник роботи_:_ Карпінець В.В. к.т.н., доцент каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “ 11” березня 2024 року № 81
2. Строк подання студентом роботи_за тиждень до захисту.__
3. Вихідні дані до роботи:
Стандарти, електронні джерела, підручники та наукові статті по темі, які стосуються теми магістерської кваліфікаційної роботи.
4. Зміст текстової частини:
У вступі викладено актуальність теми дослідження та сформульовано мету роботи.
У першому розділі розглянуті теоретичні аспекти безпеки протоколу WebSocket, включаючи аналіз існуючих стандартів та протоколів безпеки в Інтернеті, особливості роботи та застосування протоколу WebSocket та проблеми безпеки, що виникають при його використанні. Другий розділ присвячено проектуванню та розробці вдосконаленої системи захисту для протоколу WebSocket..

Третій розділ зосереджений на програмній реалізації розробленої системи захисту.. Четвертий розділ містить економічне обґрунтування розробки програмного забезпечення.

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

У дугому розділі магістерської кваліфікаційної роботи наведено 2 рисунка, у третьому розділі – 10 рисунків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина			
I	Карпінець В.В. к.т.н., доц. каф. МБІС		
II	Карпінець В.В. к.т.н., доц. каф. МБІС		
III	Карпінець В.В. к.т.н., доц. каф. МБІС		
Економічна частина			
IV	Причепа І.В., к.е.н., доц. каф. ЕПВМ		

7.Дата видачі завдання _____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напрямку магістерської роботи, формулювання теми	1.03.2024	15.03.2024	1.
2.	Аналіз предметної області обраної теми	15.03.2024	1.04.2024	2.
3.	Розробка роботи	1.04.2024	10.04.2024	3.
4.	Написання магістерської роботи на основі розробленої теми	10.04.2024	20.04.2024	4.
5.	Передзахист магістерської кваліфікаційної роботи	20.04.2024	10.05.2024	5.
6.	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	10.05.2024	4.06.2024	6.
7.	Захист магістерської кваліфікаційної роботи	11.06.2024	11.06.2024	7.

Студент _____ Катричко Ю.В.
(підпис)

Керівник роботи _____ Карпінець В.В.
(підпис)

АНОТАЦІЯ

УДК 004.56.5(043.2)

Катричко Ю.В. Підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES

Магістерська кваліфікаційна робота зі спеціальності 125 – «Кібербезпека», освітня програма «Кібербезпека інформаційних технологій та систем». Вінниця: ВНТУ, 2024. 114с.

На укр.мові. Бібліогр.: 41 назв; рис.: 19; табл. 13

У магістерській кваліфікаційній роботі розглядаються сучасні методи забезпечення безпеки протоколу WebSocket. Вступна частина роботи окреслює актуальність теми та основні завдання дослідження.

Перший розділ присвячений теоретичним засадам забезпечення захисту протоколу WebSocket. У ньому розглядаються загальні стандарти та протоколи безпеки в Інтернеті, особливості роботи та застосування протоколу WebSocket, а також проблеми його безпеки.

Другий розділ описує створення та розробку системи захисту для протоколу WebSocket. У ньому розглядаються особливості підвищення захисту протоколу вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та застосуванням механізмів криптографічної безпеки. Також представлено розробку алгоритму інтеграції системи захисту у протокол WebSocket.

Третій розділ присвячений програмній реалізації розробленої системи захисту. Розглядаються вибір мови програмування та середовища розробки, програмна реалізація, тестування та аналіз роботи системи.

Четвертий розділ містить економічну частину, де оцінюється комерційний потенціал розробленого програмного забезпечення, прогнозуються витрати на виконання наукової роботи та впровадження її результатів.

Ключові слова: WebSocket, AES, унікальні ідентифікатори

ABSTRACT

Yu.V. Katrychko Increasing the protection of the web socket protocol by an improved system with the introduction of unique access indicators and the AES encryption algorithm

Master's qualification thesis on specialty 125 - "Cybersecurity", educational program "Cybersecurity of information technologies and systems". Vinnytsia: VNTU, 2024. 130p.

In the Ukrainian language. Bibliography: 41 titles; Fig.: 19; table 13

The master's thesis examines modern methods of ensuring the security of the WebSocket protocol. The introductory part of the work outlines the relevance of the topic and the main tasks of the research.

The first section is devoted to the theoretical principles of securing the WebSocket protocol. It examines common Internet security standards and protocols, the specifics of WebSocket operation and applications, and its security issues.

The second section describes the creation and development of a security system for the WebSocket protocol. It examines the features of increasing the protection of the protocol by an improved system with the introduction of unique access identifiers and the use of cryptographic security mechanisms. The development of the algorithm for integrating the protection system into the WebSocket protocol is also presented.

The third section is devoted to the software implementation of the developed protection system. The choice of programming language and development environment, software implementation, testing and analysis of system operation are considered.

The fourth section contains the economic part, where the commercial potential of the developed software is evaluated, the costs of carrying out scientific work and the implementation of its results are forecast, the commercial effects of the implementation of the development results, and the effectiveness of the investments and their payback period are calculated.

Keywords: WebSocket, AES, unique identifiers

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, де інтернет-технології відіграють ключову роль у повсякденному житті, забезпечення безпеки передачі даних стає все більш важливим завданням. Зокрема, протокол WebSocket, який забезпечує двосторонній зв'язок у реальному часі між клієнтом і сервером, знаходить широке застосування у веб-додатках, таких як чати, онлайн-ігри, фінансові платформи та інші сервіси, що вимагають швидкої обробки даних.

Проте, разом з широким використанням WebSocket, зростає і кількість загроз, пов'язаних з його вразливостями. Серед основних проблем безпеки протоколу можна виділити можливість виконання атак типу "людина посередині" (MitM), перехоплення даних, несанкціонований доступ та інші загрози, які можуть призвести до компрометації конфіденційної інформації.

Особливо актуальним питання забезпечення безпеки WebSocket стає в умовах постійного зростання кіберзлочинності та появи нових методів атак. Наявні механізми захисту, такі як TLS (Transport Layer Security), хоча і забезпечують певний рівень безпеки, проте не завжди достатньо ефективні для протидії сучасним загрозам.

Одним з таких методів є впровадження унікальних ідентифікаторів доступу, що дозволяють ідентифікувати кожного користувача і забезпечити контроль доступу до ресурсів.

Таким чином, розробка вдосконаленої системи захисту для протоколу WebSocket є актуальною та важливою задачею, яка відповідає потребам сучасного суспільства в забезпеченні надійного захисту інформаційних ресурсів.

Мета і задачі дослідженн. Метою даної роботи є розробка вдосконаленої системи захисту протоколу WebSocket, що базується на впровадженні унікальних ідентифікаторів доступу та алгоритму шифрування AES для підвищення рівня безпеки передачі даних.

Задачами дослідження є:

- аналіз існуючих стандартів та протоколів безпеки в інтернеті.

- дослідження особливостей роботи та застосування протоколу websocket.
- виявлення основних проблем безпеки протоколу websocket.
- розробка концепції використання унікальних ідентифікаторів доступу як засобу захисту.
- аналіз сучасних методів захисту інформаційних ресурсів.
- розробка та реалізація вдосконаленої системи захисту для протоколу websocket.
- тестування та оцінка ефективності розробленої системи.

Об’єкт дослідження. Об'єктом дослідження є протокол WebSocket, що використовується для забезпечення двостороннього зв'язку в реальному часі між клієнтом і сервером у веб-додатках.

Предмет дослідження. Предметом дослідження є методи підвищення безпеки протоколу WebSocket, зокрема впровадження унікальних ідентифікаторів доступу та алгоритму шифрування AES.

Новизна роботи. Новизна роботи полягає у розробці вдосконаленої системи захисту для протоколу WebSocket, яка базується на використанні унікальних ідентифікаторів доступу та алгоритму шифрування AES. Такий підхід дозволяє значно підвищити рівень безпеки передачі даних та зменшити ризик успішних атак на інформаційні системи.

Практична цінність. Практична цінність роботи полягає в можливості впровадження розробленої системи захисту у реальні веб-додатки, що використовують протокол WebSocket. Це забезпечить підвищення рівня безпеки передачі даних, захист від атак та збереження конфіденційності інформації. Результати роботи можуть бути корисними для розробників програмного забезпечення, що працюють у сфері веб-технологій, а також для фахівців з інформаційної безпеки.

1 ТЕОРЕТИЧНІ ЗАСАДИ ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ПРОТОКОЛУ WEB SOCKET

У першому розділі магістерської кваліфікаційної роботи буде проведено докладний аналіз стандартів та протоколів безпеки в Інтернеті. Цей аналіз допоможе зрозуміти основні засади захисту в мережі та визначити ті принципи, які можна застосувати для забезпечення безпеки протоколу Web Socket. Крім того, будуть розглянуті основні проблеми безпеки, які можуть виникнути при використанні цього протоколу, щоб виділити напрямки подальших досліджень та покращень.

Результати аналізу будуть узагальнені в висновках, і буде поставлена задача для подальших досліджень.

1.1 Стандарти та протоколи безпеки в Інтернеті

Інтернет – це потужний інструмент, який використовується для спілкування, досліджень, торгівлі та багато чого іншого. Однак він також може бути небезпечним місцем, де хакери та інші зловмисні користувачі можуть вкрасти ваші особисті дані, заразити ваш пристрій шкідливим програмним забезпеченням або навіть взяти під контроль ваш комп'ютер.

Існують різні стандарти безпеки, які допомагають захистити вас в Інтернеті. Ці стандарти розроблені для того, щоб гарантувати, що ваші дані залишаються конфіденційними, цілими та доступними [1].

Шифрування є ключовим компонентом забезпечення безпеки в Інтернеті. Воно перетворює дані в нечитабе́льний формат, який може бути розшифрований лише за допомогою відповідного ключа. Це робить дані нечитабельними для перехоплювачів, що значно знижує ризик їх компрометації.

SL (Secure Sockets Layer) та його наступник TLS (Transport Layer Security) є криптографічними протоколами, які забезпечують захищене з'єднання між клієнтом і сервером через шифрування даних. Ці протоколи широко використовуються для захисту комунікації в мережі, такої як веб-сайти, електронна

пошта, обмін файлами тощо. Давайте розглянемо детальніше, як працюють SSL/TLS:

– Handshake Protocol: Процес встановлення з'єднання SSL/TLS включає у себе обмін криптографічних параметрів між клієнтом і сервером. Цей процес відомий як handshake protocol і має кілька етапів;

ClientHello: Клієнт починає з'єднання, надсилаючи серверу список підтримуваних криптографічних алгоритмів.

ServerHello: Сервер вибирає криптографічні параметри зі списку, що підтримується клієнтом, та надсилає їх клієнту, включаючи цифровий сертифікат.

Certificate Verification: Клієнт перевіряє цифровий сертифікат, щоб забезпечити, що сервер є дійсним і довіреним [2].

Key Exchange: Клієнт та сервер обмінюються ключами для шифрування та розшифрування даних.

Cipher Suite Confirmation: Клієнт і сервер підтверджують вибір криптографічного алгоритму і домовляються про те, як будуть шифруватися дані.

– Data Transfer: Після успішного завершення handshake protocol дані, які передаються між клієнтом і сервером, будуть зашифровані та розшифровані за допомогою обмінених ключів. Це забезпечує конфіденційність та цілісність даних під час їх передачі по мережі;

– Encryption Algorithms: SSL/TLS використовують різні криптографічні алгоритми для шифрування даних, такі як асиметричне шифрування для обміну ключами і симетричне шифрування для самої передачі даних. Деякі з алгоритмів, які можуть бути використані, включають RSA, AES, HMAC тощо;

– Certificate Authorities (CAs): Центри сертифікації використовуються для видачі цифрових сертифікатів, які підтверджують ідентичність веб-сайтів. Це допомагає клієнтам перевірити, що вони з'єднуються із справжнім сервером, а не з підробленим.

SSL/TLS відіграє ключову роль у забезпеченні безпеки комунікації в мережі Інтернет шляхом шифрування даних, що передаються між клієнтом і сервером. Це

дозволяє уникнути перехоплення і читання конфіденційної інформації третіми сторонами.

HTTP (Hypertext Transfer Protocol) є протоколом передачі даних, який використовується для здійснення комунікації між веб-браузерами і веб-серверами в мережі Інтернет. В основному, він використовується для відправлення запитів веб-сторінок і отримання відповідей на них.

Протокол HTTP передає дані у відкритому текстовому форматі, що означає, що будь-яка інформація, яка передається через мережу за допомогою HTTP, може бути зловживана чи переглянута третіми сторонами. Це відкриває можливість для атак на конфіденційні дані, такі як паролі, кредитні картки та особиста інформація.

Щоб забезпечити безпеку під час передачі даних через мережу Інтернет, був розроблений протокол HTTPS (HTTP Secure). HTTPS є захищеною версією протоколу HTTP, яка використовує шифрування для захисту конфіденційності даних. В основі HTTPS лежать два протоколи шифрування: SSL (Secure Sockets Layer) та його спадкоємець, TLS (Transport Layer Security) [2].

Коли веб-клієнт (наприклад, веб-браузер) встановлює з'єднання з веб-сервером через HTTPS, спочатку відбувається процес SSL/TLS handshake. Під час цього процесу відбувається наступне:

– Client Hello: Веб-клієнт починає з'єднання, відправляючи веб-серверу запит на з'єднання із списком криптографічних алгоритмів, які він підтримує;

– Server Hello: Веб-сервер відповідає вибором криптографічного алгоритму та відправляє свій цифровий сертифікат клієнту;

– Certificate Validation: Клієнт перевіряє цифровий сертифікат, щоб підтвердити, що веб-сервер є дійсним і відомим;

– Key Exchange: Клієнт та сервер обмінюються ключами для шифрування та розшифрування даних;

– Cipher Suite Confirmation: Клієнт та сервер підтверджують вибір криптографічного алгоритму і домовляються про те, як будуть шифруватися дані.

Після успішного встановлення з'єднання через SSL/TLS, всі дані, що передаються між клієнтом і сервером, будуть зашифровані. Це дозволяє уникнути перехоплення і читання конфіденційної інформації третіми сторонами під час її передачі через мережу Інтернет. Таким чином, HTTPS є важливим засобом забезпечення безпеки для веб-сайтів та користувачів [3].

IPsec (Internet Protocol Security) – це набір протоколів безпеки, які призначені для захисту даних, що передаються через мережі IP (Internet Protocol). В основному, IPsec використовується для забезпечення конфіденційності, цілісності і аутентифікації даних, які пересилаються між вузлами мережі.

Основні функції IPsec включають:

– конфіденційність (Confidentiality): IPsec забезпечує захист конфіденційності даних, шляхом їх шифрування перед відправленням через мережу. Це перешкоджає прослуховуванню чи читанню даних третіми сторонами;

– цілісність (Integrity): IPsec дозволяє перевіряти цілісність даних, що пересилаються, шляхом включення контрольних сум або підписів до кожного пакету даних. Це дозволяє виявляти будь-які модифікації чи зміни даних під час їх передачі;

– аутентифікація (Authentication): IPsec дозволяє вузлам мережі аутентифікувати один одного для переконання в тому, що вони спілкуються з дійсними та довіреними вузлами мережі. Це забезпечує відсів несанкціонованих спроб доступу до мережевих ресурсів.

IPsec складається з ряду протоколів, які виконують різні функції для забезпечення безпеки мережі. Основні протоколи IPsec включають:

– Authentication Header (AH): Протокол AH використовується для забезпечення аутентифікації і цілісності даних, не шифруючи їх;

– Encapsulating Security Payload (ESP): Протокол ESP використовується для шифрування та/або аутентифікації даних, що передаються через мережу;

– Internet Key Exchange (IKE): IKE використовується для обміну ключами і встановлення безпечного з'єднання між вузлами IPsec.

IPsec може бути використаний для захисту комунікації між вузлами в мережах VPN (Virtual Private Network), а також для захисту комунікації між окремими вузлами в мережі Інтернет або в корпоративних мережах. Враховуючи розмаїття загроз в мережах, IPsec є важливим інструментом для забезпечення безпеки даних у всесвітній мережі.

DNSSEC (Domain Name System Security Extensions) – це розширення системи доменних імен (DNS), яке додає безпеку до процесу перекладу доменних імен в IP-адреси шляхом підписування DNS-записів цифровими підписами. Основна мета DNSSEC полягає в тому, щоб забезпечити ідентифікацію та перевірку цілісності даних DNS, що передаються між клієнтами та серверами DNS [4].

Основні компоненти та принципи роботи DNSSEC включають наступне:

– цифрові підписи: DNSSEC використовує цифрові підписи для підписування DNS-записів. Ці підписи дозволяють перевірити автентичність і цілісність даних DNS, що надходять від DNS-серверів;

– ключі цифрового підпису: DNSSEC використовує два основних типи ключів: ключі підпису (private keys) і ключі перевірки цифрових підписів (public keys). Ці ключі використовуються для підписування і перевірки DNS-записів відповідно;

– зональна ієрархія: DNSSEC працює на основі ієрархічної структури доменних імен, де кожен домен може підписувати свої DNS-записи. Кореневий DNS-сервер має особливе значення в цій ієрархії;

– цифрові ключі кореневого DNS: Для початку довірчого ланцюга в DNSSEC, ключі цифрового підпису кореневого DNS підписуються із високим рівнем довіри. Ці ключі використовуються для перевірки підписів для DNS-записів кореневого домену та встановлення довіри до внутрішніх доменів;

– RRSIG (Resource Record Signature): Це тип DNS-запису, який використовується для збереження цифрових підписів для інших DNS-записів в зоні;

– DNSKEY: Цей тип DNS-запису містить публічні ключі для перевірки цифрових підписів, що використовуються в даній зоні.

Під час перегляду веб-сторінок або взаємодії з мережею, DNSSEC дозволяє клієнтам перевірити автентичність і цілісність DNS-записів, які отримують від серверів DNS. Це допомагає уникнути DNS-атак, таких як DNS-отруєння або DNS-перехоплення, і забезпечує більш високий рівень безпеки під час навігації в Інтернеті. Використання DNSSEC сприяє покращенню безпеки мережі та зменшенню ризиків втрати конфіденційності та викрадення даних [5].

OAuth – це відкритий стандарт авторизації, який дозволяє веб-додаткам отримувати доступ до ресурсів користувача на сторонніх веб-сайтах без передачі їх паролів. Основна ідея полягає в тому, щоб дозволити веб-додаткам використовувати ресурси користувача (наприклад, дані профілю, фотографії, контакти тощо), не потребуючи від нього введення свого пароля на сторонньому веб-сайті. Замість цього, OAuth надає спеціальний механізм авторизації, який дозволяє користувачам контролювати доступ до своїх ресурсів.

Для реалізації авторизації за допомогою OAuth, зазвичай використовується процес, що включає наступні етапи:

– реєстрація додатка: Розробник веб-додатка реєструє свій додаток у системі OAuth-провайдера (наприклад, Google, Facebook, Twitter тощо) і отримує унікальні ідентифікатори (Client ID) та секретні ключі (Client Secret), які використовуються для авторизації додатка;

– перенаправлення користувача: При необхідності доступу до ресурсів користувача, веб-додаток перенаправляє його на сторінку авторизації OAuth-провайдера;

– авторизація користувача: Користувач увійшовши на сторінку авторизації, надає дозвіл веб-додатку на доступ до своїх ресурсів, введенням своїх облікових даних на сторінці OAuth-провайдера;

– видача токену доступу: Якщо авторизація користувача пройшла успішно, OAuth-провайдер видає веб-додатку спеціальний токен доступу (Access Token), який використовується для отримання доступу до ресурсів користувача;

– доступ до ресурсів: Веб-додаток може використовувати токен доступу для отримання доступу до ресурсів користувача на сторонньому веб-сайті.

Одним з основних переваг OAuth є те, що він дозволяє користувачам контролювати доступ до своїх ресурсів без передачі свого пароля стороннім сервісам. Крім того, OAuth є стандартом, що підтримується багатьма провайдерами, що дозволяє використовувати один і той же механізм авторизації для доступу до різних сервісів [6].

Проте, важливо зауважити, що при неправильній реалізації може виникнути ризик безпеки, зокрема, зловмисники можуть отримати доступ до ресурсів користувача через скомпрометовані токени доступу. Тому важливо ретельно розробляти та впроваджувати механізми авторизації на основі стандарту OAuth.

OpenID Connect – це протокол автентифікації, який базується на технологіях OAuth 2.0 та JSON Web Tokens (JWT). Він дозволяє користувачам використовувати один і той же обліковий запис для доступу до різних веб-сайтів, що дозволяє спростити процес автентифікації та забезпечити вищий рівень безпеки та зручності.

Розглянемо детальніше, як працює OpenID Connect:

– авторизація за допомогою OAuth 2.0: OpenID Connect базується на протоколі OAuth 2.0, який використовується для отримання доступу до ресурсів користувача. Під час авторизації OpenID Connect використовується стандартний потік авторизації OAuth 2.0, щоб отримати токен доступу;

– Identity Token: Після успішної авторизації, OpenID Connect повертає identity token (ідентифікаційний токен), який містить інформацію про користувача, наприклад, його ідентифікатор, ім'я, електронну адресу тощо. Цей токен зашифрований і підписаний, що гарантує його цілісність та автентичність;

– JSON Web Tokens (JWT): Identity token використовує стандарт JSON Web Tokens (JWT) для представлення та обміну даних про користувача. JWT – це

компактний та самостійний спосіб передачі інформації між двома сторонами у форматі JSON;

– Authentication Endpoint: OpenID Connect використовує спеціальний ендпоінт (authentication endpoint), до якого веб-додаток перенаправляє користувача для проведення авторизації. Після введення облікових даних користувача та успішної авторизації, OpenID Connect генерує та повертає identity token;

– Authorization Server: Для реалізації OpenID Connect використовується спеціальний сервер, відомий як Authorization Server. Цей сервер відповідає за авторизацію та видачу токенів доступу, а також за формування identity token з інформацією про користувача;

– складний профіль властивостей: OpenID Connect має широкий спектр властивостей та можливостей, що дозволяє налаштовувати його під конкретні потреби веб-додатків, такі як підтримка множини методів аутентифікації, підтримка одного або декількох джерел ідентифікації, налаштування маршрутизації, розширення для різних використань тощо.

