НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: Авторизація в розподілених системах

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-01
(шифр групи)

Струкало Валентин Володимирович
(прізвище, ім'я, по батькові) (підпис)

Керівник: доцент кафедри ІБ к.т.н. Родіонов А.М.
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент доцент кафедри ММАД, к.т.н. Лавренюк А.М.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«___»_____2024 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Струкалу Валентину Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Авторизація в розподілених система,
керівник роботи к. т. н., доцент кафедри ІБ, Родіонов Андрій Миколайович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету «31» травня 2024 р. № √2251-с

2. Термін подання здобувачем вищої освіти роботи 10 червня 2024 р.

3. Вихідні дані до роботи Власні дослідження та література пов'язана з авторизацією, з токенами доступу.

4. Зміст роботи Огляд різних методів авторизації в розподілених системах, дослідження авторизації за токенами JWT та їх аналіз я безпечного чи небезпечного варіанту авторизації.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Дата видачі завдання 20.09.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення з темою дипломної роботи	20.12.2023 – 31.01.2024	виконано
2	Визначення цільових задач дипломної роботи	01.01.2024 – 12.01.2024	виконано
3	Узгодження теми	15.01.2024	виконано

4	Збір матеріалів	16.01.2024 – 30.01.2024	виконано
5	Ознайомлення з вимогами оформлення дипломної роботи	31.01.2024 – 02.02.2024	виконано
6	Аналіз сучасних методів авторизації	05.02.2024 – 09.02.2024	виконано
7	Вивчення розподілених систем	12.02.2024 – 23.02.2024	виконано
8	Дослідження методів авторизації	26.02.2024 – 08.03.2024	виконано
9	Визначення актуальності використання JWT	11.03.2024 – 15.03.2024	виконано
10	Глибинний аналіз механізмів роботи JWT token	18.03.2024 – 05.04.2024	виконано
11	Дослідження вразливостей JWT	08.04.2024 – 19.04.2024	виконано
12	Розробка методів вирішення недоліків JWT	22.04.2024 – 17.05.2024	виконано
13	Аналіз виконаної роботи, формування висновків	20.05.2024 – 28.05.2024	виконано
14	Оформлення дипломної роботи	29.05.2024 – 05.06.2024	виконано

Здобувач вищої освіти _____ Валентин СТРУКАЛО
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Андрій РОДІОНОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота має обсяг 67 сторінок, 32 ілюстрацій, 1 таблиці, 1 додаток та 14 літературних джерел.

Об’єкт дослідження: Авторизація на основі токенів JWT.

Предмет дослідження: Вразливості JWT.

Мета дослідження: Розробка та покращення безпечного методу авторизації на основі JWT.

Методи дослідження: аналіз(аналіз джерел з теоретичним матеріалом, який стосується авторизації в розподілених системах та авторизації на основі токенів JWT), розробка коду для покращення захисту від атак на JWT, порівняння розроблених методів зі стандартними.

Ключові слова: авторизація, token, jwt, header, payload.

ABSTRACT

The volume of the thesis is 67 pages, 32 illustrations, 1 tables, 1 appendix and 14 sources of literature.

Object of research: JWT-Based Authorization.

Subject of research: JWT Vulnerabilities.

Purpose of the study: Development and Improvement of a Secure JWT-Based Authorization Method.

Research methods: Analysis (reviewing sources with theoretical material related to authorization in distributed systems and JWT-based authorization, developing code to improve protection against JWT attacks, comparing the developed methods with standard ones).

Keywords: authorization, token, jwt, header, payload.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JWT – JSON Web Tokens

JSON – JavaScript Object Notation

XML – eXtensible Markup Language

XSS – Cross-Site Scripting

CSRF – Cross-site request forgery

CRM – Customer Relationship Management

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

ВСТУП

Наше сьогодення тісно пов’язане з технологіями і сучасний цифровий світ, де розподілені системи інформаційних технологій займають важливу роль. А саме вирішення таких завдань як безпека та авторизація з кожним днем все актуальніші та складніші, адже кожного дня світ розвивається та оновлюється. Авторизація, як процес перевірки ідентичності користувача, набуває особливого значення у забезпеченні конфіденційності та цілісності інформації.

У цій роботі буде досліджено сучасні методи авторизації в розподілених системах. Особлива увага буде приділена аналізу, порівнянню різних методів, підходів до авторизації та їхня ефективність, зручність, швидкість, а головне безпека.

Сьогодні дуже багато різних варіантів авторизації використовуючи різні форми токенів, який дозволяє ідентифікувати користувача.

У цьому контексті, методи авторизації, які базуються на технології токенів, стають все більш популярними та важливими. Один із найбільш відомих та широко використовуваних стандартів токенізації - JSON Web Token (JWT) - отримує все більше уваги в індустрії інформаційної безпеки та розподілених систем.

Однак, з популярністю JWT виникають нові виклики та питання, пов'язані з його безпекою та ефективністю.

Мета роботи: Розробка та покращення безпечного методу авторизації на основі JWT.

Завдання роботи:

- Аналіз авторизації за допомогою JWT;
- Огляд вразливостей, які можливі за даної авторизації;

- Розробка модельного прикладу авторизації з використанням JWT;
- Розробка механізмів покращення безпеки доступу;
- Підсумок запропонованих методів покращення.