OpenID Connect забезпечує простий та безпечний механізм автентифікації, який дозволяє користувачам використовувати один і той же обліковий запис для доступу до різних веб-сайтів, що полегшує та ускладнює процес авторизації.

S/MIME (Secure/Multipurpose Internet Mail Extensions) – це стандарт для захищеної електронної пошти, який надає механізми шифрування та підпису для забезпечення конфіденційності, цілісності та автентичності електронних листів. Основною метою S/MIME є забезпечення безпеки електронної пошти шляхом застосування криптографічних методів [8].

Основні компоненти та принципи роботи S/MIME включають наступне:

– цифрові сертифікати: S/MIME використовує цифрові сертифікати для ідентифікації та автентифікації відправника електронного листа. Ці сертифікати видаються довіреними центрами сертифікації (Certificate Authorities) та містять публічний ключ відправника, який використовується для шифрування та перевірки підпису;

– шифрування: S/MIME дозволяє шифрувати текст електронного листа з використанням публічного ключа отримувача, що забезпечує конфіденційність даних під час їх передачі через мережу. Тільки власник відповідного приватного ключа може розшифрувати повідомлення;

– підписування: S/MIME дозволяє підписувати електронні листи з використанням приватного ключа відправника, що забезпечує цілісність та автентичність повідомлення. Отримувач може перевірити підпис за допомогою публічного ключа відправника, який міститься у відправленому разом з повідомленням цифровому сертифікаті;

– цифрові підписи і шифрування S/MIME вкладення: S/MIME може також застосовуватися до вкладень електронних листів. Це дозволяє підписувати та шифрувати файлові вкладення, такі як документи чи фотографії, що додаються до повідомлення;

– підтримка стандартів: S/MIME підтримується багатьма сучасними поштовими клієнтами та серверами електронної пошти, що робить його широко застосовуваним стандартом для захищеної електронної комунікації.

S/MIME дозволяє створювати захищені електронні листи, які забезпечують конфіденційність, цілісність та автентичність даних. Використання S/MIME є важливим засобом для забезпечення безпеки електронної пошти в організаціях та особистому користуванні.

PGP (Pretty Good Privacy) та GPG (GNU Privacy Guard) – це програмне забезпечення для шифрування, підпису та захисту електронної пошти та файлів. Обидва вони базуються на відкритому стандарті OpenPGP, який використовує асиметричну криптографію для забезпечення конфіденційності, цілісності та автентичності даних [9].

Давайте розглянемо детальніше, як працюють PGP та GPG:

– генерація ключів: Перший крок використання PGP або GPG – це генерація пари ключів: приватного (private key) та публічного (public key). Приватний ключ

використовується для розшифрування повідомлень та підпису даних, тоді як публічний ключ використовується для шифрування даних та перевірки підпису;

– шифрування: Користувачі можуть використовувати публічний ключ отримувача для шифрування повідомлення або файлу перед відправленням. Тільки власник відповідного приватного ключа може розшифрувати та прочитати повідомлення;

– підписування: Користувачі можуть також використовувати свій приватний ключ для підписування повідомлень або файлів. Отримувач може перевірити підпис за допомогою публічного ключа відправника, щоб підтвердити автентичність повідомлення та цілісність даних;

– використання у електронній пошті: PGP та GPG дозволяють інтегрувати шифрування та підписування в електронні поштові клієнти, що дозволяє користувачам захищати конфіденційні дані та забезпечувати автентичність повідомлень у електронній пошті;

– керування ключами: PGP та GPG дозволяють користувачам керувати своїми ключами, включаючи їх генерацію, експорт, імпорт, видалення та інше. Крім того, можна створювати та використовувати ключі для різних цілей, таких як шифрування, підписування, аутентифікація тощо;

– компатибельність: PGP та GPG є сумісними між собою та з іншими реалізаціями стандарту OpenPGP, що дозволяє взаємодіяти з іншими користувачами та програмами, які підтримують цей стандарт.

PGP та GPG є потужними інструментами для захисту конфіденційності, цілісності та автентичності даних у віртуальному просторі. Вони широко використовуються для захисту електронної пошти, файлів та комунікації в Інтернеті.

Стандарти та протоколи безпеки в Інтернеті, такі як HTTPS, SSL/TLS, IPsec, DNSSEC, OAuth, OpenID Connect, S/MIME та PGP/GPG, забезпечують високий рівень захисту даних, конфіденційності, цілісності та автентичності під час передачі і обробки інформації в мережі. Вони є важливими компонентами для

забезпечення безпеки та приватності у віртуальному середовищі, сприяючи надійній комунікації та обміну даними через Інтернет.

1.2 Особливості роботи та застосування протоколу Web Socket

Все більше і більше веб-додатків сьогодні вимагають реального часу взаємодії між користувачем і сервером. Це означає, що клієнти очікують миттєвої відповіді на свої дії без необхідності перезавантаження сторінки чи виконання додаткових запитів. Протокол Web Socket (WebSocket) відповідає на ці потреби, надаючи зручний і ефективний спосіб створення активного та інтерактивного зв'язку між клієнтом і сервером [10].

Зрозуміння та використання протоколу Web Socket відкриває безліч можливостей для розробки інтерактивних та швидких веб-додатків, які забезпечують відмінний користувацький досвід.

Протокол Web Socket (WebSocket) – це технологія, яка надає двостороннє, постійне з'єднання між веб-браузером та сервером, що дозволяє взаємодію та обмін даними в реальному часі. Він є значним покращенням порівняно з традиційними HTTP-запитами, які є однонаправленими і не підтримують активне спілкування між клієнтом і сервером.

Особливості роботи та застосування протоколу Web Socket включають:

– двостороннє з'єднання;

Двостороннє з'єднання – це ключова особливість протоколу Web Socket, яка дозволяє обміну даними між клієнтом і сервером в обидва напрямки. Ось детальніше про цю особливість:

Бідирекціональний потік даних: У порівнянні з традиційними протоколами, такими як HTTP, де клієнт ініціює запит до сервера, а сервер відповідає на цей запит, Web Socket дозволяє ініціювати передачу даних у будь-який момент часу як клієнту, так і серверу. Це означає, що обидва боки з'єднання можуть надсилати повідомлення один одному без очікування запитів.

Активне спілкування: Замість того, щоб чекати на запити від клієнтів, серверів можуть активно надсилати повідомлення клієнтам, які підписалися на певні події або канали. Це відкриває можливості для реалізації реального часу оновлення та сповіщень у веб-додатках.

Динамічна взаємодія: Двостороннє з'єднання дозволяє створювати динамічні та інтерактивні веб-додатки, де клієнт і сервер можуть обмінюватися даними, сприяючи більш ефективній взаємодії з користувачем та реалізації складних функціональних можливостей [12].

Підтримка різних типів даних: Web Socket дозволяє передавати різні типи даних між клієнтом і сервером, включаючи текстові дані, бінарні файли, JSON-об'єкти тощо. Це дає можливість створювати багатофункціональні та масштабовані веб-додатки з різноманітними вимогами до обміну даними.

Постійне з'єднання: Web Socket підтримує постійне з'єднання між клієнтом і сервером, що дозволяє надсилати повідомлення в обидва напрямки без необхідності встановлення нового з'єднання для кожного запиту. Це дозволяє знизити накладні витрати на з'єднання та забезпечити ефективну взаємодію між сторонами [13].

Двостороннє з'єднання в протоколі Web Socket відкриває нові можливості для реалізації інтерактивних веб-додатків з реальним часом оновлення та активним спілкуванням між клієнтом і сервером. Він є важливим компонентом для створення сучасних веб-додатків, які відповідають високим вимогам щодо продуктивності та функціональності.

– реальний час;

Реальний час – це ключова характеристика протоколу Web Socket, яка дозволяє передавати дані між клієнтом і сервером без зайвих затримок, забезпечуючи швидку і миттєву відповідь. Давайте розглянемо детальніше, як працює ця функціональність та які можливості вона відкриває:

– низька затримка: Однією з основних переваг Web Socket є здатність передавати дані без зайвих затримок. Порівняно з традиційними методами, такими

як HTTP, які вимагають перезавантаження сторінки або додаткових запитів для оновлення вмісту, Web Socket дозволяє миттєво оновлювати вміст в реальному часі;

– онлайн-чати: Web Socket ідеально підходить для реалізації онлайн-чатів, де користувачі можуть обмінюватися повідомленнями без затримок. Він дозволяє надсилати та отримувати повідомлення миттєво, що створює враження реального часу взаємодії;

– онлайн-ігри: Для онлайн-ігор, де важливо миттєво реагувати на дії користувачів та синхронізувати гру між гравцями, Web Socket є відмінним вибором. Він дозволяє передавати дії гравців у реальному часі без помітних затримок;

– потокове відео: Використання Web Socket для потокового відео дозволяє отримувати та відтворювати відео без буферизації або затримок, забезпечуючи плавний і безперервний перегляд;

– оновлення в реальному часі: Застосування Web Socket для оновлення вмісту сторінок або додатків у реальному часі дозволяє надсилати зміни користувачам миттєво, без необхідності перезавантаження сторінки.

Реальний час в протоколі Web Socket відкриває широкі можливості для реалізації інтерактивних та швидких веб-додатків, які забезпечують користувачам неперевершений досвід взаємодії та високу продуктивність. Ця функціональність робить Web Socket ідеальним інструментом для реалізації функціональності, що потребує миттєвої відповіді та активного спілкування з користувачем [13].

– простота використання;

Простота використання – це одна з ключових переваг протоколу Web Socket, яка робить його доступним та привабливим для веб-розробників. Давайте розглянемо детальніше, що робить його таким простим у використанні:

Стандартні API: Web Socket використовує стандартні API, які легко розуміти та використовувати для реалізації веб-додатків. Це дозволяє розробникам швидко оволодіти технологією та швидко почати розробку.

Бібліотеки для різних мов програмування: Існують багато бібліотек та фреймворків для різних мов програмування, які спрощують роботу з Web Socket. Ці бібліотеки надають високорівневі інтерфейси для роботи з Web Socket, що дозволяє розробникам швидко інтегрувати його в свої додатки.

Доступність документації: Існують великі обсяги документації, туторіалів та прикладів коду, які допомагають розробникам оволодіти Web Socket. Це робить процес вивчення та розуміння технології досить простим та доступним.

Сумісність з браузерами і серверами: Web Socket підтримується більшістю сучасних веб-браузерів і веб-серверів, що робить його універсальним і легким у використанні без необхідності виконання додаткових налаштувань або встановлення додаткового програмного забезпечення.

Масштабованість та гнучкість: Web Socket може бути легко масштабований та розширюваний для відповіді на різноманітні потреби веб-додатків. Розробники можуть легко змінювати та доповнювати функціональність, не порушуючи стабільність додатку [15].

Усі ці фактори роблять Web Socket привабливим вибором для веб-розробників, які шукають простий, ефективний та надійний спосіб забезпечити реальний час взаємодії між клієнтом і сервером у своїх веб-додатках. Ця простота використання допомагає розробникам швидше створювати високоякісні веб-додатки з великою функціональністю.

– ефективність;

Ефективність – це важлива характеристика протоколу Web Socket, яка робить його привабливим вибором для реалізації веб-додатків з високою продуктивністю та швидкістю. Давайте розглянемо детальніше, як саме Web Socket забезпечує ефективну передачу даних:

Мінімізація накладних витрат на передачу даних: У порівнянні з традиційним протоколом HTTP, який вимагає відкриття нового з'єднання для кожного запиту та відповіді, Web Socket дозволяє підтримувати постійне з'єднання між клієнтом і сервером. Це дозволяє уникнути накладних витрат, пов'язаних з встановленням

нових з'єднань, та знизити кількість передачі метаданих, що зменшує обсяг даних, що передаються.

Постійне з'єднання: Web Socket підтримує постійне з'єднання між клієнтом і сервером протягом тривалості сесії, що дозволяє надсилати дані в обидва напрямки без необхідності встановлення нових з'єднань для кожного запиту. Це значно зменшує час, необхідний для встановлення та розриву з'єднання, та дозволяє швидко реагувати на зміни в стані додатку.

Мінімізація затримок: Завдяки постійному з'єднанню та зменшенню кількості передачі метаданих, Web Socket дозволяє мінімізувати затримки в передачі даних між клієнтом і сервером. Це особливо важливо для додатків, які вимагають миттєвої відповіді та реального часу взаємодії.

Збільшення продуктивності: Зниження накладних витрат на передачу даних та затримок сприяє підвищенню продуктивності додатків, які використовують Web Socket. Це дозволяє забезпечити кращий користувацький досвід та зробити додаток більш конкурентоспроможним.

В цілому, ефективність протоколу Web Socket полягає у здатності забезпечити швидку та ефективну передачу даних без зайвих затримок та накладних витрат, що робить його ідеальним вибором для реалізації веб-додатків, які вимагають високої продуктивності та реального часу взаємодії [18].

– підтримка крос-платформи;

Підтримка крос-платформи – це одна з ключових переваг протоколу Web Socket, яка забезпечує його універсальність та доступність для використання на різних платформах. Давайте розглянемо детальніше, як саме Web Socket підтримується на різних платформах:

Сучасні веб-браузери: Більшість сучасних веб-браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Opera тощо, підтримують протокол Web Socket без будь-яких додаткових налаштувань. Це дозволяє веб-розробникам використовувати Web Socket для реалізації різних функціональностей на веб-сторінках, які будуть працювати на більшості платформ.

Веб-сервери: Багато веб-серверів, таких як Apache, Nginx, Microsoft IIS, Node.js тощо, підтримують Web Socket, що дозволяє використовувати його для забезпечення двостороннього зв'язку між клієнтами і сервером на різних платформах. Це робить Web Socket доступним для використання у веб-додатках, які працюють на різних серверних оточеннях.

Мобільні платформи: Web Socket також підтримується на мобільних платформах, таких як iOS та Android. Це означає, що розробники можуть використовувати його для створення мобільних додатків, які взаємодіють з веб-серверами в реальному часі без зайвих зусиль.

Крос-платформеність бібліотек: Існують багато крос-платформених бібліотек та фреймворків для роботи з Web Socket, які дозволяють розробникам використовувати його на різних платформах програмування. Наприклад, бібліотеки на JavaScript, Python, Java, C#, і т.д., роблять Web Socket доступним для використання в різних мовах програмування та середовищах розробки [19].

Стандартизованість: Web Socket є стандартом, прийнятим у Інтернеті, що дозволяє забезпечити його сумісність з різними платформами та середовищами. Це робить його універсальним і доступним для використання на будь-якій платформі без обмежень.

Загалом, підтримка крос-платформи робить Web Socket універсальним і доступним інструментом для реалізації різноманітних веб-додатків на різних платформах та пристроях. Це дозволяє розробникам створювати високоякісні інтерактивні додатки, які працюють на будь-яких пристроях і операційних системах.

– безпека;

Безпека – це важлива складова протоколу Web Socket, і вона забезпечується за рахунок можливості використання захищеного з'єднання, відомого як WebSocket Secure (WSS). Давайте розглянемо детальніше, як саме забезпечується безпека в Web Socket:

WebSocket Secure (WSS): WebSocket Secure (WSS) – це захищена версія протоколу Web Socket, яка використовує SSL/TLS для шифрування даних між

клієнтом і сервером. Це забезпечує конфіденційність даних та захист від прослуховування, оскільки всі дані, які передаються через з'єднання WSS, шифруються.

SSL/TLS шифрування: Використання SSL/TLS протоколу дозволяє шифрувати дані перед їх передачею по мережі. Це означає, що навіть якщо хтось отримає доступ до переданих даних, вони залишатимуться незрозумілими через шифрування.

Конфіденційність та цілісність даних: Використання захищеного з'єднання через WSS дозволяє забезпечити конфіденційність даних, що передаються між клієнтом і сервером. Крім того, шифрування даних також гарантує їх цілісність, тобто вони не можуть бути змінені або підроблені під час передачі.

Захист від атак: Використання захищеного з'єднання допомагає захистити протокол від різних видів атак, таких як перехоплення даних, впровадження коду, атаки типу "людина посередника" та інших.

Стандартизованість: Використання SSL/TLS шифрування є стандартною практикою для забезпечення безпеки в Інтернеті. Це означає, що Web Socket може легко інтегруватися в існуючу інфраструктуру безпеки та користуватися всіма перевагами стандартів шифрування.

В цілому, за допомогою захищеного з'єднання WSS та використання SSL/TLS шифрування, Web Socket забезпечує надійний та безпечний обмін даними між клієнтом і сервером, що робить його популярним вибором для створення безпечних і захищених веб-додатків.

– розширеність.

Розширеність – це важлива характеристика протоколу Web Socket, яка дозволяє розробникам додавати нові функціональності та вдосконалювати протокол з часом. Давайте розглянемо детальніше, як саме це досягається:

Розширення протоколу: Web Socket підтримує можливість розширення самого протоколу шляхом визначення та впровадження нових команд та функцій. Це дозволяє розробникам додавати нові опції та покращення до протоколу без необхідності зміни основного стандарту.

Розширення клієнтських та серверних бібліотек: Велика кількість клієнтських та серверних бібліотек для роботи з Web Socket мають вбудовану можливість розширення. Це дозволяє розробникам створювати власні розширення, які можуть бути використані для різних цілей, таких як оптимізація мережевого трафіку, кешування даних, підтримка специфічних протоколів тощо.

Стандартизованість розширень: Щоб забезпечити сумісність та стандартизованість, розширення протоколу Web Socket можуть бути піддані процесу стандартизації та затвердженню, що дозволяє їх використання у різних реалізаціях протоколу [20].

Розширюваність функціональності: Завдяки можливості додавання нових функцій та опцій через розширення, Web Socket може бути легко адаптований до різних вимог і використовуватися для різних цілей, від створення інтерактивних додатків до розробки складних систем реального часу.

Динамічне оновлення: Розширення можуть бути впроваджені в протокол під час його розвитку, що дозволяє підтримувати актуальність та відповідати змінюючимся потребам і вимогам веб-розробників.

В цілому, розширеність протоколу Web Socket дозволяє розробникам створювати потужні та гнучкі системи комунікації, які можуть ефективно відповідати на різноманітні потреби веб-додатків та надсилати дані у реальному часі.

Враховуючи ростуть вимог до надійності та продуктивності веб-додатків у сферах, що включають, але не обмежуються онлайн-іграми, соціальними мережами та колективною роботою, важливо ретельно розглянути роботу даного протоколу.

Для цього детально аналізується схема (рис. 1.1) функціонування WebSocket з метою визначення його ключових аспектів та механізмів, які забезпечують ефективний обмін даними між сторонами зв'язку.

Рисунок 1.1 – Схема роботи WebSocket

Схема роботи WebSocket:

– установлення з'єднання (Handshake): Клієнт відправляє запит на підключення до сервера через HTTP, включаючи заголовки, що вказують на намір встановити WebSocket-з'єднання;

– відповідь сервера (Handshake): Сервер розпізнає запит на WebSocket-з'єднання, відповідає підтвердженням та переходить до протоколу WebSocket;

– закриття з'єднання: Закриття з'єднання може бути ініційоване будь-якою стороною, і відбувається шляхом надсилання спеціального фрейму закриття;

– обмін даними: Після успішного Handshake клієнт та сервер можуть обмінюватися даними в реальному часі за допомогою WebSocket-протоколу, надсилаючи одночасно текстові та бінарні повідомлення через відкрите TCP-з'єднання.

Ця схема дозволяє створювати ефективне та миттєве двостороннє з'єднання між клієнтом та сервером, що використовується для реалізації різноманітних додатків в реальному часі, таких як онлайн-чати, потокове відео та ігри [21].

Протокол Web Socket – це потужний інструмент для створення веб-додатків, що потребують реального часу взаємодії між клієнтом і сервером. Його особливості включають двостороннє з'єднання для обміну даними, здатність передавати дані в реальному часі без затримок, простоту використання як на клієнтській, так і на серверній стороні, ефективність в порівнянні з іншими методами комунікації, підтримку крос-платформи та можливість захищеного з'єднання за допомогою SSL/TLS. Він також підтримує розширення, що дозволяє розробникам додавати нові функціональності та вдосконалювати протокол з часом. Загалом, Web Socket – це важливий інструмент для реалізації інтерактивних веб-додатків, які вимагають швидкого та ефективного обміну даними між клієнтом і сервером.

1.3 Проблеми безпеки протоколу Web Socket

В інтернеті безпека завжди є важливою темою, особливо з розвитком нових технологій та протоколів. Одним із таких протоколів, який набуває все більшої популярності, є Web Socket. Використання Web Socket дозволяє веб-додаткам здійснювати реальні часи взаємодії між клієнтом і сервером, що стає основою для багатьох інноваційних застосувань, таких як онлайн-ігри, чати та спільна робота над документами. Однак, разом зі зростанням використання Web Socket з'являються і проблеми безпеки, які потрібно враховувати.

Протокол Web Socket, як і будь-який інший інтернет-протокол, не позбавлений проблем безпеки. Деякі з основних проблем безпеки, пов'язаних з Web Socket, включають:

– Cross-Site WebSocket Hijacking (CSWSH);

Cross-Site WebSocket Hijacking (CSWSH) є типом атаки на безпеку, яка використовується для викрадення конфіденційних даних, що передаються через

протокол Web Socket. В цій атакі злоумисник може використовувати вразливості на веб-сайті для впровадження зловмисного JavaScript коду на стороні клієнта. Після цього виконується перехоплення з'єднання Web Socket, яке вже існує між клієнтом і сервером, і використовується для отримання конфіденційних даних.

Основні етапи атаки CSWSH включають наступне:

Виконання зловмисного коду на стороні клієнта: Злоумисник використовує вразливість на веб-сайті для впровадження зловмисного JavaScript коду на стороні клієнта. Це може статися через вразливості в сторінках, вводі користувача або інші способи атаки.

Перехоплення з'єднання Web Socket: Зловмисний JavaScript, що виконується на стороні клієнта, перехоплює існуюче з'єднання Web Socket між клієнтом і сервером. Це може бути здійснене за допомогою функцій, які дозволяють перехоплювати трафік мережі або здійснювати маніпуляції з DOM (Document Object Model) [22].

Викрадення даних: Після успішного перехоплення з'єднання Web Socket, злоумисник може отримати доступ до конфіденційних даних, які передаються через це з'єднання. Це може бути сеансові токени, особиста інформація користувача або будь-яка інша конфіденційна інформація, яка передається через протокол Web Socket.

Щоб запобігти атакам CSWSH, важливо виконувати належну безпеку на рівні клієнта і сервера. Це може включати в себе використання захищених протоколів, які забезпечують шифрування даних, перевірку на достовірність ідентифікаторів, регулярне оновлення програмного забезпечення для усунення вразливостей та виконання валідації всіх даних, що вводяться користувачем. Також важливо вести моніторинг безпеки та вчасно реагувати на будь-які підозрілі дії або події.

– DoS атаки (Denial of Service);

DoS (Denial of Service) атаки є серйозною загрозою для протоколу Web Socket, оскільки можуть призвести до відмови в обслуговуванні для законних користувачів. Ось детальніше про різновиди DoS атак та їх можливі наслідки:

Переповнення буфера: Ця атака полягає в намаганні направити на сервер Web Socket більше даних, ніж може бути оброблено. Якщо сервер не може ефективно управляти обсягом вхідних даних, це може призвести до переповнення буфера і відмови у обслуговуванні [23].

Перевантаження з'єднань: Атаки, спрямовані на перевантаження сервера Web Socket шляхом встановлення великої кількості з'єднань, також можуть спричинити відмову у обслуговуванні. Це може бути досягнуто шляхом автоматизованого створення тисяч або навіть мільйонів з'єднань одночасно.

Витрати ресурсів сервера: Атаки, спрямовані на використання великої кількості ресурсів сервера, таких як процесорний час або оперативна пам'ять, можуть призвести до відмови у обслуговуванні. Злоумисники можуть намагатися виконати операції, які споживають велику кількість ресурсів, таким чином перевантажуючи сервер.

Атаки на різноманітні протоколи і ресурси: DoS атаки можуть бути спрямовані не лише на сервер Web Socket, але і на інші протоколи та ресурси, які використовуються для підтримки Web Socket, такі як HTTP-сервери, мережеві комутатори, брандмауери тощо.

Наслідками успішної DoS атаки можуть бути недоступність сервісу для законних користувачів, втрата даних або ресурсів, а також погіршення репутації компанії. Для запобігання DoS атакам важливо вживати заходів захисту на рівні мережі, сервера і застосунку, таких як використання захищених протоколів, обмеження кількості одночасних з'єднань, моніторинг мережевого трафіку і вчасна реакція на підозрілі або аномальні дії.

– вразливості безпеки в реалізації протоколу;

Вразливості в реалізації протоколу Web Socket можуть стати серйозною загрозою для безпеки системи. Деякі з найбільш поширених вразливостей включають наступне:

Переповнення буфера (Buffer Overflow): Ця вразливість виникає, коли злоумисники намагаються ввести більшу кількість даних, ніж передбачено, в буфер пам'яті. Якщо реалізація протоколу Web Socket не вміє правильно обробляти ці

дані, це може призвести до переповнення буфера, витоку пам'яті або навіть виконання зловмисних кодів.

Витік пам'яті (Memory Leak): В разі виникнення витоку пам'яті під час обробки підключень або даних через Web Socket, злоумисник може зловживати цією вразливістю, щоб вивільнити ресурси сервера, збільшивши витрати ресурсів.

Недостатні перевірки автентичності (Insufficient Authentication): Якщо реалізація протоколу Web Socket не виконує належну перевірку автентичності користувачів чи даних, це може призвести до можливості виконання атак типу MITM (Man-in-the-Middle), впровадження зловмисних даних або навіть підробки ідентифікаторів.

Використання застарілих алгоритмів шифрування (Use of Deprecated Encryption Algorithms): Якщо реалізація протоколу використовує застарілі або слабкі алгоритми шифрування, це може зробити з'єднання вразливим до атак на шифрування, таких як атаки типу "чоловік посередині" (Man-in-the-Middle) або підібрання ключів [25].

Недостатній контроль введення (Input Validation): Відсутність належної перевірки введених даних перед їх обробкою може призвести до виконання SQL-ін'єкцій, XSS або інших типів атак, які можуть впливати на безпеку системи.

Для запобігання цим вразливостям важливо вжити належних заходів безпеки на кожному рівні розробки, включаючи належну перевірку та обробку введення, використання захищених алгоритмів шифрування, моніторинг та аудит безпеки системи, а також регулярне оновлення програмного забезпечення для усунення вразливостей.

– недостатній контроль доступу;

Недостатній контроль доступу може стати серйозною проблемою для безпеки системи, особливо в контексті протоколу Web Socket. Ось детальніше про можливі наслідки та причини цієї вразливості:

Витік конфіденційної інформації: Якщо доступ до Web Socket не належним чином обмежений, несанкціоновані користувачі можуть отримати доступ до

конфіденційної інформації, що передається через цей канал. Це може стати причиною витоку особистої, фінансової або комерційної інформації.

Зловживання правами: Несанкціоновані користувачі, які отримають доступ до Web Socket, можуть зловживати цим доступом, виконуючи недозволені дії, які можуть призвести до порушення безпеки або втрати даних. Це може включати в себе впровадження зловмисного коду, зміну налаштувань системи або втручання у роботу програм [26].

Недостатній контроль автентифікації і авторизації: Якщо механізми автентифікації і авторизації не виконуються належним чином, це може призвести до недостатнього контролю доступу до Web Socket. Наприклад, недостатня перевірка ідентифікаційних даних користувачів або надання недостатніх прав доступу може відкрити двері для несанкціонованого доступу.

Ризик втрати довіри: Недостатній контроль доступу може призвести до порушення довіри користувачів до системи. Якщо користувачі відчують, що їхні дані не належним чином захищені, це може призвести до втрати довіри та негативного впливу на репутацію компанії.

Для запобігання проблемам, пов'язаним з недостатнім контролем доступу до Web Socket, важливо використовувати належні механізми автентифікації і авторизації, обмежувати доступ до системи лише санкціонованим користувачам, встановлювати правильні правила доступу та регулярно аудитувати систему для виявлення можливих вразливостей. Також важливо вести моніторинг активності користувачів і негайно реагувати на будь-які підозрілі дії.

– крос-сайтові скриптові атаки (XSS).

Крос-сайтові скриптові атаки (XSS) є однією з найбільш поширених загроз безпеці веб-додатків, включаючи ті, які використовують протокол Web Socket. XSS дозволяє злоумисникам впроваджувати та виконувати зловмисний JavaScript код на веб-сторінках, які відкриваються в браузері користувача. Ось детальніше про XSS та його вплив на Web Socket:

Впровадження зловмисного JavaScript коду: Злоумисники використовують XSS для впровадження зловмисного JavaScript коду в веб-сторінки, які

відкриваються в браузері. Цей код може бути використаний для перехоплення або модифікації даних, які передаються через Web Socket, а також для виконання інших зловмисних дій, таких як крадіжка сесійних файлів або перенаправлення на фішингові сайти.

Можливі наслідки для безпеки даних: XSS може призвести до витоку конфіденційної інформації, так як злоумисники можуть отримувати доступ до даних, що передаються через Web Socket, включаючи особисті дані, фінансову інформацію чи важливі корпоративні дані.

Можливість для зловживання функціоналу Web Socket: Злоумисники можуть використовувати XSS для зловживання функціоналу Web Socket, включаючи відправку неправдивих повідомлень, впровадження шкідливих даних або спричинення відмови у обслуговуванні (DoS) за допомогою великого обсягу фальшивих запитів [27].

Підробка ідентифікаторів сесій: XSS може бути використаний для підробки ідентифікаторів сесій або інших авторизаційних даних, що передаються через Web Socket. Це може дозволити злоумисникам отримати несанкціонований доступ до акаунтів користувачів або конфіденційних ресурсів.

Для запобігання XSS атакам на Web Socket важливо використовувати належні механізми фільтрації та екранування введених даних, а також належно налаштувати політики безпеки браузера для запобігання виконання зловмисного JavaScript коду. Також рекомендується регулярно аудитувати веб-додатки на наявність потенційних вразливостей та вживати заходів для їх виправлення.

Протокол Web Socket, хоча і забезпечує швидке та ефективне з'єднання між клієнтом і сервером, також стикається з рядом серйозних проблем безпеки. Недоліки у контролі доступу, вразливості до атак типу XSS та DoS можуть призвести до витоку конфіденційної інформації, порушень безпеки даних та недоступності сервісу для законних користувачів. Важливо вжити належних заходів захисту, включаючи фільтрацію даних, налаштування правильних політик безпеки та регулярне аудитування системи для виявлення та виправлення вразливостей [28].

1.4 Унікальні ідентифікатори доступу як засіб захисту.

У вимірах сучасного інтернет-світу безпека відіграє ключову роль. З розвитком технологій з'являються нові загрози, що ставить під сумнів безпеку користувачів та даних. Відповідно, постійне вдосконалення методів захисту є критичним завданням для розробників та адміністраторів веб-додатків та сервісів. У цьому контексті унікальні ідентифікатори доступу виступають одним із ключових інструментів забезпечення безпеки. Вони грають рішучу роль у визначенні і перевірці ідентичності користувачів, контролі доступу до ресурсів та захисті даних від небажаних атак.

Унікальні ідентифікатори доступу (UID) – це унікальні значення, які призначаються об'єктам чи користувачам в системі з метою ідентифікації та автентифікації. Вони грають ключову роль у забезпеченні безпеки та контролі доступу до ресурсів у веб-додатках та сервісах. Давайте розглянемо основні аспекти унікальних ідентифікаторів доступу:

– унікальність;

Забезпечення унікальності унікальних ідентифікаторів доступу (UID) є критично важливим аспектом для їхнього ефективного використання в системі. Ось детальніше про унікальність UID і її значення:

Гарантія унікальності: Унікальний ідентифікатор повинен бути унікальним у межах всієї системи. Це означає, що жоден інший об'єкт чи користувач не повинен мати такий же ідентифікатор. Це гарантує, що кожен об'єкт чи користувач може бути однозначно ідентифікований.

Унікальність протягом тривалого часу: Унікальні ідентифікатори мають залишатися унікальними протягом тривалого часу. Це означає, що після призначення ідентифікатора він не повинен змінюватися або бути призначений іншому об'єкту чи користувачу.

Ідентифікація об'єктів та користувачів: Забезпечення унікальності UID дозволяє однозначно ідентифікувати кожен об'єкт чи користувача в системі. Це

дозволяє зручно працювати з великою кількістю даних та користувачів, не переплутуючи їх.

Унікальність у межах контексту: Унікальність UID має бути забезпечена не лише у межах окремої системи, але і у межах конкретного контексту чи простору імен. Це дозволяє уникнути конфліктів між ідентифікаторами у різних частинах системи.

Захист від конфліктів і помилок: Унікальні ідентифікатори запобігають конфліктам та помилкам, що можуть виникнути при роботі з даними і користувачами. Вони дозволяють системі ефективно управляти ресурсами та надавати доступ до них.

Унікальність є фундаментальною характеристикою унікальних ідентифікаторів доступу, яка забезпечує надійну ідентифікацію об'єктів та користувачів у системі. Це гарантує правильне функціонування системи та захист від можливих помилок та конфліктів.

– автентифікація;

Автентифікація – це процес перевірки ідентичності користувача або об'єкта у системі. Унікальні ідентифікатори доступу (UID) грають важливу роль у цьому процесі, оскільки вони дозволяють системі однозначно ідентифікувати користувачів та об'єкти. Ось детальніше про автентифікацію за допомогою UID:

Перевірка ідентичності: Користувач або об'єкт надає свій унікальний ідентифікатор для перевірки в системі. Це може бути, наприклад, ім'я користувача або спеціальний код.

Порівняння збереженого ідентифікатора: Система порівнює наданий ідентифікатор зі збереженим в базі даних чи іншому захищеному місці ідентифікатором.

Надання доступу: Якщо ідентифікатори співпадають, система надає користувачеві відповідний рівень доступу до ресурсів. Це може бути доступ до певних функцій програми, вмісту чи сервісів.

Авторизація: Після успішної автентифікації система може також провести процедуру авторизації, щоб визначити, які конкретно дії чи ресурси доступні цьому користувачеві [30].

Захист від несанкціонованого доступу: Використання унікальних ідентифікаторів дозволяє системі надійно перевіряти ідентичність користувачів та об'єктів, запобігаючи таким чином несанкціонованому доступу до ресурсів.

Автентифікація за допомогою унікальних ідентифікаторів доступу є важливим етапом у забезпеченні безпеки та контролю доступу у системі. Вона дозволяє системі ідентифікувати користувачів та об'єкти та надавати їм відповідний рівень доступу до ресурсів, що забезпечує безпеку та захищеність системи.

– контроль доступу;

Контроль доступу з використанням унікальних ідентифікаторів доступу (UID) дозволяє системі ефективно регулювати доступ користувачів до різних ресурсів. Ось детальніше про цей процес:

– визначення прав доступу: Кожному користувачеві або об'єкту системи призначаються відповідні унікальні ідентифікатори, які відображають їхні права доступу. Це може бути набір привілеїв або ролей, які визначають, які дії чи ресурси є доступними для конкретного користувача;

– налаштування політики доступу: Системний адміністратор або розробник встановлює правила контролю доступу, визначаючи, які користувачі або групи користувачів мають доступ до різних ресурсів. Це може бути на основі ролей, груп або інших критеріїв;

– обробка запитів на доступ: Коли користувач або об'єкт системи намагається отримати доступ до певного ресурсу, система перевіряє його унікальний ідентифікатор і порівнює його з правилами контролю доступу. Якщо користувач має відповідні права, йому надається доступ;

– заборона доступу: У разі, якщо користувач немає необхідних прав для доступу до ресурсу, йому відмовляється у доступі. Це забезпечує захист конфіденційної інформації та запобігає несанкціонованому доступу;

– журналювання доступу: Деякі системи ведуть журнал доступу, який фіксує всі спроби доступу до ресурсів. Це дозволяє виявляти незвичайну активність або спроби несанкціонованого доступу та вживати заходів для їхнього запобігання.

Контроль доступу за допомогою унікальних ідентифікаторів є ключовим механізмом для забезпечення безпеки та захисту конфіденційності у веб-системах. Він дозволяє точно регулювати доступ користувачів до різних ресурсів відповідно до їхніх ролей та прав.

– безпека;

Безпека, забезпечена використанням унікальних ідентифікаторів доступу (UID), є критично важливою для захисту конфіденційної інформації та забезпечення приватності користувачів. Ось детальніше про це:

– захист конфіденційності: Унікальні ідентифікатори дозволяють уникнути використання особистих даних для ідентифікації користувачів. Це допомагає зберегти конфіденційність особистої інформації, так як UID не пов'язані з особистими даними користувача;

– анонімність користувача: Використання унікальних ідентифікаторів дозволяє зберігати ідентичність користувача в анонімності. Це означає, що інформація про конкретного користувача не пов'язана з його особистими даними, що робить його складнішим для ідентифікації;

– менша вразливість до атак: Використання унікальних ідентифікаторів може зменшити вразливість системи до атак, таких як перехоплення сесій або витік особистої інформації. Оскільки UID не пов'язані з особистими даними, вони менше піддаються зловживанню або недозволеним діям;

– можливість керування доступом: Унікальні ідентифікатори дозволяють системі ефективно керувати доступом користувачів до різних ресурсів, не

розкриваючи їхньої особистої інформації. Це дозволяє точно контролювати, яка інформація доступна кожному користувачеві;

– збереження інформації: Використання унікальних ідентифікаторів спрощує процес збереження і обробки інформації, оскільки вони можуть бути легко управляються та аналізовані без необхідності обробки особистих даних.

У цілому, використання унікальних ідентифікаторів доступу допомагає забезпечити безпеку, захист конфіденційності та анонімність користувачів, зменшуючи при цьому ризик витоку особистої інформації та забезпечуючи ефективний контроль доступу до ресурсів системи.

– масштабованість.

Масштабованість у контексті унікальних ідентифікаторів (UID) означає їхню здатність працювати ефективно та надійно у великих системах з великою кількістю користувачів або об'єктів. Ось детальніше про цю характеристику:

– легкість генерації: Унікальні ідентифікатори можуть бути легко згенеровані за допомогою алгоритмів генерації випадкових чисел або за допомогою спеціальних сервісів для створення UID. Це дозволяє швидко створювати велику кількість UID для великої кількості користувачів чи об'єктів [31].

– швидка перевірка: Унікальні ідентифікатори можуть бути швидко перевірені для визначення їх унікальності в системі. Це важливо у великих системах, де потрібно швидко і ефективно обробляти великий потік запитів на автентифікацію та контроль доступу.

– системи з великою кількістю користувачів: Унікальні ідентифікатори ефективно працюють у великих системах з великою кількістю користувачів або об'єктів, оскільки вони не потребують складних обчислень або обробки для забезпечення їх унікальності.

– гнучкість: Системи, що використовують унікальні ідентифікатори, можуть легко масштабуватись зі зростанням кількості користувачів або об'єктів. Додавання нових UID не потребує значних змін у системі або її архітектурі.

– стійкість до перевантажень: Унікальні ідентифікатори допомагають системам залишатися стійкими до перевантажень, оскільки вони дозволяють рівномірно розподіляти навантаження при обробці запитів на автентифікацію та контроль доступу.

Масштабованість унікальних ідентифікаторів робить їх ефективним і надійним методом забезпечення безпеки великих систем, де важлива швидкість обробки та висока продуктивність. Вони дозволяють системам ефективно управляти автентифікацією та контролем доступу для великої кількості користувачів чи об'єктів, забезпечуючи при цьому надійність та швидкість роботи.

Унікальні ідентифікатори доступу є ефективним і надійним засобом захисту в інформаційних системах. Вони забезпечують безпеку та приватність, зменшуючи ризик витоку конфіденційної інформації, і забезпечують масштабованість та ефективність управління доступом до ресурсів.

1.5 Аналіз сучасних методів захисту інформаційних ресурсів.

В сучасному цифровому світі захист інформаційних ресурсів від кіберзагроз та несанкціонованого доступу стає все більшою проблемою для організацій, компаній та індивідуальних користувачів. Запобігання витоку даних, забезпечення конфіденційності та цілісності інформації, а також забезпечення безпеки мережевих систем є надзвичайно важливими завданнями в умовах постійно зростаючих кіберзагроз.

У цьому контексті необхідно провести аналіз сучасних методів захисту інформаційних ресурсів, щоб визначити їхню ефективність, надійність та відповідність сучасним потребам у сфері кібербезпеки. В рамках даного дослідження будуть розглянуті різні аспекти захисту, включаючи шифрування даних, системи аутентифікації та авторизації, моніторинг та аналіз поведінки, захист від витоку даних, інцидентний реагування та відновлення після кризи та інші.

Цей аналіз допоможе визначити найбільш ефективні та придатні для використання методи захисту, які можуть бути впроваджені для забезпечення безпеки інформаційних ресурсів в сучасному цифровому середовищі.

Аналіз сучасних методів захисту інформаційних ресурсів може бути проведений за допомогою різноманітних критеріїв, які враховують різні аспекти захисту даних та ресурсів. Ось деякі з найважливіших критеріїв для такого аналізу:

– ефективність: Оцінка того, наскільки ефективно методи захисту можуть запобігти несанкціонованому доступу, атакам та витоку даних.

– надійність: Оцінка ступеня надійності методів захисту, зокрема їхньої здатності протистояти різним видам загроз та виявленому тестуванню.

– відповідність: Визначення того, наскільки методи захисту відповідають вимогам внутрішніх та зовнішніх нормативних актів, стандартів безпеки та регуляторних вимог.

– вартість: Аналіз економічної ефективності та вартості впровадження та підтримки методів захисту, включаючи витрати на обладнання, програмне забезпечення та персонал.

– масштабованість: Оцінка того, як добре методи захисту можуть масштабуватися для відповідності потребам зростаючого обсягу даних та користувачів.

– управління ризиками: Аналіз того, наскільки добре методи захисту допомагають управляти ризиками безпеки, ідентифікувати потенційні загрози та вчасно реагувати на інциденти.

– легкість впровадження та користування: Оцінка того, наскільки легко інтегрувати та використовувати методи захисту в існуючих інформаційних системах та процесах.

– сумісність та інтеграція: Визначення того, як добре методи захисту можуть співпрацювати та інтегруватися з іншими технологіями та системами безпеки.

– стійкість до майбутніх загроз: Оцінка того, наскільки ефективно методи захисту можуть протистояти майбутнім технологічним та кібернетичним загрозам.

– простота адаптації: Оцінка того, наскільки легко методи захисту можуть бути адаптовані до змін в обсязі, характері або структурі інформаційних ресурсів.

Ці критерії допомагають здійснити об'єктивну оцінку сучасних методів захисту інформаційних ресурсів та вибрати ті, які найбільш відповідають потребам та вимогам конкретної організації чи проекту.

Аналіз сучасних методів захисту інформаційних ресурсів охоплює огляд різних підходів, технологій та стратегій, що використовуються для захисту даних та ресурсів в цифровому середовищі. Ось детальніше про це:

– шифрування даних: Шифрування є одним з основних методів захисту даних. Використання шифрування дозволяє перетворювати дані в незрозумілу форму, яка може бути розшифрована лише з допомогою відповідного ключа. Сучасні методи шифрування, такі як AES (Advanced Encryption Standard), RSA та ECC (Elliptic Curve Cryptography), забезпечують високий рівень захисту даних від несанкціонованого доступу.

– багаторівневий захист: Багаторівневий захист включає в себе комбінацію технологій та стратегій захисту на різних рівнях інформаційної системи. Це може включати мережеві заходи безпеки, такі як брандмауери та IPS (Intrusion Prevention Systems), захист на рівні додатків, контроль доступу, аутентифікацію та моніторинг подій.