Об’єкт дослідження: Авторизація на основі токенів JWT.

Предмет дослідження: Вразливості JWT.

Актуальність: На сьогоднішній день розподілені системи є більш зручними та популярними за централізовані. Проте зростання популярності таких систем також збільшує ризики несанкціонованого доступу до даних та ресурсів. Тому розробка методів для забезпечення безпеки авторизації в розподілених системах є важливим напрямком досліджень. Впровадження ефективних методів авторизації дозволить знизити ризики несанкціонованих доступів, забезпечуючи тим самим вищу надійність і захищеність систем.

1 Розподіленні системи та методи авторизації

1.1 Поняття про розподілену систему

Поняття «розподілена система» в літературних джерелах визначають по-різному. Історично у процесі розвитку апаратної та програмної складових розподілених систем формувалися різні розуміння їх як систем, що відображалося на їх визначеннях. Розуміння того, як має визначатися розподілена система, постійно змінюється. [1]

Розподілена система – це набір незалежних комп'ютерів, які користувач сприймає як єдину об'єднану систему. [1]

Розподіленою називають систему з просторово розподіленими компонентами, які не використовують ніякої спільної пам'яті й не підлягають децентралізованому адмініструванню. Для реалізації спільних цілей можлива кооперація компонентів. Якщо цими компонентами пропонуються послуги або використовуються запропоновані послуги, то виникає клієнт-серверна система. [1]

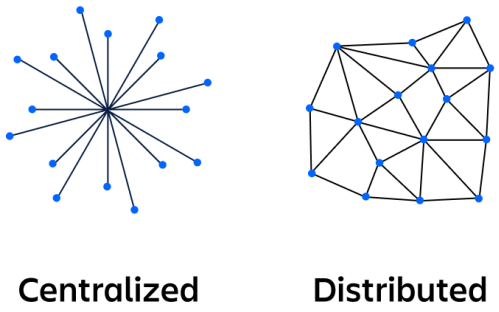

Рисунок 1.1 – Різниця між централізованою та розподіленою системою

1.2 Модель клієнт-сервер

Терміни «клієнт» і «сервер» використовуються не лише для позначення програмних модулів, але й комп'ютерів або пристроїв, підключених до мережі. Комп'ютер, який надає свої ресурси іншим комп'ютерам у мережі, називається сервером, а той, що споживає ці ресурси, – клієнтом. Іноді один і той самий комп'ютер може виконувати функції як сервера, так і клієнта одночасно. Мережні служби завжди є розподіленими програмами загального призначення, які складаються з кількох частин, що взаємодіють між собою, причому кожна частина зазвичай реалізується на окремому комп'ютері в мережі. Це дозволяє виконувати також розподілені програми користувачів. Розподілена прикладна програма складається з кількох компонентів, кожен з яких виконує певну завершену частину завдання. [1]

Наприклад, одна частина програми, що працює на комп'ютері користувача, може забезпечувати спеціалізований графічний інтерфейс; інша частина, що працює на потужному віддаленому комп'ютері, може здійснювати статистичну обробку даних, введених користувачем; а третя частина може заносити отримані результати в базу даних на комп'ютері зі встановленою стандартною СУБД. [1]

У базовій моделі клієнт-сервер усі процеси в розподілених системах поділяють на дві групи. Процеси, які прямо реалізують деяку службу, наприклад службу файлової системи або бази даних, є серверами (servers). Процеси, які вимагають служби у серверів через надсилання запиту і подальшого очікування відповіді від сервера, – клієнти (clients). Взаємодію між клієнтом та сервером називають режимом запит-відповідь (request-reply behavior).

Розглядаючи велику кількість прикладних програм типу клієнтсервер, передбачених для організації доступу користувачів до баз даних, часто рекомендують поділяти їх на три рівні:

- рівень інтерфейсу користувача;

- рівень обробки;

- рівень даних.

Рівень користувацького інтерфейсу містить усе необхідне для безпосереднього спілкування з користувачем, наприклад, для керування дисплеєм. **До рівня обробки** належить прикладне програмне забезпечення, а **до рівня даних** –дані, з якими доводиться працювати. [1]

1.3 Особливості розподілених систем

Існує низка причин, через які централізовані системи замінюють на розподілені:

- **Масштабованість**: Розподілені системи легко масштабуються, що дозволяє реалізувати нові вимоги інформаційної системи при розширенні підприємства.
- **Інтеграція рішень**: Вони можуть інтегруватися з наявними системними компонентами без необхідності створювати нову систему з новою функціональністю.
- **Мінімізація ризиків**: Поступове системне розширення мінімізує ризик перевантаження окремих компонентів.
- **Ефективне керування**: Організаційне керування потужністю розподіленої системи забезпечує ефективну з точки зору вартості реалізацію, при цьому система залишається гнучкою та легко адаптується.
- **Автономне керування**: Власник ресурсу може самостійно керувати компонентом і вільно втручатися та реконфігурувати систему відповідно до своїх потреб.

- **Автономність компонентів**: Окремі частини розподіленої системи повністю автономні, тому у разі помилки або виходу з ладу одного з компонентів інші продовжують працювати без перерв.
- **Спільне використання ресурсів**: Розподілені системи дозволяють спільно використовувати дані та пристрої, створюючи ілюзію їх безпосереднього під’єднання до комп’ютера кожного користувача, що сприяє гнучкому розподілу роботи.
- **Оперативний доступ**: Вони забезпечують користувачам оперативний доступ до великої кількості корпоративної інформації незалежно від їхнього місця перебування.