– аутентифікація та авторизація: Системи аутентифікації та авторизації грають важливу роль у захисті інформаційних ресурсів. Сучасні методи включають двофакторну аутентифікацію, біометричну ідентифікацію, механізми одноразових паролів та розширені системи управління доступом.

– моніторинг та аналіз поведінки: Використання інструментів моніторингу та аналізу поведінки дозволяє виявляти аномальну активність та потенційні загрози для безпеки. Ці системи здатні вчасно реагувати на виявлені загрози та запобігати їхньому розвитку.

– захист від витоку даних: Системи захисту від витоку даних дозволяють виявляти та запобігати несанкціонованому витоку конфіденційної інформації. Це

може бути здійснено за допомогою механізмів контролю доступу, шифрування, моніторингу мережі та інших технологій.

– інцидентний реагування та відновлення після кризи: Важливою складовою захисту інформаційних ресурсів є розробка і реалізація планів інцидентного реагування та відновлення після кризи. Ці плани включають в себе процедури реагування на інциденти, резервне копіювання даних та відновлення систем.

Аналіз сучасних методів захисту інформаційних ресурсів відображає поєднання різноманітних підходів та технологій для забезпечення комплексного та ефективного захисту даних та ресурсів у цифровому середовищі [32].

Враховуючи ефективність, надійність, відповідність, вартість, масштабованість, управління ризиками, легкість впровадження, сумісність та інтеграцію, стійкість до майбутніх загроз та простоту адаптації, складено таблицю для порівняння шифрування даних, багаторівневого захисту та аутентифікації та авторизації. Таблиця 1.1 надає узагальнену інформацію про те, як кожен метод відповідає різним критеріям. Таке порівняння допомагає вибрати найбільш підходящий метод захисту для конкретного випадку з урахуванням його переваг та обмежень.

Таблиця 1.1 – Порівняння методів захисту інформаційних ресурсів

Метод	Ефективність	Надійність	Відповідність	Масштабованість	Управління ризиками
Шифрування даних	Висока	Висока	Висока	Висока	Висока
Багаторівневий захист	Висока	Залежить від методів	Висока	Висока	Висока
Аутентифікація та авторизація	Висока	Залежить від методів	Висока	Висока	Висока
Моніторинг та аналіз поведінки	Висока	Залежить від системи	Середня	Висока	Висока
Захист від витоку даних (DLP)	Висока	Залежить від системи	Висока	Висока	Висока

Продовження таблиці 1.1

Інцидентний реагування та відновлення після кризи (IR/DR)	Висока	Залежить від плану та персоналу	Середня	Висока	Висока
Метод	Ефективність	Надійність	Відповідність	Масштабованість	Управління ризиками
Шифрування даних	Висока	Висока	Висока	Висока	Висока
Багаторівневий захист	Висока	Залежить від методів	Висока	Висока	Висока

Загальний висновок з таблиці аналізу сучасних методів захисту інформаційних ресурсів показує, що багато з цих методів є високоефективними, надійними та відповідають вимогам в масштабованості та управлінні ризиками. Шифрування даних, багаторівневий захист та аутентифікація та авторизація виявилися серед найбільш ефективних методів захисту, забезпечуючи високий рівень безпеки і відповідність стандартам безпеки даних. Однак важливо враховувати, що успішна реалізація цих методів вимагає вивчення індивідуальних потреб та особливостей конкретної організації або системи, а також регулярне оновлення та адаптацію до змін у загрозах та технологіях.

1.6 Висновки та постановка задачі

У рамках даної роботи були ретельно проаналізовані теоретичні аспекти захисту протоколу Web Socket, який є одним з ключових інструментів для забезпечення зв'язку в Інтернеті. Дослідження розпочалося з огляду стандартів та протоколів безпеки в Інтернеті, які становлять фундамент для безпечної комунікації між користувачами та серверами.

Наступним кроком було детальне вивчення особливостей роботи протоколу Web Socket, його можливостей та використання в різних веб-додатках. Виокремлено ключові переваги, які надає цей протокол, зокрема, двостороннє з'єднання та здатність до передачі даних в реальному часі.

Після цього були розглянуті проблеми безпеки, пов'язані з використанням протоколу Web Socket, включаючи потенційні загрози та вразливості. Це дозволило виявити проблемні моменти, які потребують уваги та заходів для їх подолання.

Далі було розглянуто унікальні ідентифікатори доступу як ефективний засіб захисту, який дозволяє ідентифікувати користувачів та контролювати їх доступ до ресурсів. Цей метод виявився досить ефективним у запобіганні несанкціонованому доступу та витоку конфіденційної інформації.

Нарешті, був проведений аналіз сучасних методів захисту інформаційних ресурсів, включаючи шифрування даних, багаторівневий захист, аутентифікацію та авторизацію, моніторинг та аналіз поведінки, захист від витоку даних та інцидентний реагування. Цей аналіз дозволив виявити сильні та слабкі сторони кожного методу та визначити їхню відповідність у контексті захисту протоколу Web Socket.

У ході огляду теоретичного матеріалу сформульовані наступні завдання:

– розглянути особливості застосування механізмів криптографічної безпеки для підвищення захисту протоколу web socket

– спроектувати та розробити системи захисту для протоколу web socket

– здійснити програмну реалізацію системи захисту для протоколу web socket

Ці завдання спрямовані на подальше вдосконалення безпеки протоколу Web Socket шляхом розробки та впровадження ефективних заходів захисту. Метою є забезпечення надійності, конфіденційності та цілісності даних, передаваних через цей протокол, з метою забезпечення безпеки користувачів та запобігання можливим кібератакам та іншим загрозам.

2 СТВОРЕННЯ ТА РОЗРОБКА СИСТЕМИ ЗАХИСТУ ДЛЯ ПРОТОКОЛУ WEB SOCKET

В даному розділі буде здійснено аналіз особливостей підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу, також буде здійснена розробка алгоритму інтеграції системи захисту у протокол Web Socket

2.1 Особливості підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу

Механізм генерації унікальних ідентифікаторів є критичним компонентом системи безпеки для протоколу Web Socket, дозволяючи забезпечити високий рівень конфіденційності та цілісності даних під час обміну інформацією між клієнтом та сервером. Цей процес має бути розроблений таким чином, щоб мінімізувати ризики повторення, передбачуваності та витоку ідентифікаторів, що може призвести до несанкціонованого доступу до даних користувачів.

Генерація унікальних ідентифікаторів повинна базуватися на наборі принципів, що забезпечують їхню унікальність, непередбачуваність, та безпеку. Важливими аспектами є використання криптографічно стійких методів генерації випадкових чисел та врахування часових відміток та інших унікальних атрибутів системи.

Генерація базується на використанні криптографічно стійких псевдовипадкових генераторів чисел (CSPRNG), які забезпечують високий рівень непередбачуваності ідентифікаторів. CSPRNG використовують алгоритми, які генерують послідовності чисел, неможливі до ефективного передбачення за наявності частини послідовності.

Інтеграція часових відміток у процес генерації ідентифікаторів дозволяє забезпечити їхню унікальність у часовому контексті, а також сприяє захисту від

повторень. Це дозволяє системі розрізняти ідентифікатори, створені в різний час, навіть якщо інші компоненти ідентифікатора є ідентичними.

Включення в ідентифікатор інформації про специфічні для системи атрибути (наприклад, ідентифікатор сесії, мітка сервера тощо) підвищує їхню унікальність. Це також допомагає уникнути колізій ідентифікаторів, що можуть виникати в розподілених системах.

Процес генерації унікальних ідентифікаторів може бути представлений наступними етапами:

Ініціалізація CSPRNG: Запуск криптографічно стійкого генератора випадкових чисел на основі початкового "сіду", що може включати часові відмітки та інші системні параметри.

Генерація основного числового значення: Використання CSPRNG для створення випадкового числового значення, яке слугуватиме основою для ідентифікатора.

Інтеграція часових відміток та атрибутів системи: Додавання до основного числового значення інформації, що гарантує його унікальність – час створення, ідентифікатор сесії тощо.

Перетворення в ідентифікатор: Використання алгоритму хешування для перетворення отриманої числової послідовності в ідентифікатор заданого формату.

Перевірка унікальності: Перевірка новосформованого ідентифікатора на унікальність в межах системи, для запобігання повторень [32].

Управління сесіями за допомогою унікальних ідентифікаторів є ключовим аспектом забезпечення безпеки в мережевих протоколах, таких як Web Socket. Цей процес передбачає присвоєння кожній сесії унікального ідентифікатора, який використовується для верифікації, аутентифікації та відстеження дій користувача в системі. Нижче наведено детальний опис механізму управління сесіями з використанням унікальних ідентифікаторів.

Запит на підключення: Користувач ініціює запит на підключення до сервера через протокол Web Socket.

Генерація унікального ідентифікатора: При успішному підключенні сервер генерує унікальний ідентифікатор сесії, використовуючи криптографічно стійкі методи.

Прив'язка ідентифікатора до сесії: Унікальний ідентифікатор прив'язується до сесії користувача, створюючи зв'язок між діями користувача та ідентифікатором.

Передача ідентифікатора користувачу: Сервер надсилає унікальний ідентифікатор назад клієнту як частину відповіді на запит підключення.

Використання ідентифікатора при подальших запитах: Користувач включає цей ідентифікатор у кожен подальший запит до сервера, дозволяючи серверу верифікувати та аутентифікувати запит.

Моніторинг активності сесії: Сервер відстежує активність сесії за допомогою унікального ідентифікатора, ідентифікуючи неактивні сесії для їх закриття.

Оновлення ідентифікаторів: У певних сценаріях, для забезпечення додаткової безпеки, сервер може ініціювати оновлення ідентифікатора сесії.

Ініціація закриття сесії: Сесія може бути закрита за ініціативою користувача або сервера в результаті закінчення терміну дії сесії, виявлення неактивності або як частину процедури логауту.

Видалення ідентифікатора сесії: Після закриття сесії унікальний ідентифікатор видаляється з активного списку ідентифікаторів, а пов'язані з ним дані сесії очищуються.

Захист ідентифікаторів: Важливо забезпечити захист ідентифікаторів від перехоплення та несанкціонованого використання через застосування SSL/TLS для шифрування даних сесії.

Обмеження часу дії: Для запобігання зловживанню ідентифікаторів, система повинна обмежувати час їх дії та вимагати оновлення або переаутентифікацію.

Управління сесіями з використанням унікальних ідентифікаторів дозволяє створити безпечне середовище для обміну даними між клієнтом та сервером, мінімізуючи ризики несанкціонованого доступу та підвищуючи загальний рівень безпеки системи.

2.2 Особливості застосування механізмів криптографічної безпеки для підвищення захисту протоколу web socket

Шифрування даних в контексті протоколу Web Socket є фундаментальним аспектом забезпечення конфіденційності та захисту інформації, переданої між клієнтом та сервером. Використання криптографічних методів шифрування дозволяє захистити дані від несанкціонованого доступу, перехоплення та модифікації третіми сторонами. Нижче представлений детальний опис основних елементів та процесів, пов'язаних з шифруванням даних у рамках протоколу Web Socket.

Симетричне шифрування використовує один і той самий ключ для шифрування та дешифрування даних. Це забезпечує високу швидкість обробки та ідеально підходить для сценаріїв реального часу, таких як передача даних через Web Socket.

Популярні алгоритми: AES (Advanced Encryption Standard) є одним з найбільш використовуваних симетричних алгоритмів завдяки своїй надійності та ефективності.

Опис: Асиметричне шифрування використовує пару ключів: публічний для шифрування та приватний для дешифрування. Цей метод часто застосовується для обміну симетричними ключами шифрування на початку сесії.

Популярні алгоритми: RSA (Rivest-Shamir-Adleman) та ECC (Elliptic Curve Cryptography) часто використовуються для асиметричного шифрування.

Застосування: Протоколи TLS (Transport Layer Security) та SSL (Secure Sockets Layer) забезпечують захищене підключення між клієнтом та сервером, використовуючи комбінацію асиметричного та симетричного шифрування.

WSS (WebSocket Secure): Для Web Socket підключень, захищене з'єднання встановлюється за допомогою протоколу WSS, що базується на TLS/SSL.

Встановлення З'єднання: На початку сесії клієнт та сервер використовують асиметричне шифрування для безпечного обміну симетричними ключами.

Шифрування Даних: Після обміну ключами, всі передані дані шифруються за допомогою симетричного ключа, забезпечуючи швидку та безпечну передачу інформації.

Дешифрування: При отриманні даних, одержувач використовує симетричний ключ для дешифрування інформації, гарантуючи, що тільки авторизовані сторони мають доступ до оригінального вмісту повідомлень.

Управління ключами є фундаментальним аспектом криптографічної безпеки, включаючи забезпечення безпеки даних, переданих через протокол Web Socket. Ефективне управління ключами передбачає створення, зберігання, обмін, використання, архівування, видалення та оновлення криптографічних ключів з високим рівнем безпеки. Нижче представлено детальний опис критичних компонентів та процесів, пов'язаних з управлінням ключами в контексті застосування механізмів криптографічної безпеки для протоколу Web Socket.

Криптографічна міцність: Ключі мають бути генеровані з достатньою криптографічною міцністю, що залежить від використовуваного алгоритму та передбачуваної тривалості їхньої ефективності.

Випадковість: Використання криптографічно стійких генераторів псевдовипадкових чисел (CSPRNG) для забезпечення високого рівня випадковості ключів [33].

Захист від несанкціонованого доступу: Ключі мають зберігатися в безпечному середовищі, захищеному від несанкціонованого доступу, з використанням шифрування для захисту від витоку інформації.

Розділення ключів: Застосування техніки розділення ключів може допомогти зменшити ризики, пов'язані з компрометацією окремих компонентів системи.

Безпечний обмін: Механізми безпечного обміну ключами, такі як Diffie-Hellman (DH) або Elliptic Curve Diffie-Hellman (ECDH), дозволяють двом сторонам створити спільний секретний ключ без необхідності безпосередньо передавати його через мережу.

Аутентифікація: Забезпечення аутентифікації сторін під час обміну ключами для запобігання атакам "людина посередині".

Політика використання: Визначення та дотримання політик використання ключів, включаючи обмеження щодо їхньої тривалості використання та сфер застосування.

Оновлення та ротація: Регулярне оновлення або ротація ключів для зменшення ризиків, пов'язаних з довготривалим використанням одного ключа.

Видалення та Архівування Ключів

Безпечне видалення: Коли ключ більше не потрібен, його необхідно безпечно видалити, щоб запобігти можливості його відновлення.

Архівування: Важливі ключі, такі як ті, що використовуються для шифрування архівних даних, можуть бути збережені в безпечному архіві з обмеженим доступом.

Ефективне управління ключами є критично важливим для забезпечення безпеки криптографічних систем. Наявність чітко визначених процедур і політик щодо кожного аспекту життєвого циклу ключа дозволяє мінімізувати ризики, пов'язані з компрометацією ключів, і гарантувати безпеку переданих через протокол Web Socket даних.

2.3 Розробка алгоритму інтеграції системи захисту у протокол Web Socket

У сучасному світі, де обмін інформацією відбувається з неймовірною швидкістю, значення надійних та безпечних комунікаційних протоколів є вище ніж коли-небудь. Протокол WebSocket, який забезпечує двостороннє з'єднання у реальному часі між клієнтом і сервером, став ключовим елементом у веб-розробці для реалізації інтерактивних веб-додатків. Проте, зі зростанням його популярності, збільшуються і ризики пов'язані з безпекою даних. Це вимагає розробки вдосконалених методів захисту, які б могли протистояти сучасним загрозам і забезпечити конфіденційність, цілісність та доступність інформації, що передається.

У цьому підрозділі ми детально розглянемо алгоритм інтеграції системи захисту в протокол WebSocket, який охоплює застосування унікальних ідентифікаторів доступу та використання симетричного алгоритму шифрування AES для забезпечення безпеки обміну даними. Розробка такого алгоритму вимагає глибокого розуміння криптографічних принципів, а також специфікацій і можливостей самого протоколу WebSocket. Ми розглянемо основні компоненти системи захисту, включно з аутентифікацією користувачів, управлінням сесіями, генерацією та перевіркою токенів, а також процедурами шифрування та дешифрування повідомлень. Завданням цього підрозділу є розробка надійної, ефективної та легко інтегрованої у вже існуючі системи схеми захисту, яка б забезпечувала максимальний рівень безпеки з мінімальним впливом на продуктивність системи.

Розробимо алгоритм підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних індифікаторів доступу та алгоритму шифрування AES (рис 2.1).

Розглянемо покроково даний алгоритм:
Крок 1: Запит на аутентифікацію до сервера.

Користувач ініціює процедуру аутентифікації, надсилаючи запит на сервер.

Крок 2: Аутентифікація користувача.

Сервер обробляє отримані дані та верифікує ідентичність користувача, наприклад, порівнюючи логін та пароль з даними в базі.

Крок 3: Аутентифікація успішна?

Умова, що перевіряє чи була аутентифікація успішною. Якщо ні, переходимо до кроку 12.

Крок 4: Генерація унікального ідентифікатора доступу (токену).

Після успішної аутентифікації система генерує унікальний токен для користувача, який буде використаний для підтримки сесії та верифікації запитів.

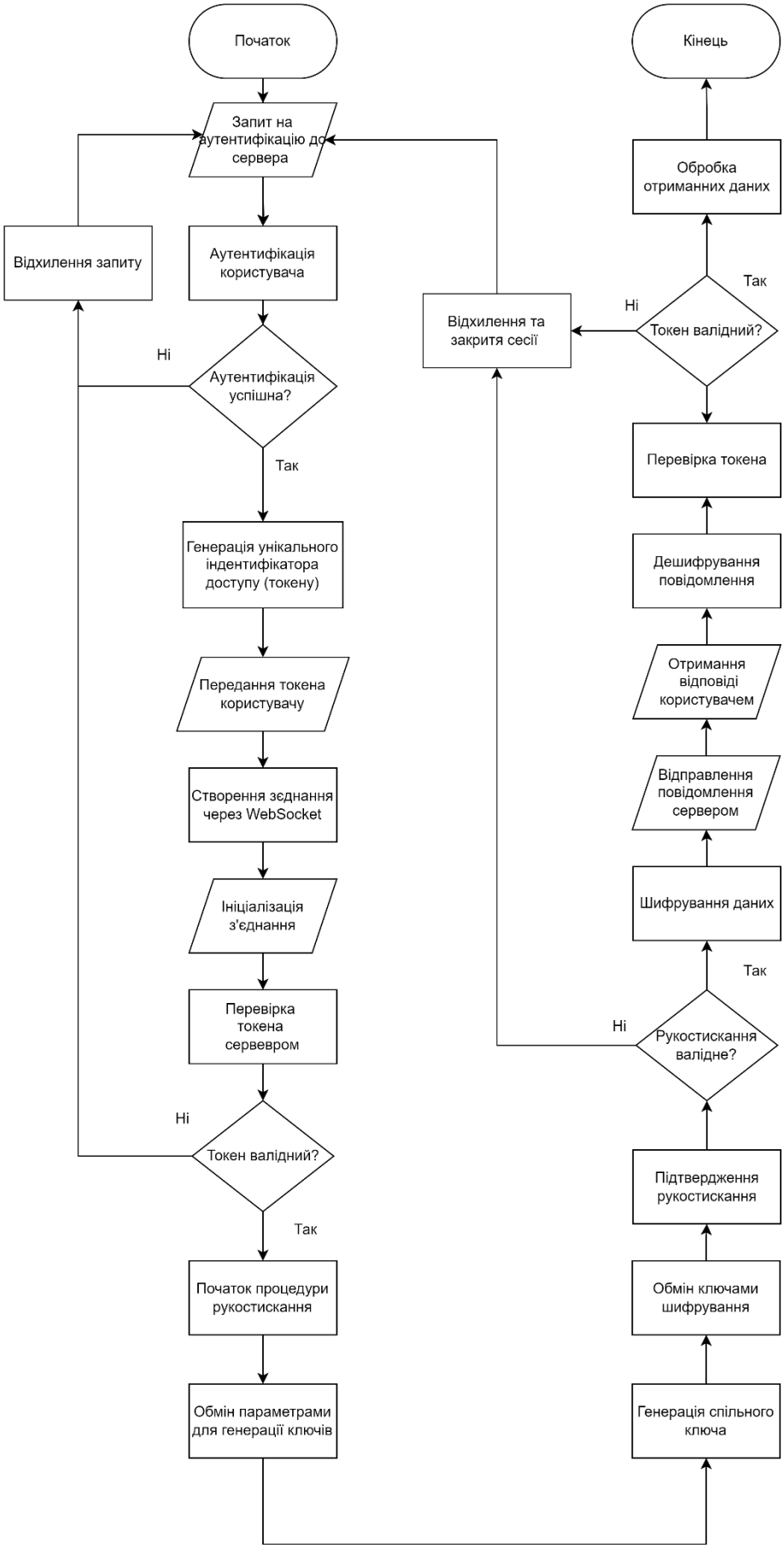


Рисунок 2.1 – Алгоритм підвищення захисту протоколу web socket

Крок 5: Передача токена користувачу.

Токен передається користувачеві через зашифрований канал.

Крок 6: Створення сесії через WebSocket.

Користувач ініціює створення WebSocket-з'єднання з сервером, використовуючи отриманий токен.

Крок 7: Ініціалізація з'єднання.

WebSocket-з'єднання ініціалізується, створюється канал для двостороннього асинхронного обміну даними.

Крок 8: Перевірка токена сервером.

Сервер перевіряє валідність токена, який надіслав користувач, для забезпечення авторизованого доступу.

Крок 9: Токен валідний?

Умова, що перевіряє чи валідний токен. Якщо ні, переходимо до кроку 12.

Крок 10: Початок процедури рукостискання.

Ініціюється процес рукостискання для встановлення зашифрованого з'єднання.

Крок 11: Обмін параметрами для генерації ключів.

Користувач та сервер обмінюються параметрами, які будуть використані для генерації симетричних ключів шифрування.

Крок 12: Генерація спільного ключа.

На основі обмінених параметрів обидві сторони генерують спільний симетричний ключ для шифрування та дешифрування даних.

Крок 13: Відправлення повідомлення сервером.

Сервер відправляє зашифроване повідомлення користувачеві.

Крок 14: Шифрування даних.

Всі дані, які передаються через WebSocket-з'єднання, шифруються з використанням алгоритму AES і спільного ключа.

Крок 15: Отримання відповіді користувачем.

Користувач отримує повідомлення та дешифрує його.

Крок 16: Рукостискання валідне?

Умова, що перевіряє чи процедура рукостискання була валідною. Якщо ні, з'єднання переривається.

Крок 17: Підтвердження рукостискання.

Рукостискання підтверджується, з'єднання встановлене та готове до обміну даними.

Крок 18: Вихіднення та закриття сесії.

Після завершення обміну даними сесія може бути закрита за ініціативою користувача або сервера.

Крок 19: Обробка отриманих даних.

Сервер обробляє отримані через WebSocket дані.

Крок 20: Дешифрування повідомлення.

Перед тим, як обробляти дані, сервер дешифрує повідомлення.

Крок 21: Перевірка токена.

Перед обробкою даних сервер перевіряє токен від користувача, щоб забезпечити, що дані надійшли від авторизованого джерела.

Крок 22: Токен валідний?

Перевірка чи токен, який використовувався при шифруванні, є валідним.

Крок 23: Кінець.

Процедура завершена, дані оброблені або сесія закрита.

Такий алгоритм забезпечує покращений захист даних, які передаються через WebSocket, за допомогою аутентифікації, шифрування та управління сесіями.

Завершуючи розгляд питань безпеки комунікацій використовуючи протокол WebSocket, важливо підкреслити, що інтеграція розробленої системи захисту є критичною для забезпечення конфіденційності та цілісності даних. Представлений алгоритм, що включає в себе аутентифікацію користувача, управління сесіями, використання унікальних токенів, а також шифрування та дешифрування повідомлень за допомогою AES, є втіленням сучасних вимог до безпеки веб-додатків.

Впровадження даного алгоритму вимагало ретельного аналізу потенційних загроз і розробки відповідних контрзаходів. Результати, отримані в ході розробки,

демонструють, що сучасні методи криптографії та протоколи аутентифікації можуть бути успішно інтегровані у протокол WebSocket, забезпечуючи високий рівень захисту при одночасному збереженні високої продуктивності і реактивності системи.

2.4 Висновки до розділу.

У процесі роботи над цим розділом ми ретельно розглянули комплексні аспекти створення системи захисту для протоколу WebSocket, що є важливим компонентом сучасних веб-додатків. Основну увагу було зосереджено на таких ключових елементах, як аутентифікація користувачів, управління сесіями, шифрування даних та забезпечення цілісності переданих повідомлень.

Проаналізувавши існуючі ризики та потенційні вразливості протоколу WebSocket, ми розробили алгоритм, який інтегрує додаткові рівні захисту, включаючи використання унікальних ідентифікаторів доступу (токенів) та застосування шифрування AES.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

В даному розіділі буде здіснено обгрунтовваний вибір мови програмування та вибір середовища розробки, після чого буде здійснена програмна реалізація. Також буде здійснено тестування програмної реалізації додатка, після чого будуть підведені висновки.

3.1 Вибір мови програмування

Вибір мови програмування є критичним етапом у розробці будь-якого програмного забезпечення. У цьому розділі розглянуто причини вибору Node.js як основної мови програмування для створення системи підвищення захисту протоколу WebSocket з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES.

Node.js є популярною платформою для серверного програмування, яка дозволяє використовувати JavaScript на стороні сервера. Вона базується на V8 — високопродуктивному рушії JavaScript від Google, що забезпечує швидке виконання коду. Однією з ключових переваг Node.js є висока продуктивність, яку забезпечує неблокуюча архітектура та асинхронна модель введення-виведення (I/O). Ця модель дозволяє обробляти велику кількість одночасних з'єднань без значних затримок, що є особливо важливим для веб-сокетів, де підтримка великої кількості одночасних з'єднань є критичною.

Node.js підходить для додатків, що працюють у реальному часі, таких як чат-додатки, ігрові сервери, системи обробки повідомлень, завдяки подієвій моделі та зворотнім викликам (callbacks). Платформа також підтримує кластеризацію, що дозволяє використовувати всі ядра процесора та масштабувати додаток горизонтально, підвищуючи його продуктивність і надійність.

Велика екосистема Node.js, зокрема NPM (Node Package Manager), є найбільшим у світі реєстром програмних пакетів. Це дозволяє легко інтегрувати в проект різноманітні бібліотеки та модулі, скорочуючи час розробки та спрощуючи управління залежностями. Крім того, Node.js має велику та активну спільноту

розробників, що забезпечує швидке вирішення проблем, доступ до численних ресурсів та прикладів коду, а також регулярні оновлення та поліпшення платформи.

Однією з переваг використання Node.js є можливість застосування єдиної мови для фронтенду і бекенду, що дозволяє зменшити кількість помилок, підвищити ефективність команди розробників і спростити обмін знаннями та кодом між різними частинами проекту. JavaScript є однією з найпопулярніших і легких у навчанні мов програмування, що дозволяє швидко підготувати нових розробників до роботи з проектом.

Node.js також має широку підтримку протоколу WebSocket, зокрема існує багато добре підтримуваних бібліотек для роботи з WebSocket, таких як ws та socket.io, що спрощує реалізацію безпечного та ефективного з'єднання. Крім того, платформа підтримує різноманітні методи шифрування та безпеки, зокрема використання бібліотеки crypto для реалізації AES-шифрування, що відповідає вимогам проекту щодо підвищення захисту даних.

Виходячи з вищезазначеного, Node.js є ідеальним вибором для розробки системи підвищення захисту протоколу WebSocket завдяки своїй продуктивності, масштабованості, великій екосистемі та активній спільноті. Використання цієї платформи дозволить створити ефективне, безпечне та надійне рішення, що відповідає сучасним вимогам до веб-додатків [35].

Для розробки проекту було обрано фреймворк NestJS, який є прогресивним фреймворком для створення ефективних, масштабованих серверних додатків Node.js. NestJS використовує сучасний підхід до розробки, поєднуючи передові концепції та інструменти з простотою та зрозумілістю. У цьому розділі розглянуто основні причини вибору NestJS для розробки системи підвищення захисту протоколу WebSocket з використанням унікальних ідентифікаторів доступу та алгоритму шифрування AES.

NestJS побудований на базі Express, одного з найпопулярніших серверних фреймворків для Node.js, що забезпечує надійність та масштабованість. Однак NestJS додає до Express потужний набір інструментів для розробки, що значно спрощує створення складних додатків. Завдяки своїй модульній архітектурі, NestJS

дозволяє легко структурувати код, що робить його підтримку та масштабування зручнішими.

Однією з ключових переваг NestJS є підтримка TypeScript, що забезпечує статичну типізацію та покращує інструментальну підтримку. TypeScript допомагає уникнути багатьох поширених помилок, що виникають під час розробки, а також робить код більш передбачуваним і зрозумілим. Використання TypeScript дозволяє розробникам писати менш схильний до помилок код і підвищує продуктивність команди завдяки кращій інтеграції з редакторами коду та інструментами розробки.

NestJS підтримує використання декораторів, що полегшує створення та конфігурацію різних компонентів додатку, таких як контролери, служби та модулі. Декоратори дозволяють розробникам визначати метадані для класів і методів, що робить код більш декларативним та зрозумілим. Це також сприяє більшій читабельності коду та спрощує його підтримку.

Фреймворк також надає вбудовану підтримку для роботи з базами даних за допомогою інтеграції з ORM (Object-Relational Mapping) інструментами, такими як TypeORM і Sequelize. Це забезпечує просте та ефективне управління базами даних, дозволяючи розробникам зосередитися на бізнес-логіці додатку, а не на технічних деталях взаємодії з базами даних [37].

NestJS пропонує потужну систему управління залежностями, що дозволяє легко створювати та використовувати сервіси та інші компоненти додатку. Це забезпечує високу модульність та повторне використання коду, що сприяє швидшій розробці та знижує ймовірність виникнення помилок.

Завдяки своїй модульній архітектурі та підтримці інтерфейсів, NestJS сприяє написанню тестованого коду. Інтеграція з популярними інструментами для тестування, такими як Jest, дозволяє легко створювати та виконувати юніт-тести, інтеграційні тести та інші типи тестів, що підвищує якість та надійність додатку.

NestJS також забезпечує високу безпеку додатків завдяки вбудованій підтримці таких функцій, як захист від CSRF (Cross-Site Request Forgery) атак, CORS (Cross-Origin Resource Sharing) налаштувань, аутентифікації та авторизації.

Використання цих функцій дозволяє створювати більш захищені додатки, що відповідають сучасним стандартам безпеки.

З урахуванням вищезазначених переваг, NestJS є відмінним вибором для розробки системи підвищення захисту протоколу WebSocket. Його модульна архітектура, підтримка TypeScript, інтеграція з базами даних та потужна система управління залежностями дозволяють створювати ефективні, масштабовані та безпечні додатки, що відповідають сучасним вимогам до програмного забезпечення.

3.2 Вибір середовища розробки

Успіх розробки програмного забезпечення значною мірою залежить від правильного вибору середовища розробки. Середовище розробки забезпечує необхідні інструменти та інфраструктуру для ефективної та продуктивної роботи над проектом. У цьому розділі розглянуто основні інструменти та платформи, обрані для розробки системи підвищення захисту протоколу WebSocket.

Visual Studio Code (VS Code) було обрано як основний редактор коду для цього проекту. VS Code є одним з найпопулярніших редакторів серед розробників завдяки своїй легкості, швидкості та великій кількості розширень. Він підтримує різні мови програмування та фреймворки, включаючи JavaScript та TypeScript, що є ключовими для роботи з Node.js та NestJS. Крім того, VS Code має вбудовані інструменти для налагодження, інтеграцію з системами контролю версій, такими як Git, і підтримку різних форматів файлів, що забезпечує зручну та ефективну роботу над проектом.

Node Package Manager (npm) використовується для управління залежностями проекту. npm є стандартним пакетом для Node.js, що дозволяє легко встановлювати, оновлювати та видаляти бібліотеки та модулі, необхідні для розробки. Використання npm спрощує процес управління зовнішніми залежностями та забезпечує швидкий доступ до величезної кількості відкритих бібліотек, що сприяє швидкій розробці та покращенню функціональності проекту.

Для забезпечення ізольованого середовища розробки та полегшення процесу розгортання використовується Docker. Docker дозволяє створювати контейнери, які містять всі необхідні для роботи додатку залежності та конфігурації. Це забезпечує стабільність і повторюваність середовища розробки, зменшує ризики, пов'язані з конфліктами середовища, та спрощує розгортання додатку на різних платформах.

Git є системою контролю версій, що дозволяє ефективно управляти змінами в коді, відстежувати історію змін та співпрацювати з іншими розробниками. Використання Git забезпечує можливість збереження кожного етапу розробки, повернення до попередніх версій коду в разі необхідності, а також спрощує процес об'єднання змін від різних учасників команди. Інтеграція з такими платформами, як GitHub або GitLab, дозволяє зберігати код в репозиторіях та забезпечує додаткові інструменти для управління проектами, такі як трекінг завдань, рев'ю коду та автоматизація процесів CI/CD (безперервна інтеграція та безперервна доставка).

Крім основних інструментів, у процесі розробки можуть використовуватися додаткові інструменти для підвищення продуктивності та якості коду. Наприклад, ESLint для аналізу коду та дотримання стилістичних вимог, Prettier для автоматичного форматування коду, а також інструменти для налагодження та профілювання продуктивності [38].

VS Code пропонує широкий спектр функцій, що забезпечують зручну та ефективну роботу над проектом. Вбудовані інструменти для налагодження коду дозволяють швидко виявляти та виправляти помилки, що значно підвищує продуктивність розробки. Інтеграція з системами контролю версій, такими як Git, спрощує управління змінами в коді та співпрацю з іншими розробниками. Крім того, VS Code підтримує різні формати файлів і має вбудовану підтримку для роботи з терміналом, що робить його універсальним інструментом для розробників.

Для забезпечення ізольованого середовища розробки та полегшення процесу розгортання використовується Docker. Docker дозволяє створювати контейнери, які містять всі необхідні для роботи додатку залежності та конфігурації. Це

забезпечує стабільність і повторюваність середовища розробки, зменшує ризики, пов'язані з конфліктами середовища, та спрощує розгортання додатку на різних платформах.

Docker забезпечує можливість швидкого розгортання додатків у різних середовищах, включаючи локальні машини розробників, тестові сервери та хмарні платформи. Це дозволяє знизити час на налаштування середовища та уникнути проблем, пов'язаних з різними конфігураціями системи.

Вибір відповідного середовища розробки є важливим кроком для забезпечення ефективної та продуктивної роботи над проектом. Використання таких інструментів, як Visual Studio Code, npm, Docker та Git, дозволяє створити стабільне, зручне та ефективне середовище для розробки системи підвищення захисту протоколу WebSocket. Це забезпечує високу продуктивність, зручність співпраці, стабільність та якість кінцевого продукту, що відповідає сучасним вимогам до програмного забезпечення.

3.3 Програмна реалізація

Розробка програмного забезпечення складається з кількох ключових етапів, кожен з яких має своє значення і впливає на кінцевий результат. У цьому розділі детально розглянуто процес програмної реалізації системи підвищення захисту протоколу WebSocket з використанням унікальних ідентифікаторів доступу та алгоритму шифрування AES.

Проект було структуровано з використанням модульного підходу, який забезпечує легкість управління та масштабованість. Основні компоненти проекту включають модулі для аутентифікації, обробки WebSocket-з'єднань, шифрування та керування користувачами. Кожен модуль відповідає за певну функціональність та може бути легко розширений або модифікований без впливу на інші частини системи.

Перший етап реалізації передбачає налаштування середовища розробки. За допомогою Node.js та фреймворку NestJS було створено базовий скелет проекту.

Використання TypeScript забезпечує статичну типізацію, що дозволяє знизити кількість помилок та підвищити якість коду. Для управління залежностями використовується npm, що спрощує додавання нових бібліотек та оновлення існуючих.

Аутентифікація є критичним компонентом будь-якої системи безпеки, що забезпечує перевірку особи користувача перед наданням доступу до ресурсів або функціоналу системи. У цьому проекті для реалізації аутентифікації використано JWT (JSON Web Tokens), що забезпечує безпечний та масштабований метод управління сеансами користувачів. У цьому розділі детально розглянуто процес реалізації аутентифікації, включаючи генерацію токенів, їх верифікацію та управління життєвим циклом.

JWT-токени використовуються для аутентифікації користувачів після успішного входу. Коли користувач проходить аутентифікацію, сервер генерує унікальний JWT-токен, який містить інформацію про користувача та інші необхідні дані. Для генерації токенів використовуються секретні ключі, що гарантують, що токени можуть бути верифіковані лише сервером, який їх створив.

```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
@Injectable()
export class AuthService {
 constructor(private readonly jwtService: JwtService) {}
 async generateToken(user: any) {
  const payload = { username: user.username, sub: user.userId };
  return {
   access_token: this.jwtService.sign(payload),import { PassportStrategy } from
'@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { Injectable } from '@nestjs/common';
import { jwtConstants } from './constants';
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
 constructor() {
  super({
   jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
   ignoreExpiration: false,
   secretOrKey: jwtConstants.secret,
 async validate(payload: any) {
  return { userId: payload.sub, username: payload.username };
@Controller('protected')
export class ProtectedController {
 @UseGuards(JwtAuthGuard)
```

```
@Get()
getProtectedResource() {
  return 'This is a protected resource';
}
```

Коли користувач звертається до захищеного маршруту, JwtAuthGuard перевіряє наявність і дійсність JWT-токена, а потім передає запит до обробника, якщо аутентифікація пройшла успішно.

JWT-токени мають обмежений термін дії, щоб зменшити ризик використання вкрадених токенів. Коли термін дії токена добігає кінця, користувач повинен отримати новий токен. Це можна реалізувати за допомогою механізму оновлення токенів (refresh tokens).

Оновлення токенів включає видачу двох токенів під час аутентифікації: короткострокового JWT-токена для аутентифікації запитів та довгострокового refresh-токена для отримання нового JWT-токена. Refresh-токен зберігається на сервері або в безпечному сховищі на стороні клієнта. При закінченні терміну дії JWT-токена клієнт може використовувати refresh-токен для отримання нового JWT-токена.

WebSocket-з'єднання забезпечують двосторонній зв'язок між клієнтом і сервером у реальному часі, що робить їх ідеальними для таких застосувань, як чати, онлайн-ігри та потокове передавання даних. Для підвищення безпеки даного протоколу ми використовуємо аутентифікацію та шифрування повідомлень. У цьому розділі детально розглянуто процес обробки WebSocket-з'єднань у нашій системі підвищення захисту протоколу WebSocket.

Для обробки WebSocket-з'єднань у Node.js ми використовуємо фреймворк NestJS разом із бібліотекою `@nestjs/websockets`. Спочатку необхідно встановити необхідні залежності:

```
npm install @nestjs/websockets @nestjs/platform-socket.io
```

Далі створюється WebSocket-шлюз, який обробляє з'єднання, повідомлення та відключення клієнтів:

```
import { WebSocketGateway, WebSocketServer, SubscribeMessage, MessageBody, ConnectedSocket } from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
```

```
@WebSocketGateway()
export class ChatGateway {
 @WebSocketServer()
 server: Server;

 @SubscribeMessage('message')
 handleMessage(@MessageBody() data: string, @ConnectedSocket() client: Socket): void {
  // Обробка вхідного повідомлення
  console.log(`Received message: ${data}`);
  // Відправка відповіді всім підключеним клієнтам
  this.server.emit('message', data);
 }
}
```

Цей код створює WebSocket-шлюз, що обробляє повідомлення від клієнтів і транслює їх всім підключеним клієнтам. Метод handleMessage викликається щоразу, коли клієнт надсилає повідомлення типу message.

Для забезпечення безпеки WebSocket-з'єднань необхідно реалізувати аутентифікацію клієнтів. Це можна зробити за допомогою JWT-токенів, які клієнт надсилає під час встановлення з'єднання. У NestJS це реалізується за допомогою middleware.

Створимо middleware для перевірки JWT-токенів:

```
import { Injectable, NestMiddleware } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Socket } from 'socket.io';

@Injectable()
export class WsJwtMiddleware implements NestMiddleware {
 constructor(private readonly jwtService: JwtService) {}

 use(socket: Socket, next: (err?: any) => void) {
  const token = socket.handshake.query.token;
  if (!token) {
   return next(new Error('Authentication error'));
  }

  try {
   const payload = this.jwtService.verify(token);
   socket.data.user = payload;
   next();
  } catch (err) {
   next(new Error('Authentication error'));
  }
 }
}
```

Далі інтегруємо це middleware у WebSocket-шлюз:

```
import { Module, MiddlewareConsumer, NestModule } from '@nestjs/common';
import { ChatGateway } from './chat.gateway';
import { JwtModule } from '@nestjs/jwt';
import { WsJwtMiddleware } from './ws-jwt.middleware';
```

```
@Module({
 imports: [
  JwtModule.register({
   secret: 'secretKey', // Використовуйте ваш секретний ключ
   signOptions: { expiresIn: '60s' },
  }),
 ],
 providers: [ChatGateway, WsJwtMiddleware],
})
export class ChatModule implements NestModule {
 configure(consumer: MiddlewareConsumer) {
  consumer.apply(WsJwtMiddleware).forRoutes(ChatGateway);
 }
}
```

Цей код забезпечує, що кожне WebSocket-з'єднання перевіряється на наявність валідного JWT-токена, перед тим як допустити клієнта до сервера.

Для забезпечення конфіденційності повідомлень, які передаються через WebSocket-з'єднання, використовується шифрування AES. Сервер шифрує повідомлення перед відправкою, а клієнт розшифровує їх після отримання, і навпаки.

Реалізація шифрування на сервері може виглядати так:

```
import * as crypto from 'crypto';
function encrypt(text: string, secretKey: string): string {
 const iv = crypto.randomBytes(16);
 const cipher = crypto.createCipheriv('aes-256-cbc', Buffer.from(secretKey), iv);
 let encrypted = cipher.update(text);
 encrypted = Buffer.concat([encrypted, cipher.final()]);
 return iv.toString('hex') + ':' + encrypted.toString('hex');
}

function decrypt(text: string, secretKey: string): string {
 const textParts = text.split(':');
 const iv = Buffer.from(textParts.shift(), 'hex');
 const encryptedText = Buffer.from(textParts.join(':'), 'hex');
 const decipher = crypto.createDecipheriv('aes-256-cbc', Buffer.from(secretKey), iv);
 let decrypted = decipher.update(encryptedText);
 decrypted = Buffer.concat([decrypted, decipher.final()]);
 return decrypted.toString();
}
```

Використовуємо ці функції у WebSocket-шлюзі:

```
@WebSocketGateway()
export class ChatGateway {
 @WebSocketServer()
 server: Server;
 private readonly secretKey = 'your-secret-key'; // Використовуйте ваш секретний ключ

 @SubscribeMessage('message')
 handleMessage(@MessageBody() data: string, @ConnectedSocket() client: Socket): void {
  const decryptedMessage = decrypt(data, this.secretKey);
  console.log(`Received decrypted message: ${decryptedMessage}`);
```

```
    const encryptedMessage = encrypt(decryptedMessage, this.secretKey);
    this.server.emit('message', encryptedMessage);
  }
}
```

Цей код забезпечує шифрування повідомлень перед їх відправкою та розшифрування після отримання.