1.4 Авторизація. Основні відомості

Авторизація – це те, що користувачу дозволяється робити після входу. Це надання і перевірка прав на вчинення будь-яких дій в системі. [2]

Авторизація визначає рівень і тип доступу до ресурсів, які має користувач. Це відповідає на запитання, хто що може робити з вашими даними та програмами. Після автентифікації користувача за допомогою його облікових даних, таких як ім’я користувача та пароль, його авторизація визначатиме попередньо визначене меню операційних систем, програм, функціональних можливостей, а також рівень і можливість вносити зміни до основних даних. [2]

Авторизація не надає доступ до кожного ресурсу. Натомість різні користувачі мають доступ до різних ресурсів. Компанії використовують різні процеси, щоб визначити, які співробітники можуть отримати доступ до ресурсу на основі того, що їм потрібно для виконання своєї роботи. Обмежуючи доступ до певних матеріалів, підприємства мають спосіб захистити свої найважливіші дані, включаючи інтелектуальну власність, ідентифікаційні дані споживачів, інформацію про заробітну плату і тому подібне. Крім того, це може бути інструментом, який допоможе

співробітникам швидше знаходити необхідні ресурси та інформацію, не переглядаючи кожен документ і папку компанії. Як правило, організації дотримуються принципу найменших повноважень, надаючи доступ до інформації. Цей підхід гарантує, що користувачі та співробітники мають доступ лише до інформації, яка потрібна для їх ролі, і не більше того. Шкода, яку може заподіяти потенційний зловмисник, зведена до мінімуму. [2]

Авторизація дозволяє доступ до певних програм і інформації, за часту, за допомогою імені користувача та пароля. Користувач може отримати доступ до текстового процесора, електронної пошти, CRM тощо. З базовою авторизацією користувачі матимуть окремий ідентифікатор користувача та пароль для кожної системи. Наприклад, співробітникам може знадобитися один вхід для CRM, інший для електронної пошти, інший для доступу до сервера тощо. Коли їм потрібен доступ до нових систем або інформації, вони надсилають запити адміністратору, який розглядає запит і надає додаткові облікові дані для входу. [2]

Тобто, авторизація – це надання прав, привілеїв на доступ до ресурсу. На рисунку нижче продемонстровано простий приклад того, що таке авторизація:

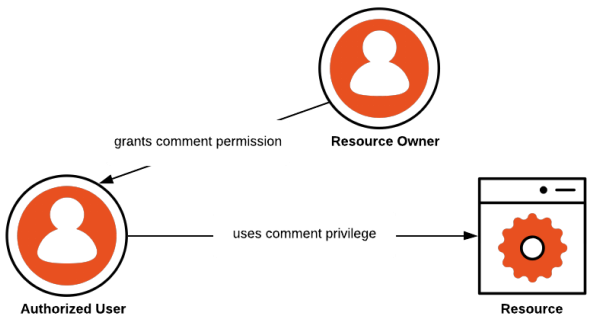

Рисунок 1.2 – Приклад авторизації

1.5 Різниця між автентифікацією та авторизацією

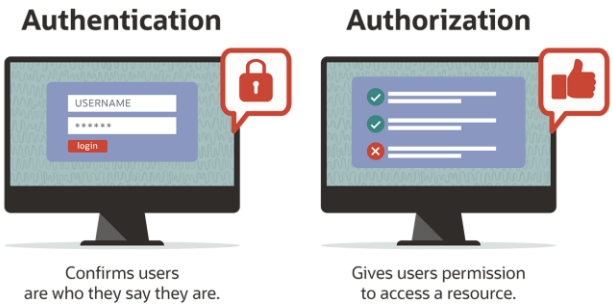

Рисунок 1.3 – Різниця між автентифікацією авторизацією

Автентифікація та авторизація часто використовуються як взаємозамінні, але є деякі важливі відмінності. [3]

Автентифікація — це спосіб для користувачів підтвердити, що вони є тими, за кого себе видають. Це можна зробити за допомогою імен користувачів і паролів. І все більшої популярності набуває двох факторна автентифікація. Для цього потрібні не лише ім’я користувача та пароль, а й ще один рівень безпеки через інший пристрій, найчастіше через телефонну програму або текстове повідомлення з кодом на телефон користувача. Авторизація слідує за автентифікацією. Те, що користувачі автентифіковані, це не означає, що вони отримують доступ до всіх програм, послуг і даних у корпоративній мережі. Натомість вони отримують доступ лише до тих ресурсів, які їм дозволено використовувати. [3]

Два етапи контролю доступу, авторизація та автентифікація, необхідні для взаємодії користувача з корпоративною мережею. Якщо користувачі не автентифіковані, вони не можуть користуватися жодними службами, оскільки не можуть увійти в мережу. Якщо користувач не авторизований, він не зможе користуватися жодними службами, навіть якщо його автентифіковано. [3]

Ці два поняття є залежними один від одного і вони працюють разом, йдучи один за одним для підтвердження користувача та надання йому відповідних прав доступу до ресурсу тощо.

Авторизація не лише перевіряє, чи має користувач відповідні права для виконання конкретних дій, але й встановлює ці права не лише при вході до системи, але й під час будь-якої спроби виконати маніпуляції з даними.

Автентифікація та авторизація відрізняються тим, що перша є одноразовою процедурою для поточного сеансу, тоді як остання є постійним процесом, який відбувається перед початком будь-якої операції. Під час автентифікації користувач надає свої ідентифікаційні дані, такі як логін/пароль, відбиток пальця, сертифікат або PIN-код картки. У разі успішної автентифікації авторизація відбувається автоматично на сервері, і дії користувача на цей процес не впливають. [3]

1.6 Способи автентифікації веб-програм

- Автентифікація на основі файлів cookie;
- Доступ сторонніх розробників (API token, OAuth);
- SAML;
- Автентифікація за допомогою веб-токенів.

1.6.1 Автентифікація на основі файлів cookie

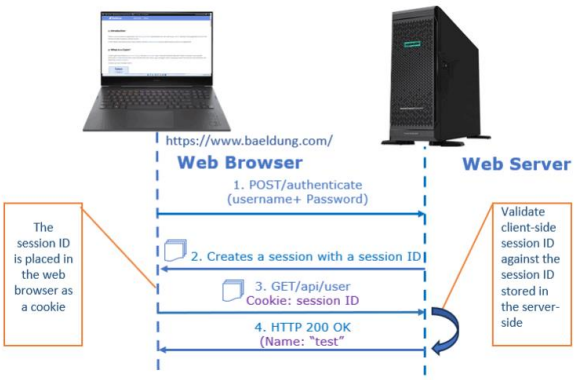

Рисунок 1.4 - Автентифікація на основі файлів cookie

Принцип роботи файлів cookie [4]:

- Використовуючи свої дані для входу, користувач намагається увійти;
- Сервер створює сеанс із ідентифікатором сеансу після перевірки користувача за допомогою наданих облікових даних;
- Веб-браузер (клієнт) зберігає ідентифікатор сеансу як файл cookie;
- При наступних запитах ідентифікатори сесії на стороні клієнта та сервера порівнюються, і якщо вони збігаються, запит приймається;
- Крім того, сеанс завершується як на стороні клієнта, так і на стороні сервера, якщо користувач виходить із програми.

Переваги cookies [4]:

- Легко використовувати та застосовувати: Cookies - це простий і зрозумілий механізм, який легко реалізувати як на стороні клієнта, так і на стороні сервера.
- Мають можливість скасовувати свою дійсність: Cookies можна налаштувати на автоматичне скасування терміну дії через певний проміжок часу, що робить їх більш безпечними, коли користувач виходить із системи або закриває браузер.
- Зберігають мало місця: Cookies зазвичай мають невеликий розмір, що робить їх мінімальним навантаженням на пропускну здатність мережі та час завантаження сторінки.

Недоліки cookies [4]:

- Вразливі до атак XSS та CSRF: Cookies можуть бути скомпрометовані за допомогою атак XSS та CSRF, що може дозволити зловмисникам викрасти дані користувачів або сеанси автентифікації.
- Мають обмеження за розміром та кількістю: Браузери мають обмеження щодо того, скільки cookies можна зберігати та скільки місця вони можуть займати. Це може призвести до проблем, якщо веб-сайт використовує багато cookies або якщо cookies мають великий розмір.

- Містять конфіденційну інформацію про користувача: Cookies можуть містити конфіденційну інформацію про користувача, таку як ідентифікатор користувача, налаштування або дані сеансу. Це робить їх мішенню для зловмисників, які можуть намагатися викрасти цю інформацію.
- Масштабування стає проблемою, коли багато користувачів входять у систему: Якщо веб-сайт має багато користувачів, які одночасно входять у систему, використання cookies може призвести до проблем з масштабуванням. Це пов'язано з тим, що серверу потрібно зберігати та обробляти велику кількість cookies.

1.6.2 Доступ сторонніх розробників (API token, OAuth)

Api token

Токен API, часто називають токеном доступу, автентифікації або авторизації, є унікальним ідентифікатором, який надає обмежений доступ до певних ресурсів і послуг, наданих через API додатка. Ці токени відіграють вирішальну роль у забезпеченні безпеки та збереженні конфіденційності користувачів у сфері веб-, мобільних і серверних додатків, особливо при доступі до даних або виконанні дій всередині додатка. [4]

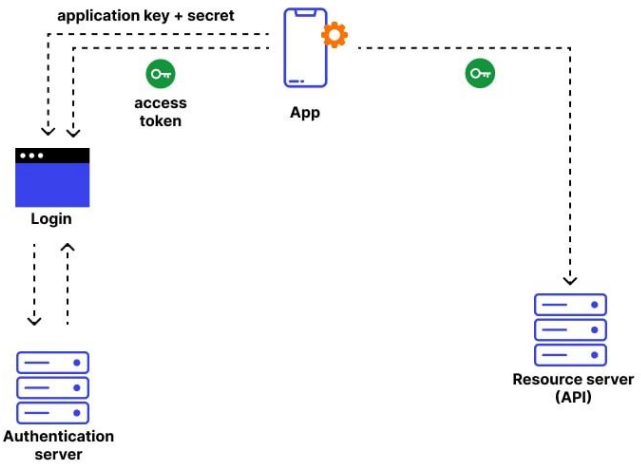

Рисунок 1.5 – Принцип Api token

Переваги:

- Прості та гнучкі, можуть мати різні формати та властивості залежно від реалізації та вимог безпеки

Недоліки:

- Вразливі до перехоплення або крадіжки, якщо не передаються через HTTPS або не зберігаються безпечно, не надають жодної інформації про користувача або програму, яка їх використовує, ускладнюють аудит або ведення журналу

OAuth

OAuth – це відкритий стандартний протокол або фреймворк авторизації, який надає додаткам можливість "безпечного визначеного доступу". Наприклад, ви можете повідомити Facebook, що ExampleWeb.com може отримати доступ до вашого профілю або публікувати оновлення на вашу стіну, не передаючи пароль від Facebook ExampleWeb. Це значно зменшує ризик: у разі порушення безпеки ExampleWeb ваш пароль від Facebook залишається безпечним.

OAuth не передає дані пароля, а використовує токени авторизації для підтвердження ідентичності між споживачами і постачальниками послуг. OAuth - це протокол автентифікації, який дозволяє затверджувати одному додатку взаємодіяти з іншим від вашого імені, не видаючи ваш пароль.

Переваги:

- Дозволяє користувачеві надавати доступ до своїх ресурсів або даних на одному сайті іншому сайту без розкриття своїх облікових даних, може використовувати різні типи токенів, такі як JWT, API-токені.

Недоліки:

- Складний і вимагає множинних сторін і взаємодій, вносить деякі ризики для безпеки, такі як фішингові атаки, витік токенів або повторне відтворення токенів.

1.6.3 SALM

SAML (Security Assertion Markup Language) базується на XML та є більш універсальним. Використовуючи URL ідентифікатора постачальника послуг (IDP), користувач може увійти в систему, і система перенаправить його з XML-даними назад на сторінку додатку. Крім того, для отримання інформації про користувача потрібно розкодувати XML. SAML також пропонує механізм для впровадження одноразового входу.

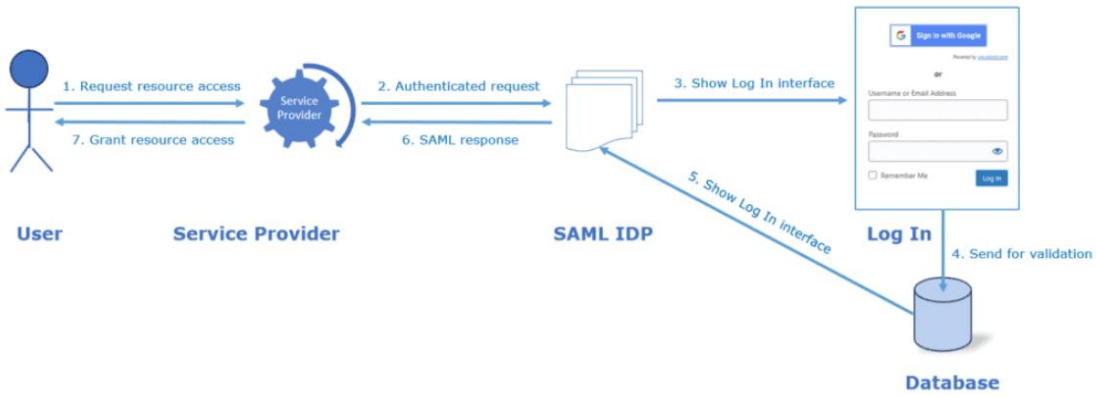

Рисунок 1.6 – Схема роботи SAML

Спочатку служба ідентифікації надає постачальнику послуг власні облікові дані користувача. Далі, кожен кінцевий користувач повинен увійти за допомогою SSO лише один раз, щоб параметри SAML можна було передавати безперервно постачальникам послуг. Крім того, постачальник послуг звертається до служб ідентифікації для перевірки автентичності запиту. Під

час цього процесу також потрібно надати дозвіл на налаштування SAML SSO. Цей процес гарантує автентифікацію та авторизацію користувача. [4]

Переваги:

- Дозволяє користувачеві входити на один сайт і отримувати доступ до іншого сайту без повторного введення своїх облікових даних, використовує твердження, які містять інформацію про ідентичність, атрибути або рішення про авторизацію користувача, можуть бути підписані та зашифровані постачальником послуг ідентифікації та перевірені постачальником послуг.

Недоліки:

- Складний і вимагає обробки та аналізу XML, має певні проблеми з продуктивністю через розмір і кількість задіяних повідомлень.

1.6.4 Автентифікація за допомогою веб-токенів

Веб-токени JSON (JWT) є найпоширенішим методом автентифікації веб-додатків . Крім того, це окремі маркери, які мають імена користувачів, ролі, права тощо.

Сервер створює JWT і надсилає його клієнту з секретною інформацією. З кожним наступним запитом клієнт надсилає JWT, що зберігається на клієнті.

Потім сервер перевіряє JWT з кожним запитом, зробленим клієнтом, перш ніж відповісти. Крім того, секретний ключ для цифрового підпису JWT гарантує цілісність і автентичність веб-програми.

Переваги JWT:

- **Масштабованість сервера**: Кількість користувачів не впливає на масштабованість сервера, оскільки JWT не зберігаються на сервері.