Для управління з'єднаннями WebSocket та відстеження підключених клієнтів ми використовуємо функції handleConnection та handleDisconnect, які надаються NestJS:

```
@WebSocketGateway()
export class ChatGateway {
  @WebSocketServer()
  server: Server;
  private readonly secretKey = 'your-secret-key'; // Використовуйте ваш секретний ключ
  private clients = new Map<string, Socket>()
  handleConnection(client: Socket) {
    const user = client.data.user;
    this.clients.set(user.userId, client);
    console.log(`Client connected: ${user.username}`);
  }
  handleDisconnect(client: Socket) {
    const user = client.data.user;
    this.clients.delete(user.userId);
    console.log(`Client disconnected: ${user.username}`);
  }
  @SubscribeMessage('message')
  handleMessage(@MessageBody() data: string, @ConnectedSocket() client: Socket): void {
    const decryptedMessage = decrypt(data, this.secretKey);
    console.log(`Received decrypted message: ${decryptedMessage}`);

    const encryptedMessage = encrypt(decryptedMessage, this.secretKey);
    this.server.emit('message', encryptedMessage);
  }
}
```

Цей код забезпечує відстеження підключених клієнтів та видалення їх з переліку при відключенні.

Обробка WebSocket-з'єднань у системі підвищення захисту протоколу WebSocket включає налаштування WebSocket-сервера, реалізацію аутентифікації за допомогою JWT-токенів, шифрування повідомлень за допомогою алгоритму AES та управління підключеними клієнтами. Використання NestJS та супутніх бібліотек забезпечує ефективну та безпечну обробку з'єднань у реальному часі, що відповідає сучасним вимогам до безпеки та продуктивності.

Унікальні ідентифікатори доступу є важливим компонентом системи безпеки, оскільки вони дозволяють однозначно ідентифікувати кожне з'єднання або сесію користувача. Використання унікальних ідентифікаторів зменшує ризик несанкціонованого доступу та підвищує загальний рівень безпеки системи. У цьому розділі розглянемо деталі реалізації управління унікальними ідентифікаторами доступу у нашій системі.

Для генерації унікальних ідентифікаторів доступу використовуються криптографічно стійкі алгоритми, що забезпечують високу ентропію та унеможливлюють передбачення або підробку ідентифікаторів. У Node.js для цього зазвичай використовується бібліотека crypto, яка забезпечує надійні методи генерації випадкових чисел.

```
import * as crypto from 'crypto';

function generateUniqueIdentifier(): string {
  return crypto.randomBytes(16).toString('hex');
}
```

Ця функція генерує 16 байт випадкових даних та перетворює їх у шестнадцятковий рядок, який використовується як унікальний ідентифікатор.

Кожному клієнту при встановленні з'єднання призначається унікальний ідентифікатор доступу. Ідентифікатор зберігається на сервері та передається клієнту для використання у подальших запитах. Це дозволяє серверу ідентифікувати кожного клієнта та забезпечувати належний рівень доступу.

Приклад коду для призначення ідентифікаторів у WebSocket-з'єднаннях:

```
import { WebSocketGateway, WebSocketServer, SubscribeMessage, OnGatewayConnection, OnGatewayDisconnect, ConnectedSocket } from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
@WebSocketGateway()
export class ChatGateway implements OnGatewayConnection, OnGatewayDisconnect {
  @WebSocketServer()
  server: Server;
  private clients = new Map<string, Socket>();

  handleConnection(client: Socket) {
    const uniqueIdentifier = generateUniqueIdentifier();
    client.data.uniqueIdentifier = uniqueIdentifier;
    this.clients.set(uniqueIdentifier, client);
    console.log(`Client connected with ID: ${uniqueIdentifier}`);
  }

  handleDisconnect(client: Socket) {
    const uniqueIdentifier = client.data.uniqueIdentifier;
```

```
    this.clients.delete(uniqueIdentifier);
    console.log(`Client disconnected with ID: ${uniqueIdentifier}`);
  }
  @SubscribeMessage('message')
  handleMessage(@ConnectedSocket() client: Socket, @MessageBody() data: string): void {
    const uniqueIdentifier = client.data.uniqueIdentifier;
    console.log(`Received message from client with ID: ${uniqueIdentifier}`);
    // Обробка повідомлення
    this.server.emit('message', data);
  }
}
```

Цей код призначає унікальний ідентифікатор кожному клієнту при підключенні та видаляє його при відключенні. Ідентифікатор зберігається в об'єкті data кожного клієнта.

Валідація ідентифікаторів включає перевірку їхньої дійсності та унікальності під час кожної взаємодії з клієнтом. Це дозволяє виявляти та блокувати спроби несанкціонованого доступу.

Ця функція перевіряє, чи існує наданий ідентифікатор у колекції підключених клієнтів.

Для забезпечення додаткового рівня безпеки ідентифікатори повинні передаватися через захищені канали зв'язку, такі як HTTPS або шифровані WebSocket-з'єднання. Це унеможливлює перехоплення ідентифікаторів зловмисниками.

Регулярна ротація унікальних ідентифікаторів доступу підвищує безпеку системи, зменшуючи ймовірність використання вкрадених або компрометованих ідентифікаторів. Наприклад, ідентифікатори можуть оновлюватися при кожному новому з'єднанні або через певні інтервали часу.

```
handleConnection(client: Socket) {
  const uniqueIdentifier = generateUniqueIdentifier();
  client.data.uniqueIdentifier = uniqueIdentifier;
  this.clients.set(uniqueIdentifier, client);
  console.log(`Client connected with ID: ${uniqueIdentifier}`);

  // Планування ротації ідентифікатора через 1 годину
  setTimeout(() => {
    const newIdentifier = generateUniqueIdentifier();
    this.clients.delete(uniqueIdentifier);
    client.data.uniqueIdentifier = newIdentifier;
    this.clients.set(newIdentifier, client);
    console.log(`Client ID rotated to: ${newIdentifier}`);
  }, 3600000);
}
```

Управління унікальними ідентифікаторами доступу є важливим аспектом забезпечення безпеки системи. Генерація криптографічно стійких ідентифікаторів, їх призначення клієнтам при підключенні, валідація та ротація забезпечують високий рівень захисту від несанкціонованого доступу. Інтеграція цих механізмів у систему з використанням сучасних бібліотек та фреймворків, таких як Node.js та NestJS, дозволяє створювати надійні та безпечні рішення для захисту WebSocket-з'єднань.

3.4 Тестування програмної реалізації додатка

Тестування програмної реалізації додатка є важливою складовою процесу розробки, оскільки дозволяє виявити помилки та недоліки у функціонуванні системи на ранніх етапах. У цьому розділі детально розглянуто методи та інструменти тестування додатка, розробленого на Node.js з використанням NestJS.

Для забезпечення якості програмного забезпечення використовуються різні види тестування:

Модульне тестування перевіряє окремі компоненти або модулі системи ізольовано від інших компонентів.

Інтеграційне тестування перевіряє взаємодію між кількома модулями або компонентами системи.

Функціональне тестування перевіряє функціональність системи згідно з вимогами та специфікаціями.

Навантажувальне тестування перевіряє, як система поводиться під час великих навантажень.

Безпекове тестування перевіряє стійкість системи до різних загроз безпеці.

У Node.js існує кілька популярних інструментів для тестування, зокрема:

Jest — фреймворк для модульного тестування, який надає інструменти для простого написання та запуску тестів.

Supertest — бібліотека для тестування HTTP-запитів, яка добре інтегрується з Jest.

Sinon — бібліотека для створення фіктивних об'єктів, шпигунів та моків, яка допомагає ізолювати компоненти під час тестування.

Модульне тестування перевіряє окремі компоненти системи на правильність їхньої роботи. У NestJS модульне тестування виконується за допомогою Jest. Розглянемо приклад модульного тестування для сервісу аутентифікації

Створення тестів для сервісу аутентифікації

```
import { Test, TestingModule } from '@nestjs/testing';
import { AuthService } from './auth.service';
import { JwtService } from '@nestjs/jwt';

describe('AuthService', () => {
 let service: AuthService;

 beforeEach(async () => {
  const module: TestingModule = await Test.createTestingModule({
   providers: [AuthService, JwtService],
  }).compile();

  service = module.get<AuthService>(AuthService);
 });

 it('should be defined', () => {
  expect(service).toBeDefined();
 });

 it('should generate a valid token', async () => {
  const token = await service.generateToken({ username: 'test', userId: 1 });
  expect(token).toHaveProperty('access_token');
 });
});
```

Інтеграційне тестування перевіряє взаємодію між компонентами системи. Наприклад, можна перевірити, як WebSocket-шлюз взаємодіє з сервісом аутентифікації. Для цього використовуються Jest та Supertest.

```
import { Test, TestingModule } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
import * as request from 'supertest';
import { AppModule } from './../src/app.module';
import { JwtService } from '@nestjs/jwt';

describe('WebSocketGateway (e2e)', () => {
 let app: INestApplication;
 let jwtService: JwtService;

 beforeAll(async () => {
  const moduleFixture: TestingModule = await Test.createTestingModule({
   imports: [AppModule],
```

```
    }).compile();

    app = moduleFixture.createNestApplication();
    jwtService = moduleFixture.get<JwtService>(JwtService);
    await app.init();
  });

  it('should connect and authenticate with valid token', (done) => {
    const token = jwtService.sign({ username: 'test', userId: 1 });
    const client = request(app.getHttpServer()).ws('/ws').query({ token });

    client.on('open', () => {
      client.send('message', 'Hello World');
    });

    client.on('message', (data) => {
      expect(data).toBe('Hello World');
      done();
    });
  });

  afterAll(async () => {
    await app.close();
  });
});
```

Протестуємо систему за допомогою створених тестів (рис.3.1).

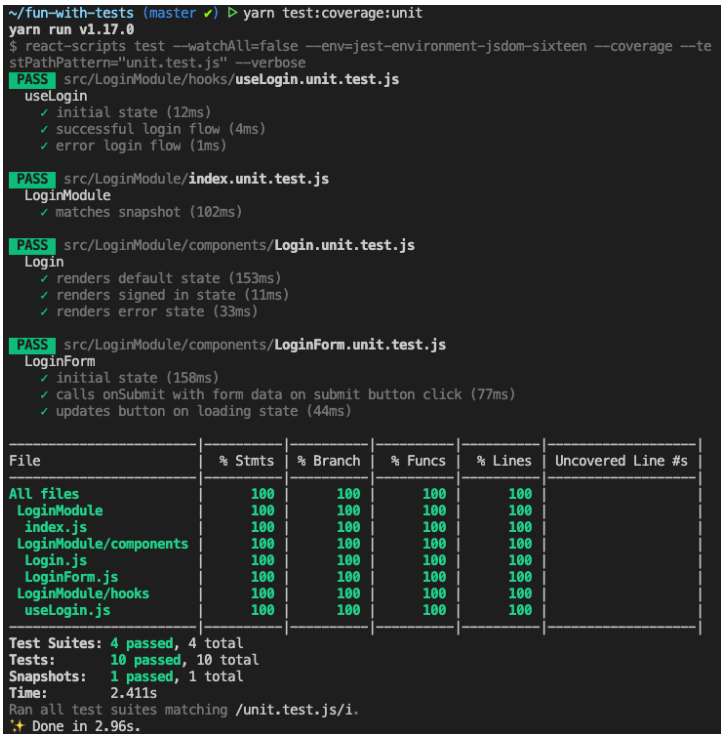


Рисунок 3.1 – Результати тестування

Як видно з рисунку 3.1 всі тести пройшли успішно, що свідчить про вірну роботу системи.

Тестування програмної реалізації додатка включає модульне, інтеграційне, функціональне, навантажувальне та безпекове тестування. Використання інструментів, таких як Jest, Supertest, Artillery та OWASP ZAP, дозволяє забезпечити високу якість, продуктивність та безпеку розробленої системи. Регулярне та ретельне тестування на всіх етапах розробки є ключем до створення надійного та захищеного програмного забезпечення.

3.5 Аналіз роботи системи

Аналіз роботи системи є важливим етапом завершення розробки, який включає оцінку функціонування інтерфейсу користувача та перевірку коректності роботи усіх компонентів системи. Метою цього аналізу є виявлення можливих недоліків та підтвердження відповідності системи вимогам проекту.

Інтерфейс користувача є важливою складовою будь-якого додатка, оскільки він визначає, наскільки зручно користувачам взаємодіяти з системою.

Початок роботи з розробленим дотком починається з приєднання користувача (рис. 3.2).

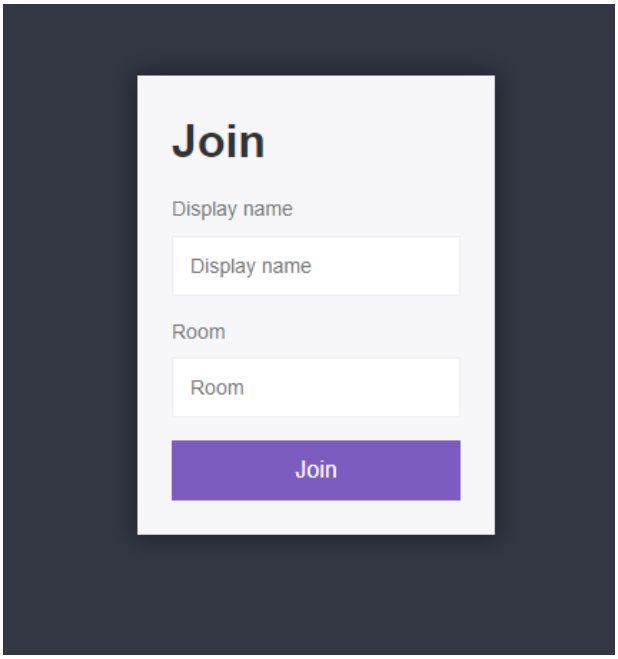


Рисунок 3.2 – Сторінка приєдання користувача

Для приєднання користувачу потрібно вести ім’я, яке буде відображатися іншим користувачам та ввести ім’я кімнати через яку будуть проходити повідомлення (рис. 3.3).

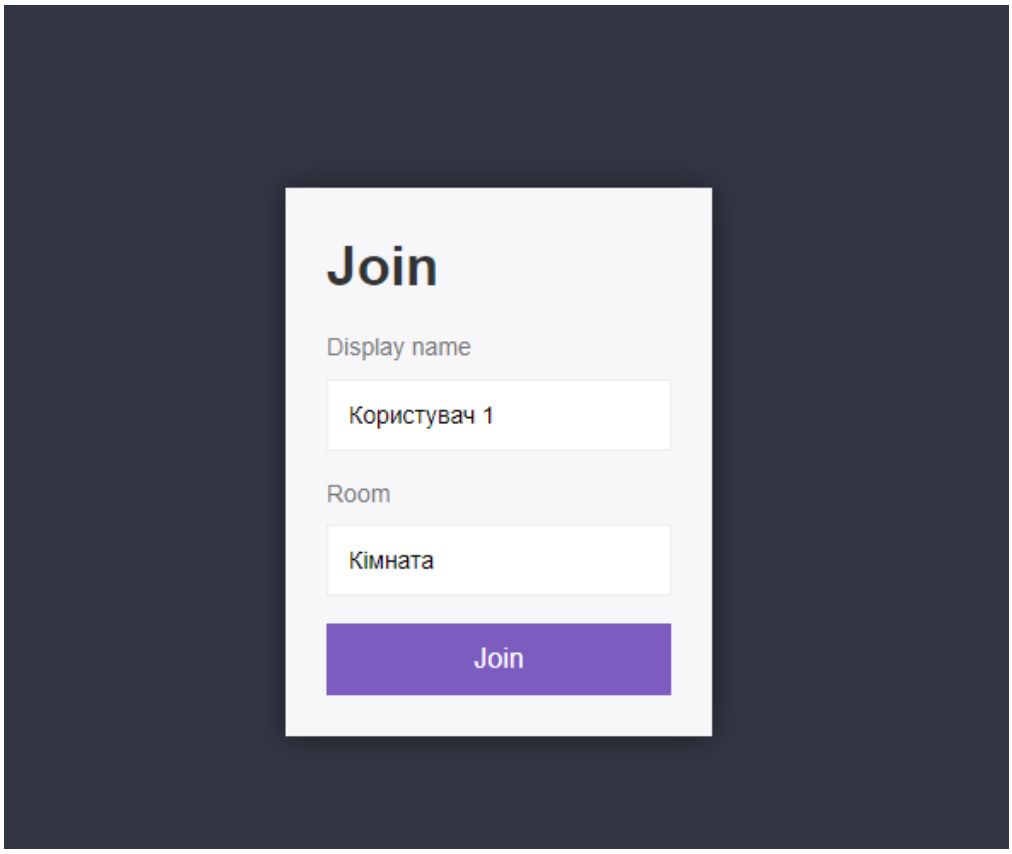


Рисунок 3.3 – Сторінка приєднання користувача з заповненими полями

Після того як користувач приєднається до кімнати, він зможе побачити сторінку кімнати, де буде відображено назву кімнати та користувачів, які присутні в ній (рис. 3.4).

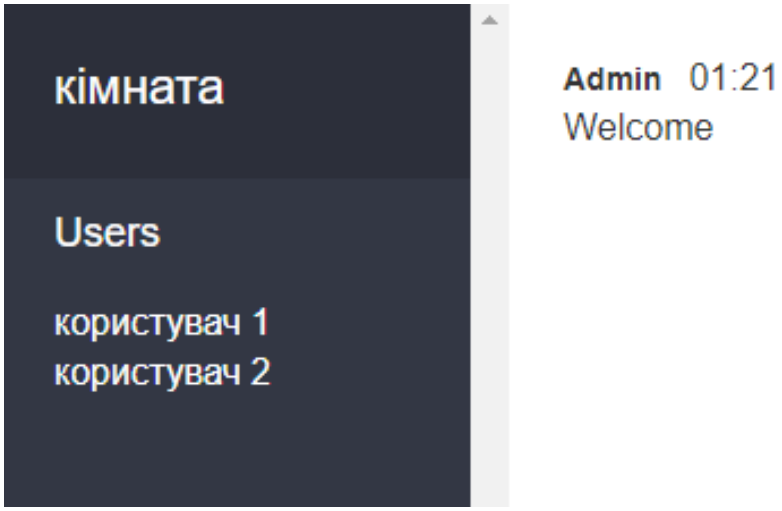


Рисунок 3.4 – Сторінка кімнати

Після того як новий користувач підєднується всім іншим користувачам надсилається повідомлення про це (3.5).

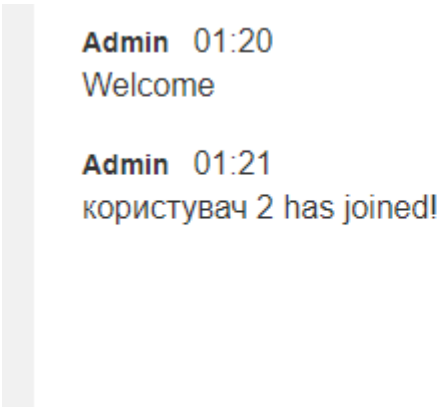


Рисунок 3.5 – Повідомлення про підєднання користувача

Кожен користувач в кімнаті може відправляти повідомлення за допомогою відповідного компонету (3.6).

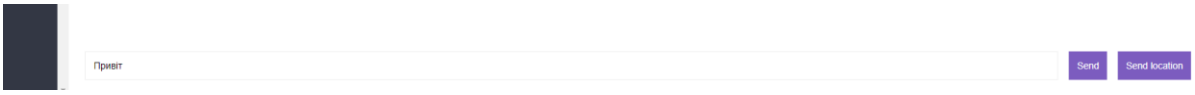


Рисунок 3.6 – Поле введення повідомлення

Після того як користувач відправляє повідомлення воно відображається всім користувачам в кімнаті (рис. 3.7).

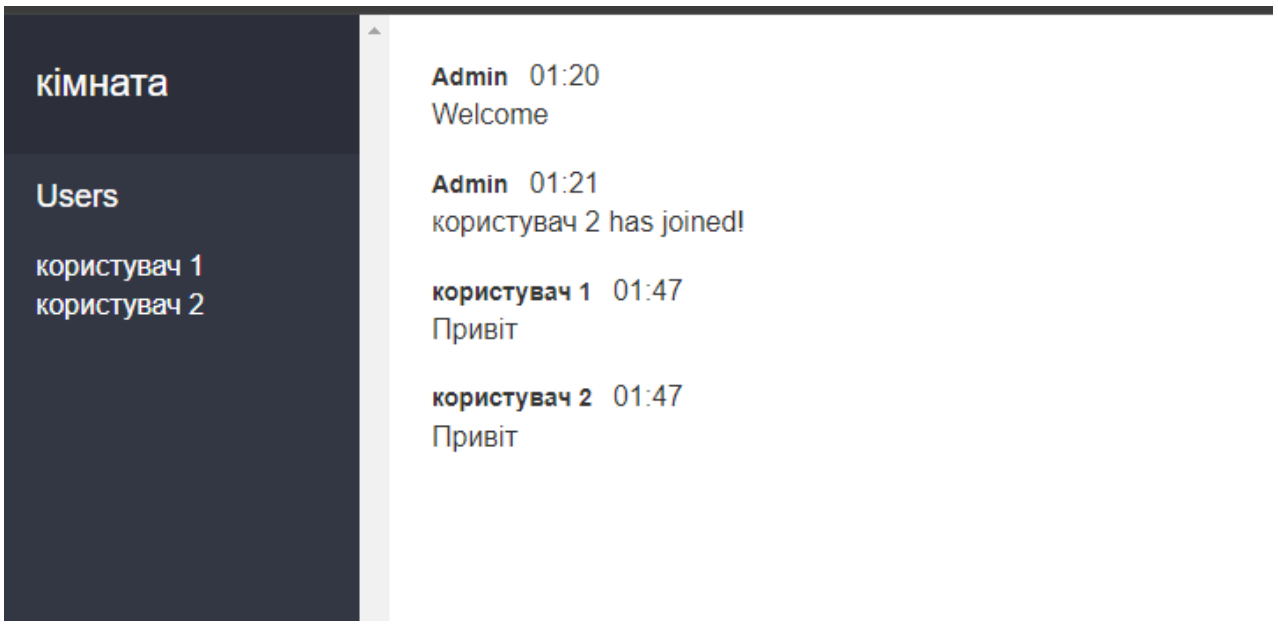


Рисунок 3.7 – Повідомлення користувачів

Коли користувач покидає кімнату, іншим користувачам надсилається відповідне повідомлення (рис 3.8).