- **Безстанність**: Веб-застосунку не потрібно зберігати жодних даних сеансу на сервері, що зменшує навантаження на сервер, покращує продуктивність та масштабованість.
- **Інформація про користувача**: JWT може містити більше інформації про користувача, ніж звичайний ідентифікатор сеансу.
- **Портативність**: JWT дозволяють автентифікацію та авторизацію між доменами та платформами.
- **Адаптивність**: JWT можуть містити будь-яку інформацію, що стосується веб-застосунку, що дозволяє точно та персоналізовано контролювати доступ.

Недоліки JWT:

- **Розмір**: Розмір JWT значно більший, ніж ідентифікатор сеансу куки.
- **Відсутність можливості скасування доступу**: JWT не може скасувати доступ користувача, навіть якщо його обліковий запис було припинено або видалено.
- **Уразливість**: Оскільки токен зберігається на стороні клієнта, він може бути вкрадений зловмисниками, які можуть використовувати його для того, щоб увійти в систему під ім'ям користувача та отримати доступ до його ресурсів.

Висновок до розділу 1

Розподілені системи мають низку переваг, які зумовлюють їхню заміну централізованими системами. Вони легко масштабуються, інтегруються з наявними компонентами, мінімізують ризик перевантаження, забезпечують ефективне керування, дозволяють автономне керування компонентами та їхню автономність, спільне використання ресурсів, і забезпечують оперативний доступ до інформації незалежно від місця перебування користувача.

Авторизація є критичним аспектом безпеки, який визначає, що користувач може робити після входу в систему. Вона забезпечує доступ до необхідних ресурсів на основі визначених прав та привілеїв, захищаючи важливі дані та підвищуючи ефективність роботи. Автентифікація підтверджує особу користувача, а авторизація визначає його права на доступ до різних ресурсів. Обидва процеси працюють разом, щоб гарантувати безпеку та належний доступ у корпоративних мережах.

2 Використання JWT для авторизації

2.1 Поняття про JWT Token

JSON Web Token (JWT) - це відкритий стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами у вигляді об'єкту JSON. Цю інформацію можна перевіряти і довіряти, оскільки вона має цифровий підпис. JWT можна підписувати за допомогою секретного ключа (за алгоритмом HMAC) або пари публічного/приватного ключа за допомогою RSA або ECDSA. [5]

JWT token використовуються для:

- **Авторизації:** це найпоширеніший сценарій використання JWT. Після входу користувача кожен наступний запит буде містити JWT, що дозволить користувачеві отримувати доступ до маршрутів, сервісів і ресурсів, які дозволені для цього токена. Одноразовий вхід - це функція, яка широко використовує JWT у наш час через його невеликий наклад і здатність легко використовуватися в різних доменах. [5]
- **Обмін інформації:** це хороший спосіб безпечної передачі інформації між сторонами. Оскільки JWT можна підписати, наприклад, використовуючи пари публічного/приватного ключа, можна бути впевненим, що відправники - ті, за кого вони себе видають. Крім того, оскільки підпис обчислюється за допомогою заголовка та навантаження, також можна перевірити, що вміст не був змінений. [5]

2.2 Структура JSON Web Token

JSON Web Tokens складається з трьох частин, розділених крапками (.), які представлені таким чином [5]:

- Header
- Payload

- Signature

Тобто вигляд jwt token має наступний: ***xxxxx.yyyyy.zzzzz***

2.2.1 Header

Заголовок зазвичай складається з двох частин: типу токена, яким є JWT, і використовуваного алгоритму підпису, такого як HMAC SHA256 або RSA.

Приклад заголовку:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Потім цей JSON кодується у форматі Base64Url, утворюючи першу частину JWT.

2.2.2 Payload

Друга частина токена - це корисне навантаження, яке містить заяви(твердження). Запит - це висловлення про сутність (зазвичай, користувача) та додаткові дані цього користувача. Існує три типи заяв:

- **Зареєстровані**: це набір попередньо визначених заяв, які не є обов'язковими, але рекомендованими, щоб надати набір корисних, взаємодійних заяв. Деякі з них: iss (видавець), exp (час закінчення), sub (предмет), aud (аудиторія) та інші. [5]
- **Публічні**: заяви можуть бути визначені на власний розсуд тими, хто використовує JWT. Проте, щоб уникнути колізій, вони повинні бути визначені в реєстрі IANA JSON Web Token або бути визначені як URI, який містить простір імен, що стійкий до колізій. [5]

- **Приватні**: це спеціальні заяви, створені для обміну інформацією між сторонами, які погоджуються використовувати їх, і не є або зареєстрованими, або публічними запитами. [5]

Приклад:

```
{
  "sub": "1234567890",
  "group": "FB-01",
  "first_name": "Strukalo",
  "iat": 1516239022
}
```

Рисунок 2.1 – Приклад payload

Для підписаних токенів ця інформація, хоча й захищена від підробки, але, в свою чергу, доступна для читання будь-кому. Тому не треба розміщувати секретну інформацію в елементах корисного навантаження або заголовка JWT, якщо вони не зашифровані. [5]

Потім цей JSON кодується у форматі Base64Url, утворюючи другу частину JWT.