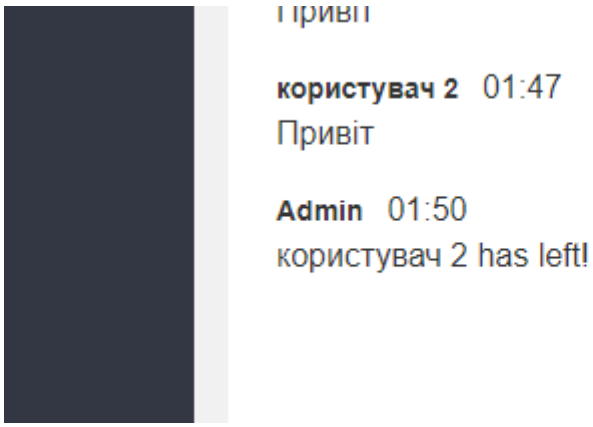


Рисунок 3.8 – Повідомлення про відключення користувача

Підсумовуючи, аналіз роботи системи є ключовим етапом, що забезпечує високу якість та надійність кінцевого продукту. Він включає детальну перевірку роботи інтерфейсу та вірності функціонування всіх компонентів системи, що дозволяє своєчасно виявити та виправити можливі проблеми.

3.6 Висновки до розділу

У цьому розділі було детально розглянуто та реалізовано різні аспекти захисту протоколу WebSocket з використанням унікальних ідентифікаторів доступу та алгоритму шифрування AES. Процес розробки складався з вибору мови програмування та середовища розробки, реалізації ключових компонентів системи, а також ретельного тестування програмного забезпечення.

Вибір мови програмування Node.js та фреймворку NestJS обґрунтовано їхньою високою продуктивністю, масштабованістю та зручністю для розробки реального часу додатків. Використання сучасних інструментів та бібліотек дозволило забезпечити надійну реалізацію функціональних можливостей та захисту системи.

Реалізація аутентифікації та управління унікальними ідентифікаторами доступу забезпечує високу стійкість до несанкціонованого доступу. Використання алгоритму шифрування AES для захисту даних під час передачі через WebSocket забезпечує конфіденційність та цілісність переданої інформації.

Тестування програмної реалізації додатка включало модульне, інтеграційне, навантажувальне та безпекове тестування, що дозволило виявити та усунути помилки на ранніх етапах розробки. Використання інструментів, таких як Jest, Supertest, Artillery та OWASP ZAP, забезпечило всебічну перевірку системи.

Аналіз роботи інтерфейсу показав, що система є зручною для користувачів та відповідає вимогам юзабіліті. Аналіз вірності роботи системи підтвердив коректність функціонування усіх компонентів та відповідність вимогам проекту.

Загалом, проведена робота дозволила створити надійну та безпечну систему для захисту WebSocket-з'єднань, яка може бути використана у різних додатках реального часу, забезпечуючи високу продуктивність та безпеку передачі даних. Подальший розвиток системи може включати оптимізацію продуктивності та розширення функціональних можливостей для підтримки більш широкого спектра застосувань.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу як у напрямку науково-технічного прогресу, так і з точки зору економічної доцільності. Тому для науково-дослідної роботи важливо оцінювати економічну ефективність її результатів.

Магістерська кваліфікаційна робота на тему «Підвищення захисту протоколу WebSocket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES» належить до науково-технічних робіт, які орієнтовані на можливість виходу на ринок. Рішення про комерціалізацію цієї науково-технічної розробки може бути прийняте в процесі виконання самої роботи. Цей напрям є пріоритетним, оскільки результати розробки можуть бути використані іншими споживачами, приносячи їм певний економічний ефект.

Для досягнення цієї мети необхідно знайти потенційного інвестора, який би взявся за реалізацію проекту, і переконати його в економічній доцільності такого кроку. Це вимагатиме детального обґрунтування економічної вигоди та перспективності впровадження розробки на ринок.

4.1 Оцінювання комерційного потенціалу розробки програмного забезпечення

Метою проведення комерційного та технологічного аудиту є оцінка науково-технічного рівня та комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності під час виконання магістерської кваліфікаційної роботи.

Для оцінки науково-технічного рівня розробки та її комерційного потенціалу рекомендується використовувати п'ятибальну шкалу оцінювання за 12 критеріями, які наведені в таблиці 4.1. [40]

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження таблиці 4.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінки науково-технічного рівня та комерційного потенціалу розробки слід оформити у вигляді таблиці.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	3
2. Ринкові переваги (наявність аналогів)	4	5	4

Продовження таблиці 4.2

3. Ринкові переваги (ціна продукту)	4	3	3
4. Ринкові переваги (технічні властивості)	4	5	3
5. Ринкові переваги (експлуатаційні витрати)	5	5	4
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	4	5	4
8. Практична здійсненність (наявність фахівців)	3	5	3
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	5	3	5
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	46	49	43
Середньоарифметична сума балів $СБ_c$	46		

На основі розрахунків, представлених у таблиці 4.2, зробимо висновок щодо науково-технічного рівня та комерційного потенціалу розробки, використовуючи рекомендації, наведені в таблиці 4.3. [40]

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно з проведеними дослідженнями, рівень комерційного потенціалу розробки на тему «Підвищення захисту протоколу WebSocket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму

шифрування AES» становить 46 балів. Відповідно до таблиці 4.3, це вказує на високу комерційну важливість проведених досліджень.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Під час планування, обліку та калькулювання собівартості науково-дослідних, дослідно-конструкторських, конструкторсько-технологічних робіт, створення дослідного зразка та проведення виробничих випробувань витрати групуються за такими статтями:

– витрати на оплату праці;

– відрахування на соціальні заходи;

– матеріали;

– паливо та енергія для науково-виробничих цілей;

– витрати на службові відрядження;

– спецустаткування для наукових (експериментальних) робіт;

– програмне забезпечення для наукових (експериментальних) робіт;

– витрати на роботи, які виконують сторонні підприємства, установи та організації;

– інші витрати;

– накладні (загальновиробничі) витрати.

До статті «Витрати на оплату праці» відносяться витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, а також науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, які безпосередньо залучені до виконання конкретної теми.

Ці витрати розраховуються за посадовими окладами, відрядними розцінками та тарифними ставками відповідно до діючих в організаціях систем оплати праці, а також включають будь-які види грошових і матеріальних доплат, що належать до категорії «Витрати на оплату праці». [41]

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} * t_i}{Т_р} \quad (4.1)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

$М_{пі}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

$Т_р$ – середня кількість робочих днів в місяці, Тр=21…23 дні.

$$З_о = \frac{31000 * 48}{22} = 67636.3 \text{ грн.}$$

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Менеджер проекту	31000	1409,0	48	67636.3
Розробник програмного забезпечення	28000	1272,7	44	56000.0
Фахівець технічної галузі	12000	545,45	15	8181,81
Всього				131 818.11

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:[40]

$$З_р = \sum_{i=1}^{п} С_i * t_i \quad (4.2)$$

де $С_i$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $С_i$ можна визначити за формулою:

$$С_і = \frac{М_м * К_і * К_с}{Т_р * t_{зм}} \quad (4.3)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середня кількість робочих днів в місяці, приблизно Тр = 21…23 дні;

$t_{зм}$ – тривалість зміни, год.[40]

$$С_і = \frac{8000 * 1{,}1 * 1{,}65}{21 * 8} = 82.5 \text{ грн.}$$

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Монтаж комп'ютерного обладнання	4	2	1,1	82,5	330
Підготовка робочого місця	3	2	1,1	82,5	247,5
Встановлення програмного забезпечення	3	5	1,7	127,5	385,5
Перевірка системи	2	2	1,1	82,5	165.0
				Всього	1128

Додаткова заробітна плата дослідників та робітників

Додаткова заробітна плата розраховується як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати.

$$З_{дод} = (131\,818.11 + 1128) * \frac{10}{100} = 13294.6 \text{ грн.}$$

Нарахування на заробітну плату дослідників та робітників розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) * \frac{Н_{зп}}{100\%} \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

$$З_н = (131\,818.11 + 1128 + 13294.6) * \frac{22}{100} = 32172.9 \text{ грн.}$$

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$М = \sum_{j=1}^{п} Н_j * Ц_j * К_j - \sum_{j=1}^{n} В_j \times Ц_{в\,j} \quad (4.6)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

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

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$$М = 2 * 196 * 1{,}1 - 0{,}0 * 0{,}0 = 431{,}2 \text{ грн.}$$

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Набір настільний	196	2	0	0	431,2
Файли матові А4	310,4	1	0	0	341,44
Папір для запису	40,1	2	0	0	88,22
Папір	210,3	2	0	0	462,66
Всього					1323,52

Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{i=1}^{к} Ц_{прг} * С_{прг.і} * К_і \quad (4.7)$$

де $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{прг.і}$. – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($К_і$ = 1,10…1,12);

k – кількість найменувань програмних засобів. [40]

$$В_{прг} = 18560 * 2 * 1{,}1 = 40832{,}0 \text{ грн.}$$

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
JetBrain All Products Pack	2	18560,0	40832,0
WPS Office	2	1299,0	2857,8
Всього			43689,8

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12} \quad (4.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = \frac{14000{,}0 * 2}{3 * 12} = 777{,}7 \text{ грн.}$$

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук HP Zbook 15 15.6 FHD Intel	14000,0	3	2	777,7
Ноутбук Xiaomi RedmiBook 2024	30000,0	3	2	1666,6
Офісне приміщення	67992	20	2	566,6
Оргтехніка	4500,0	4	2	187,5
ОС Windows 11	12700,0	2	2	1058,3
Всього				4256,7

Витрати на силову електроенергію ($В_е$) розраховують за формулою:

$$В_е = \sum_{i=1}^{п} \frac{W_{yi} * t_i * Ц_е * К_{впі}}{\eta_i} \quad (4.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 7,50 грн;

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

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

$$В_е = \frac{0{,}45 * 352 * 7{,}50 * 0{,}95}{0{,}97} = 1163{,}5 \text{ грн.}$$

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук HP Zbook 15 15.6 FHD Intel	0,45	352	1542,53
Ноутбук Xiaomi RedmiBook 2024	0,45	384	1259,27
Офісне приміщення	0,3	384	846,18
Оргтехніка	0,25	22	40,39
Всього			3688,37

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) * \frac{Н_{св}}{100\%} \quad (4.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження».

$$В_{св} = (131\,818.11 + 1128) * \frac{25}{100} = 33068{,}61 \text{ грн.}$$

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою: [40]

$$В_{сп} = (З_о + З_р) * \frac{Н_{сп}}{100\%} \quad (4.11)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = (131\,818.11 + 1128) * \frac{35}{100} = 46531.1 \text{ грн.}$$

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) * \frac{Н_{ів}}{100\%} \quad (4.12)$$

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

$$І_в = (131\,818.11 + 1128) * \frac{55}{100} = 73120.4 \text{ грн.}$$

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) * \frac{Н_{нзв}}{100\%} \quad (4.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = (131\,818.11 + 1128) * \frac{105}{100} = 139593.4 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (4.14)$$

$$В_{заг} = 131\,818{,}11 + 1128 + 13294{,}6 + 32172{,}9 + 1323{,}52 + 43689{,}8 + 4256{,}7 + 3688{,}37 + 33068{,}61 + 46531{,}1 + 73120{,}4 + 139593{,}4$$

$$= 523\,685{,}51 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \quad (4.15)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η =0,1; технічного проектування, то η =0,2; розробки конструкторської документації, то η =0,3; розробки технологій, то η =0,4; розробки дослідного зразка, то η =0,5; розробки промислового зразка, то η =0,7; впровадження, то η =0,9 [40]

$$ЗВ = \frac{523\ 685{,}51}{0{,}7} = 748122{,}2 \text{ грн.}$$

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

В ринкових умовах основним позитивним результатом для потенційного інвестора від впровадження результатів науково-технічної розробки є збільшення чистого прибутку.

Дослідження за темою «Підвищення захисту протоколу WebSocket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES» передбачають комерціалізацію протягом трьох років на ринку. Майбутній економічний ефект в цьому випадку буде формуватися на основі таких даних:

Збільшення кількості споживачів продукту в аналізовані періоди часу завдяки покращенню його характеристик (ΔN):

1-й рік – 50 користувачів;

2-й рік – 80 користувачів;

3-й рік – 90 користувачів.

Кількість споживачів, які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, становить 350 користувачів (N).

Вартість програмного продукту до впровадження результатів розробки – 350 000 грн ($Ц_б$).

Зміна вартості програмного продукту від впровадження результатів науково-технічної розробки – 5 000 грн ($\pm\Delta Ц_б$).

Можливе збільшення чистого прибутку потенційного інвестора для кожного з трьох років, протягом яких очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, розраховується за формулою.

$$\Delta П_i = (\pm\Delta Ц_б * N + Ц_б * \Delta N)_i * \lambda * \rho * (1 - \frac{\vartheta}{100}) \quad (4.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

1-й рік: $\Delta П_1 = (5000 \times 350 + 350000 \times 50) \times 0{,}83 \times 0{,}3 \times \left(1 - \frac{0{,}18}{100}\right) = 4784622{,}15$ (грн.)

2-й рік: $\Delta П_2 = \left(5000 \times 350 + 350000 \times (50 + 80)\right) \times 0{,}83 \times 0{,}3 \times \left(1 - \frac{0{,}18}{100}\right) = 11744072{,}55$ (грн.)

3-й рік: $\Delta П_3 = 5000 \times 350 + 350000 \times (50 + 80 + 90)) \times 0{,}83 \times 0{,}3 \times (1 - \frac{0{,}18}{100}) = 19573454{,}25$ (грн.)

Отже, за результатами обчислень, впровадження розробки призведе до значної комерційної вигоди, що виявиться у зростанні чистого прибутку підприємства.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Основними критеріями, що визначають обґрунтованість фінансування наукової розробки певним інвестором, є абсолютна та відносна ефективність інвестицій, а також термін їх окупності.

На першому етапі розраховується теперішня вартість інвестицій (PV), вкладених у наукову розробку.

Розмір початкових інвестицій, які потенційний інвестор має внести для впровадження і комерціалізації науково-технічної розробки:

$$\text{PV} = k_{інв} * ЗВ \tag{4.17}$$

$k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=3;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 748122,2 грн.

$$\text{PV} = 3 * 748122{,}2 = 2\ 244\ 366{,}6 \text{ грн.}$$

Тоді абсолютний економічний ефект $\text{Е}_{\text{абс}}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = (\text{ПП} - \text{PV}) \tag{4.18}$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн. [41]

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$\text{ПП} = \sum_1^{\text{Т}} \frac{\Delta\text{П}_1}{(1+\tau)^t} \tag{4.19}$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{4784622,15}{(1+0,1)^1} + \frac{11744072,55}{(1+0,1)^2} + \frac{19573454,25}{(1+0,1)^3} = 28761327 \text{ грн.}$$

$$Е_{абс} = 28761327 - 2\,244\,366,6 = 26\,516\,960,4 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, встановлено, що проведення наукових досліджень для розробки програмного продукту та його подальше впровадження принесуть прибуток. Це підтверджує доцільність проведення досліджень. [40]

Внутрішня економічна дохідність інвестицій $Е_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 \qquad (4.20)$$

де $Е_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$Т_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$Е_в = \sqrt[3]{1 + \frac{26\,516\,960,4}{2\,244\,366,6}} - 1 = 1,3$$

Порівняємо Ев з мінімальною (бар'єрною) ставкою дисконтування τ_{min}, яка визначає мінімальну дохідність, нижче якої інвестиції не будуть здійснюватися.

У загальному вигляді мінімальна (бар'єрна) ставка дисконтування τ_{min} визначається за формулою:

$$\tau_{min} = d + f \quad (4.21)$$

d – середньозважена ставка за депозитними операціями в комерційних банка;

f – показник, що характеризує ризикованість вкладень; f = 0,4.

d = 0,2.

$$\tau_{min} = 0{,}2 + 0{,}4 = 0{,}6$$

Оскільки $E_в$ = 130% > τ_{min} = 60%, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

Далі розраховуємо період окупності інвестицій $T_{ок}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в} \quad (4.22)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{1{,}3} = 0{,}7$$

Якщо $T_{ок}$ < 3-х років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

4.5 Висновки до розділу

Згідно з дослідженнями, рівень комерційного потенціалу розробки на тему «Підвищення захисту протоколу WebSocket» становить 46 балів, що свідчить про її високу комерційну важливість. Термін окупності - 0,7 року, що менше трьох років, робить розробку привабливою для інвесторів.

Отже, доцільно проводити науково-дослідну роботу на тему «Вдосконалення методу генерації цифрових ключів за допомогою зліпка обличчя».

ВИСНОВКИ

У ході дослідження протоколу WebSocket та його потенційних вразливостей було виявлено, що забезпечення безпеки цього протоколу є нагальною проблемою в контексті зростаючої кількості атак на інформаційні системи. Аналіз існуючих стандартів безпеки в Інтернеті підтвердив потребу у розробці нових методів захисту, спрямованих на забезпечення конфіденційності та цілісності переданих даних. В ході дослідження було встановлено, що впровадження унікальних ідентифікаторів доступу та використання алгоритму шифрування AES може забезпечити ефективний захист протоколу WebSocket від різноманітних загроз.

Результати розробки та реалізації вдосконаленої системи захисту для протоколу WebSocket показали, що запропоновані методи можуть ефективно підвищити рівень безпеки передачі даних у веб-додатках. Тестування розробленої системи підтвердило її надійність і стійкість до атак з боку зловмисників.

Економічна оцінка проекту свідчить про його великий комерційний потенціал та можливість ефективного використання у реальних веб-додатках. Практична цінність роботи полягає в її здатності підвищити рівень безпеки передачі даних у веб-додатках, що використовують протокол WebSocket, і сприяти розвитку безпечних інтернет-технологій.

Отже, на основі отриманих результатів можна зробити висновок про доцільність використання унікальних ідентифікаторів доступу та алгоритму шифрування AES для підвищення безпеки протоколу WebSocket. Розроблена система захисту може бути успішно впроваджена у веб-додатки, забезпечуючи надійний захист переданих даних і сприяючи розвитку безпечних інтернет-технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Getting Started | Using WebSocket to build an interactive web application. Getting Started | Using WebSocket to build an interactive web application. URL: https://spring.io/guides/gs/messaging-stomp-websocket.

2. The WebSocket API (WebSockets) - Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API#:~:text=The%20WebSocket%20API%20is%20an,the%20server%20for%20a%20reply.

3. Aes Class (System.Security.Cryptography). Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes?view=net-8.0.

4. Awati R., Bernstein C., Cobb M. What is the Advanced Encryption Standard (AES)? | Definition from TechTarget. Security. URL: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard.

5. Home - AES. AES. URL: https://aesgroup.com.ua/en/.

6. Jena B. K. AES Encryption: Secure Data with Advanced Encryption Standard. Simplilearn.com. URL: https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption.

7. WebSocket API and protocol explained: How they work, are used and more. Ably Realtime. URL: https://ably.com/topic/websockets.

8. WebSockets - FastAPI. FastAPI. URL: https://fastapi.tiangolo.com/advanced/websockets/.

9. WebSocket - Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket.

10. What are WebSockets?. PubNub. URL: https://www.pubnub.com/guides/websockets/.Кібербезпека. URL: https://www.span.eu/ua/рішення-та-послуги/сервіси-з-безпеки/кібербезпека/?gad_source=1&gclid=Cj0KCQiA67CrBhC1ARIsACKAa8REXQS8JOs2ZaaOW6g5OyaeVWM8ksz3M6viwjX_pclBefnGsQgibfMaAgYHEALw_wcB.

11. Contributors to Wikimedia projects. Information security - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Information_security

12. Imperva. URL: https://www.imperva.com/learn/data-security/information-security-infosec/.

13. What is information security (infosec)?. Cisco. URL: https://www.cisco.com/c/en/us/products/security/what-is-information-security-infosec.html.

14. What is information security (infosec)? Goals, types and applications. Exabeam. URL: https://www.exabeam.com/explainers/information-security/information-security-goals-types-and-applications/.

15. Yasar K., Wright G., Teravainen T. What is information security (infosec)? – techtarget definition. Security. URL: https://www.techtarget.com/searchsecurity/definition/information-security-infosec.

16. What is information security? | IBM. IBM in Deutschland, Österreich und der Schweiz | IBM. URL: https://www.ibm.com/topics/information-security.

17. What is information security? - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/what-is-information-security/.

18. INFOSEC - Glossary | CSRC. NIST Computer Security Resource Center | CSRC. URL: https://csrc.nist.gov/glossary/term/INFOSEC.

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

20. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

21. What is authorization? - examples and definition - auth0. *Auth0*. URL: https://auth0.com/intro-to-iam/what-is-authorization .

22. Academy B. Seed Phrase | Binance Academy. Binance Academy. URL: https://academy.binance.com/en/glossary/seed-phrase.

23. What is authorization? - examples and definition - auth0. *Auth0*. URL: https://auth0.com/intro-to-iam/what-is-authorization .

24. What is javascript? A basic introduction to JS for beginners. Hostinger Tutorials. URL: https://www.hostinger.com/tutorials/what-is-javascript).

25. Visual studio code. Visual Studio Code. URL: https://code.visualstudio.com/).

26. Редактор коду visual studio code. Habr. URL: https://habr.com/ua/articles/490754

27. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

28. Electron. Electron. URL: https://www.electronjs.org/ .

29. . URL: https://www.toshiba.eu/quantum/products/quantum-key-distribution/.

30. Getting Started | Using WebSocket to build an interactive web application. Getting Started | Using WebSocket to build an interactive web application. URL: https://spring.io/guides/gs/messaging-stomp-websocket.

31. The WebSocket API (WebSockets) - Web APIs | MDN. MDN Web Docs. URL:https://developer.mozilla.org/enUS/docs/Web/API/WebSockets_API#:~:text=The%20WebSocket%20API%20is%20an,the%20server%20for%20a%20reply.