2.2.3 Signature

Щоб створити частину підпису, вам потрібно взяти закодований заголовок, закодоване навантаження, секрет, вказаний в заголовку алгоритм, і підписати це. [5]

Наприклад, якщо ви хочете використовувати алгоритм HMAC SHA256, підпис буде створений наступним чином:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Рисунок 2.2 – Приклад signature

Підпис використовується для перевірки того, що повідомлення не було змінено по дорозі, і, у випадку токенів, підписаних приватним ключем, він також може перевірити, що відправник JWT - той, за кого він себе видає.

Вихід складається з трьох рядків Base64-URL, розділених крапками, які можна легко передавати у HTML-та HTTP-середовищах, при цьому вони є більш компактними порівняно зі стандартами на основі XML, такими як SAML. [5]

Ось приклад JWT, який має закодований заголовок і навантаження, і підписаний секретним словом:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JfaWQiOiI2NTJkMTQ1MmZlZGJhZDRiYjhlMTFkM
2EiLCJuYW1lIjoiSm9oYSIsImlhdCI6MTY5NzQ1
MzE2MywiZXhwIjoxNjk3NTM5NTYzfQ.YfX2y7no
kk62yPTeY4E6ynszF1tjjYh9ZNEGo1bOh64
```

Рисунок 2.3 - Приклад закодованого JWT

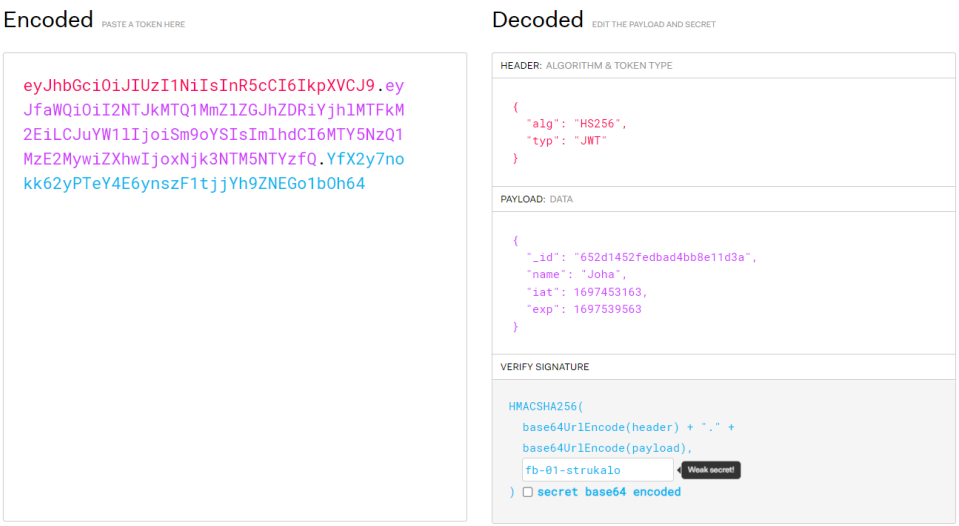

Рисунок 2.4 – Демонстрація наповнення jwt [5]

Щоразу, коли користувач хоче отримати доступ до захищеного маршруту або ресурсу, агент користувача повинен надіслати JWT, як правило, у заголовку авторизації за допомогою схеми носія . Вміст заголовка має виглядати так: **Authorization: Bearer <token>.**

Ось як це виглядає в симуляції через додаток Postman:

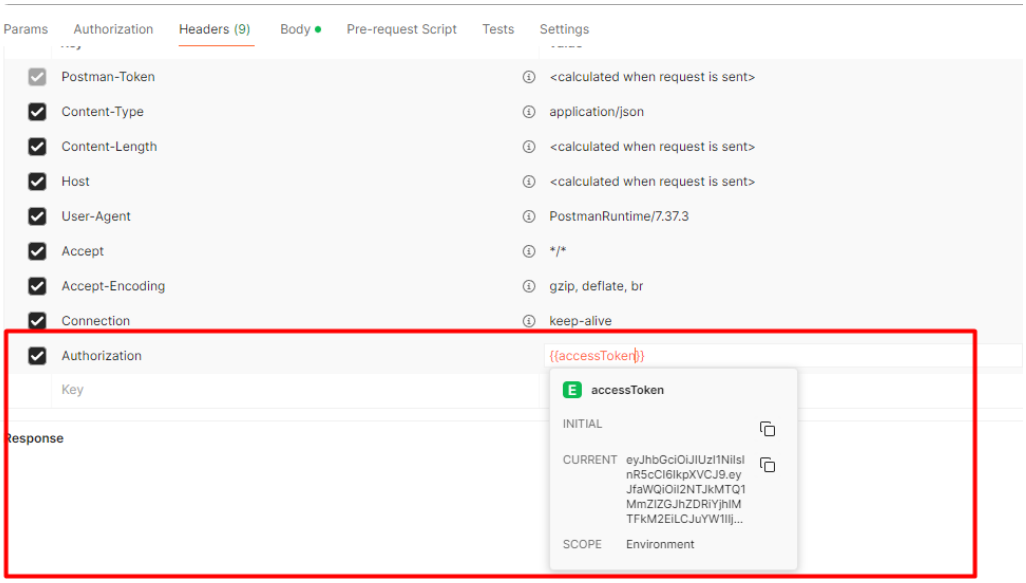

Рисунок 2.5 – Приклад заголовка в симуляції запиту

У певних випадках це може бути механізм авторизації без стану. Захищені маршрути сервера перевірятимуть наявність дійсного JWT у заголовку Authorization, і якщо він присутній, користувачу буде дозволено отримати доступ до захищених ресурсів. Якщо JWT містить необхідні дані, потреба у запитах до бази даних для деяких операцій може бути зменшена, хоча це не завжди можливо. [5]

2.3 Параметри заголовка JWT

Згідно з JWS специфікацією, лише параметр заголовка alg є обов'язковим. Проте на практиці, заголовки JWT (також відомі як JOSE заголовки) часто містять кілька інших параметрів. Ось деякі з них, які особливо цікаві для атак.

- jwk (JSON Web Key) - надає вбудований об'єкт JSON, що представляє ключ.
- jku (JSON Web Key Set URL) - надає URL, за допомогою якого сервери можуть отримати набір ключів, що містить правильний ключ.

- kid (Key ID) - надає ідентифікатор, який сервери можуть використовувати для визначення правильного ключа в тих випадках, коли є декілька ключів для вибору. Залежно від формату ключа, це може мати відповідний параметр kid.

2.4 Принцип роботи JWT token

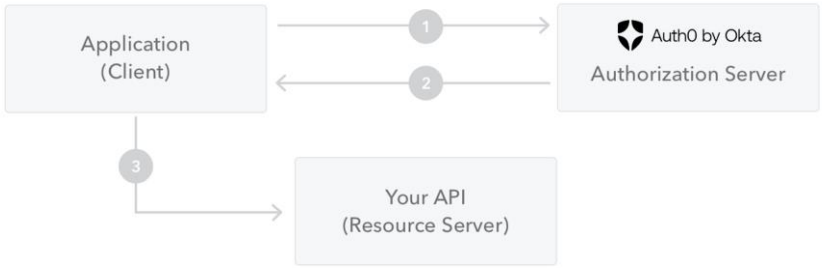

Рисунок 2.6 – Схема роботи jwt token

1. Запит додатка або клієнта на авторизацію до сервера авторизації відбувається через один із різних потоків авторизації. Наприклад, типовий веб-додаток, який відповідає вимогам OpenID Connect, пройде через кінцеву точку /oauth/authorize, використовуючи потік авторизації з кодом доступу.