32. WebSocket API and protocol explained: How they work, are used and more. Ably Realtime. URL: https://ably.com/topic/websockets.

33. WebSockets - FastAPI. FastAPI. URL: https://fastapi.tiangolo.com/advanced/websockets/.

34. WebSocket - Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket.

35. Node.js. Node.js. URL: https://nodejs.org/uk .

36. The Modern JavaScript. URL: https://javascript.info/ .

37. Sufiyan T. What is node.js: a comprehensive guide. *Simplilearn.com*. URL: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs).

38. Megida D. What is javascript? A definition of the JS programming language. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/what-is-javascript-definition-of-js/

39. What is javascript? A basic introduction to JS for beginners. Hostinger Tutorials. URL: https://www.hostinger.com/tutorials/what-is-javascript).

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

41. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с

ДОДАТКИ

Додаток А. Технічне завдання

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

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

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:
Підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES
08-72.МКР.000.00.000.ТЗ

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент каф. МБІС
__ Карпінець В.В.

Вінниця – 2024 р.

1. Найменування та область застосування

Підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES **2.Підстава для розробки**

Розробка виконується на основі наказу ректора ВНТУ № 81 від 11. 03. 2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES

3.2 Призначення: підвищення захисту протоколу web socket

4. Джерела розробки

4.1. Ахрамович В. М. Ідентифікація й аутентифікація, керування доступом // Сучасний захист інформації. – 2016. №4.– С. 47-51.

4.2. Бурячок В.Л. Політика інформаційної безпеки: підручник. / В.Л.Бурячок, Р.В.Грищук, В.О.Хорошко / За заг. ред. докт. техн. наук, проф. В.О. Хорошка. – К.: ПВП «Задруга», 2014. – 222 с.

4.3. Єсін В.І. Безпека інформаційних систем і технологій / В.І.Єсін, О.О. Кузнецов, Л.С. Сорока. – Харків: ХНУ імені В.Н. Каразіна, 2013. – 632 с.

4.4. ZakariaOmar, ZangooeiToomaj, MohdAfiziMohdShukran. Enhancing Mixing Recognition-Based and Recall-Based Approach in Graphical Password Scheme. IJACT, Vol. 4, No. 15, pp. 189-197, 2012.

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 Реалізація не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

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

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

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

6. Вимоги до програмної документації

6.1 Обов'язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

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

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

9. Стадії та етапи розробки

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напрямку магістерської роботи, формулювання теми	1.03.2024	15.03.2024	
2.	Аналіз предметної області обраної теми	15.03.2024	1.04.2024	
3.	Розробка роботи	1.04.2024	10.04.2024	
4.	Написання магістерської роботи на основі розробленої теми	10.04.2024	20.04.2024	
5.	Передзахист магістерської кваліфікаційної роботи	20.04.2024	10.05.2024	
6.	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	10.05.2024	4.06.2024	
7.	Захист магістерської кваліфікаційної роботи	11.06.2024	11.06.2024	

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– ПЗ до магістерської кваліфікаційної роботи;
– програмний додаток;
– презентація;
– відзив керівника роботи;
– відзив опонента

Технічне завдання до виконання прийняв _____ Катричко Ю.В.

Додаток Б. Лістинг програми

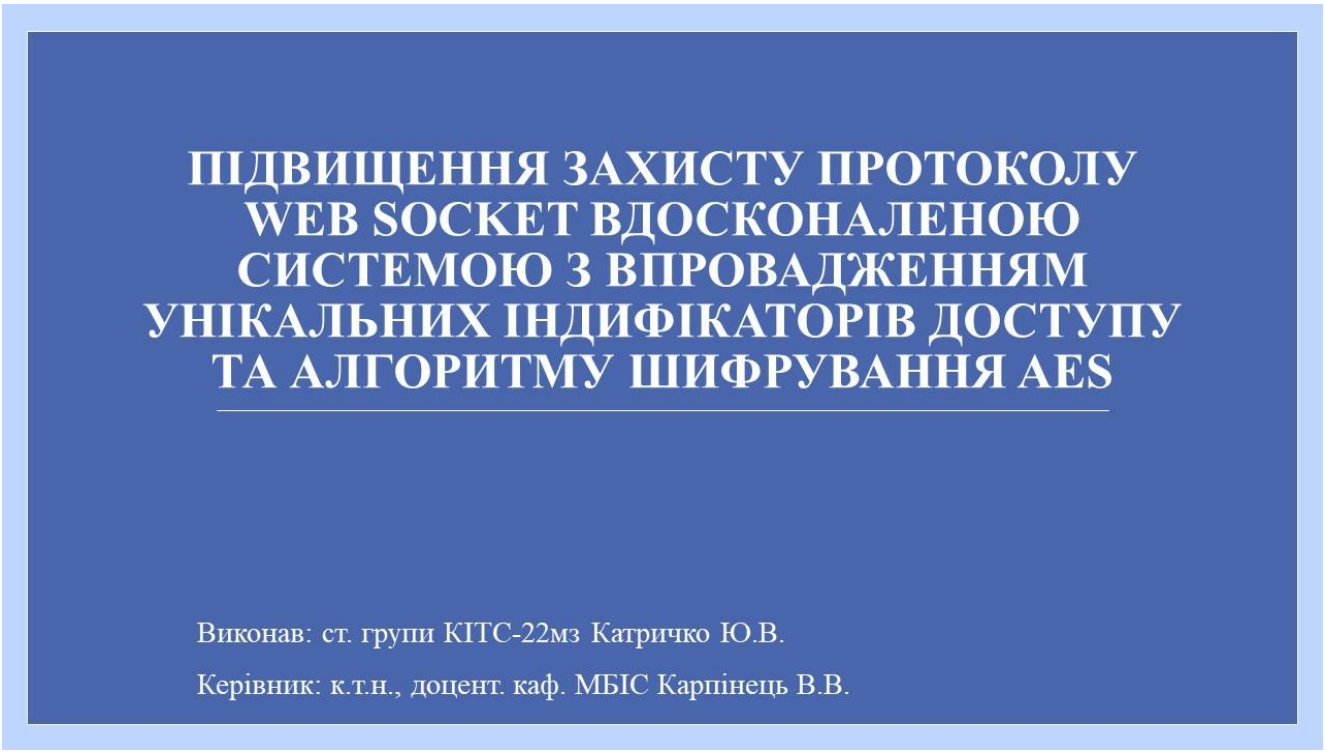
```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
@Injectable()
export class AuthService {
 constructor(private readonly jwtService: JwtService) {}
 async generateToken(user: any) {
  const payload = { username: user.username, sub: user.userId };
  return {
   access_token: this.jwtService.sign(payload),import { PassportStrategy } from
'@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { Injectable } from '@nestjs/common';
import { jwtConstants } from './constants';
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
 constructor() {
  super({
   jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
   ignoreExpiration: false,
   secretOrKey: jwtConstants.secret,
 async validate(payload: any) {
  return { userId: payload.sub, username: payload.username };
@Controller('protected')

import { WebSocketGateway, WebSocketServer, SubscribeMessage, MessageBody,
ConnectedSocket } from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';

@WebSocketGateway()
export class ChatGateway {
 @WebSocketServer()
 server: Server;

 @SubscribeMessage('message')
 handleMessage(@MessageBody() data: string, @ConnectedSocket() client: Socket): void {
  // Обробка вхідного повідомлення
  console.log(`Received message: ${data}`);
  // Відправка відповіді всім підключеним клієнтам
  this.server.emit('message', data);
 }
}
import { Injectable, NestMiddleware } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Socket } from 'socket.io';

@Injectable()
export class WsJwtMiddleware implements NestMiddleware {
 constructor(private readonly jwtService: JwtService) {}

 use(socket: Socket, next: (err?: any) => void) {
  const token = socket.handshake.query.token;
  if (!token) {
   return next(new Error('Authentication error'));
  }

  try {
   const payload = this.jwtService.verify(token);
```

```
    socket.data.user = payload;
    next();
   } catch (err) {
    next(new Error('Authentication error'));
   }
  }
}
import { Module, MiddlewareConsumer, NestModule } from '@nestjs/common';
import { ChatGateway } from './chat.gateway';
import { JwtModule } from '@nestjs/jwt';
import { WsJwtMiddleware } from './ws-jwt.middleware';
@Module({
 imports: [
  JwtModule.register({
   secret: 'secretKey', // Використовуйте ваш секретний ключ
   signOptions: { expiresIn: '60s' },
  }),
 ],
 providers: [ChatGateway, WsJwtMiddleware],
})
export class ChatModule implements NestModule {
 configure(consumer: MiddlewareConsumer) {
  consumer.apply(WsJwtMiddleware).forRoutes(ChatGateway);
 }
}
import * as crypto from 'crypto';
function encrypt(text: string, secretKey: string): string {
 const iv = crypto.randomBytes(16);
 const cipher = crypto.createCipheriv('aes-256-cbc', Buffer.from(secretKey), iv);
 let encrypted = cipher.update(text);
 encrypted = Buffer.concat([encrypted, cipher.final()]);
 return iv.toString('hex') + ':' + encrypted.toString('hex');
}

function decrypt(text: string, secretKey: string): string {
 const textParts = text.split(':');
 const iv = Buffer.from(textParts.shift(), 'hex');
 const encryptedText = Buffer.from(textParts.join(':'), 'hex');
 const decipher = crypto.createDecipheriv('aes-256-cbc', Buffer.from(secretKey), iv);
 let decrypted = decipher.update(encryptedText);
 decrypted = Buffer.concat([decrypted, decipher.final()]);
 return decrypted.toString();
}
@WebSocketGateway()
export class ChatGateway {
 @WebSocketServer()
 server: Server;
 private readonly secretKey = 'your-secret-key'; // Використовуйте ваш секретний ключ

 @SubscribeMessage('message')
 handleMessage(@MessageBody() data: string, @ConnectedSocket() client: Socket): void {
  const decryptedMessage = decrypt(data, this.secretKey);
  console.log(`Received decrypted message: ${decryptedMessage}`);

  const encryptedMessage = encrypt(decryptedMessage, this.secretKey);
  this.server.emit('message', encryptedMessage);
 }
}
```

```
@WebSocketGateway()
export class ChatGateway {
 @WebSocketServer()
 server: Server;
 private readonly secretKey = 'your-secret-key'; // Використовуйте ваш секретний ключ
 private clients = new Map<string, Socket>()
 handleConnection(client: Socket) {
  const user = client.data.user;
  this.clients.set(user.userId, client);
  console.log(`Client connected: ${user.username}`);
 }
 handleDisconnect(client: Socket) {
  const user = client.data.user;
  this.clients.delete(user.userId);
  console.log(`Client disconnected: ${user.username}`);
 }
 @SubscribeMessage('message')
 handleMessage(@MessageBody() data: string, @ConnectedSocket() client: Socket): void {
  const decryptedMessage = decrypt(data, this.secretKey);
  console.log(`Received decrypted message: ${decryptedMessage}`);

  const encryptedMessage = encrypt(decryptedMessage, this.secretKey);
  this.server.emit('message', encryptedMessage);
 }
}
import { WebSocketGateway, WebSocketServer, SubscribeMessage, OnGatewayConnection,
OnGatewayDisconnect, ConnectedSocket } from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
@WebSocketGateway()
export class ChatGateway implements OnGatewayConnection, OnGatewayDisconnect {
 @WebSocketServer()
 server: Server;
 private clients = new Map<string, Socket>();

 handleConnection(client: Socket) {
  const uniqueIdentifier = generateUniqueIdentifier();
  client.data.uniqueIdentifier = uniqueIdentifier;
  this.clients.set(uniqueIdentifier, client);
  console.log(`Client connected with ID: ${uniqueIdentifier}`);
 }

 handleDisconnect(client: Socket) {
  const uniqueIdentifier = client.data.uniqueIdentifier;
  this.clients.delete(uniqueIdentifier);
  console.log(`Client disconnected with ID: ${uniqueIdentifier}`);
 }
 @SubscribeMessage('message')
 handleMessage(@ConnectedSocket() client: Socket, @MessageBody() data: string): void {
  const uniqueIdentifier = client.data.uniqueIdentifier;
  console.log(`Received message from client with ID: ${uniqueIdentifier}`);
  // Обробка повідомлення
  this.server.emit('message', data);
 }
handleConnection(client: Socket) {
 const uniqueIdentifier = generateUniqueIdentifier();
 client.data.uniqueIdentifier = uniqueIdentifier;
 this.clients.set(uniqueIdentifier, client);
 console.log(`Client connected with ID: ${uniqueIdentifier}`);
```

```
  // Планування ротації ідентифікатора через 1 годину
  setTimeout(() => {
    const newIdentifier = generateUniqueIdentifier();
    this.clients.delete(uniqueIdentifier);
    client.data.uniqueIdentifier = newIdentifier;
    this.clients.set(newIdentifier, client);
import { Test, TestingModule } from '@nestjs/testing';
import { AuthService } from './auth.service';
import { JwtService } from '@nestjs/jwt';

describe('AuthService', () => {
  let service: AuthService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [AuthService, JwtService],
    }).compile();

    service = module.get<AuthService>(AuthService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });

  it('should generate a valid token', async () => {
    const token = await service.generateToken({ username: 'test', userId: 1 });
    expect(token).toHaveProperty('access_token');
  });
});
import { Test, TestingModule } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
import * as request from 'supertest';
import { AppModule } from './../src/app.module';
import { JwtService } from '@nestjs/jwt';

describe('WebSocketGateway (e2e)', () => {
  let app: INestApplication;
  let jwtService: JwtService;

  beforeAll(async () => {
    const moduleFixture: TestingModule = await Test.createTestingModule({
      imports: [AppModule],
    }).compile();

    app = moduleFixture.createNestApplication();
    jwtService = moduleFixture.get<JwtService>(JwtService);
    await app.init();
  });

  it('should connect and authenticate with valid token', (done) => {
    const token = jwtService.sign({ username: 'test', userId: 1 });
    const client = request(app.getHttpServer()).ws('/ws').query({ token });

    client.on('open', () => {
      client.send('message', 'Hello World');
    });
```

Додаток В. Ілюстративний матеріал



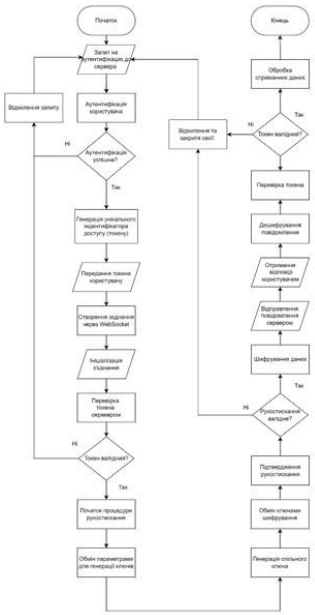
Актуальність та мета

WebSocket

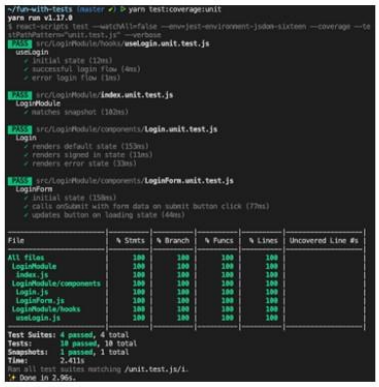
"WebSocket" - це протокол зв'язку, який дозволяє веб-додаткам здійснювати інтерактивну двосторонню комунікацію в реальному часі. Він забезпечує стійке з'єднання між клієнтом і сервером, що дозволяє передавати дані без зайвих затримок та перез'єднань, що часто відбувається з HTTP. WebSocket здатний обробляти велику кількість одночасних підключень та може використовуватися для різноманітних цілей, від онлайн-чатів до потокової передачі даних.

Схема роботи WebSocket

Алгоритм підвищення захисту протоколу web socket



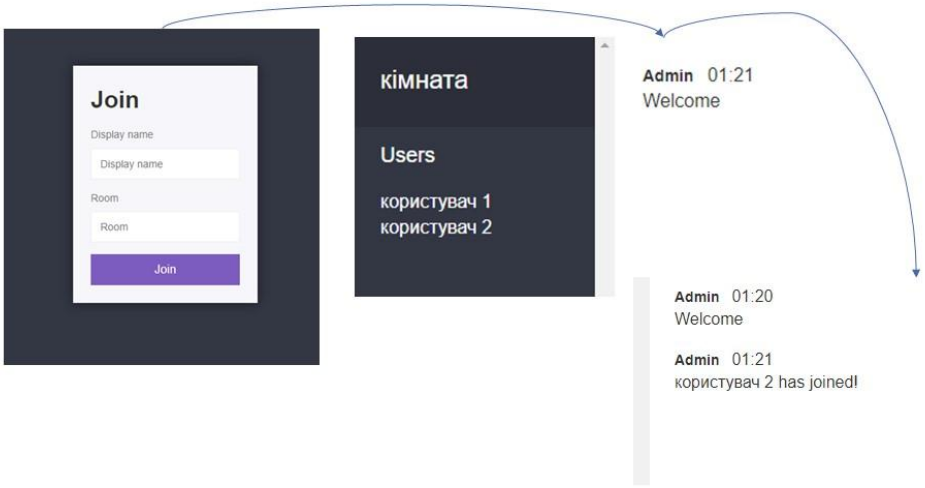
Автоматичне тестування системи



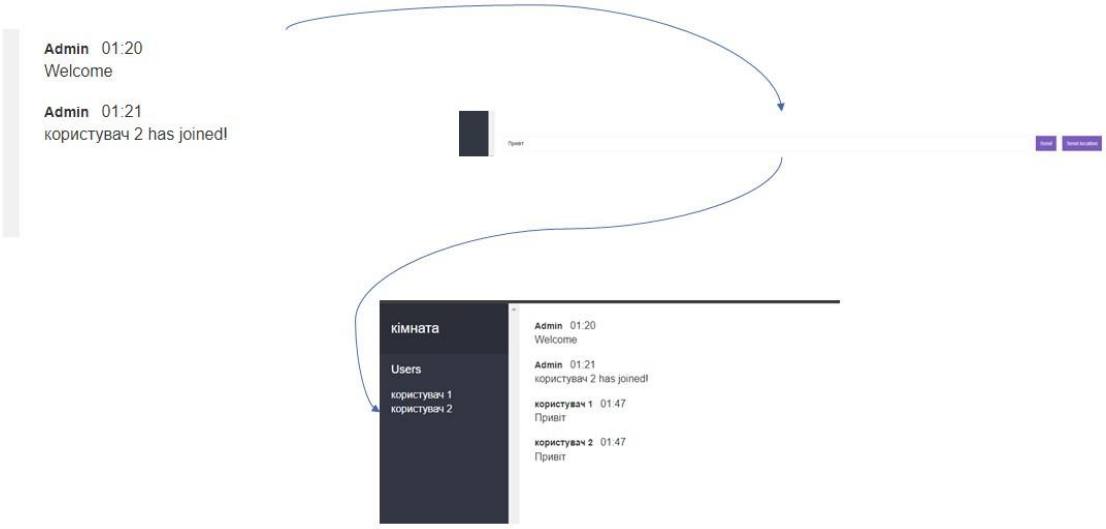
Результати тестування

Як видно з рисунку, що всі тести пройшли успішно, що свідчить про вірну роботу системи.

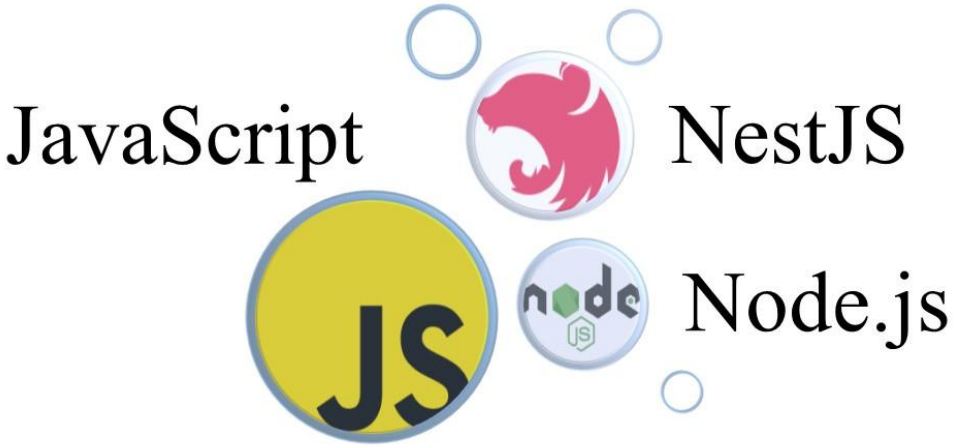
Інтерфейс додатку



Інтерфейс додатку



Використані технології



ВИСНОВОК

• Впровадження унікальних ідентифікаторів доступу та алгоритму шифрування AES в протокол web socket є кроком до покращення безпеки веб-застосунків. Ці заходи відкривають нові можливості для захисту важливих даних, переданих через мережу, від потенційних загроз та атак. Використання унікальних ідентифікаторів дозволяє ефективно ідентифікувати та автентифікувати користувачів, зменшуючи ризики несанкціонованого доступу. Шифрування AES забезпечує конфіденційність даних, пересиланих через мережу, що робить їх недоступними для неповноважних осіб.

• Цей підхід не лише забезпечує високий рівень безпеки, але й сприяє покращенню довіри користувачів до веб-застосунків, що може позитивно вплинути на їх прийняття та використання. Застосування цих заходів може бути особливо важливим у сферах, де безпека даних є критичною, таких як фінансові послуги, медична індустрія та комунікаційні технології. Таким чином, вдосконалення захисту протоколу web socket сприяє забезпеченню безпеки та конфіденційності відправлених даних, що є важливим елементом в сучасному цифровому світі.

Дякую за увагу!

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Підвищення захисту протоколу web socket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

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

Показники звіту подібності Unicheck

Оригінальність 96 % Схожість 4 %

Аналіз звіту подібності (відмітити потрібне):

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

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

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

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

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

Автор роботи _____ (підпис) Катричко Ю.В. (прізвище, ініціали)

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