2. Коли авторизація надається, сервер авторизації повертає додатку токен доступу.
3. Додаток використовує токен доступу для доступу до захищеного ресурсу, такого як API.

2.5 Алгоритми шифрування HS256, RS256

Підписи токенів створюються шляхом комбінування закодованих версій заголовка та корисного навантаження JWT, передавши їх, а також секретного значення як параметри в алгоритм, визначений у заголовку. [6]

RS256 і HS256 - найбільш поширені алгоритми, які використовуються для підпису JWT.

HS256 - це симетричний хеш-алгоритм з одним секретним ключем. Симетричний означає, що дві сторони діляться секретним ключем. Ключ використовується як для генерації підпису, так і для його перевірки. [6]

RS256 - це асиметричний алгоритм, який використовує пару публічного/приватного ключа. Постачальник ідентифікації має приватний ключ для генерації підпису. Отримувач JWT використовує публічний ключ для перевірки підпису JWT. Публічний ключ, що використовується для перевірки, і приватний ключ, який використовується для підпису токена, пов'язані між собою, оскільки вони генеруються як пара. [6]

2.5.1 RS256 чи HS256

Хоча як HS256, так і RS256 можуть бути використані для перевірки цілісності JWT, наразі рекомендованим алгоритмом є RS256.

Підпис повинен гарантувати автентичність, що означає, що вміст JWT збігається з тим, що згенеровано відправником. Як HS256, так і RS256 гарантують автентичність JWT.

RS256 також гарантує невідмовність, що означає, що він гарантує, що відправник згенерував підпис.

З RS256 можна бути впевненим, що лише власник приватного ключа і ніхто інший не може підписати токени. Крім того, якщо секретний ключ буде скомпрометований, можна обертати ключі підпису без повторного розгортання додатку з новим секретним значенням, як треба робити з HS256. [6]

HS256 можна використовувати при роботі з застарілими додатками, які не підтримують RS256. Ще один можливий випадок використання HS256

замість RS256 - це коли ваш додаток робить дуже велику кількість запитів, оскільки HS256 ефективніший за RS256. [6]

2.6 Переваги JWT Token

Розглянемо різницю між JSON Web Tokens (JWT) та Simple Web Tokens (SWT) and Security Assertion Markup Language Tokens (SAML)

- JSON менш розгорнутий, ніж XML, тому після кодування його розмір також менший, що робить JWT більш компактним порівняно з SAML. Це робить JWT хорошим вибором для передачі у HTML- та HTTP-середовищах.
- З точки зору безпеки, SWT може бути підписаний тільки симетрично за допомогою загального секретного ключа за алгоритмом HMAC. Однак токени JWT і SAML можуть використовувати пару публічного/приватного ключа у формі сертифіката X.509 для підпису. Підпис XML за допомогою XML Digital Signature без введення незрозумілих пунктів безпеки дуже складно порівняно з простотою підпису JSON.
- Парсери JSON є загальними в більшості мов програмування, оскільки вони відображаються безпосередньо на об'єкти. На відміну від цього, XML не має природного відображення документу на об'єкт. Це полегшує роботу з JWT порівняно з SAML-заявами.

Нижче наведена демонстрація розмірів JWT та SAML

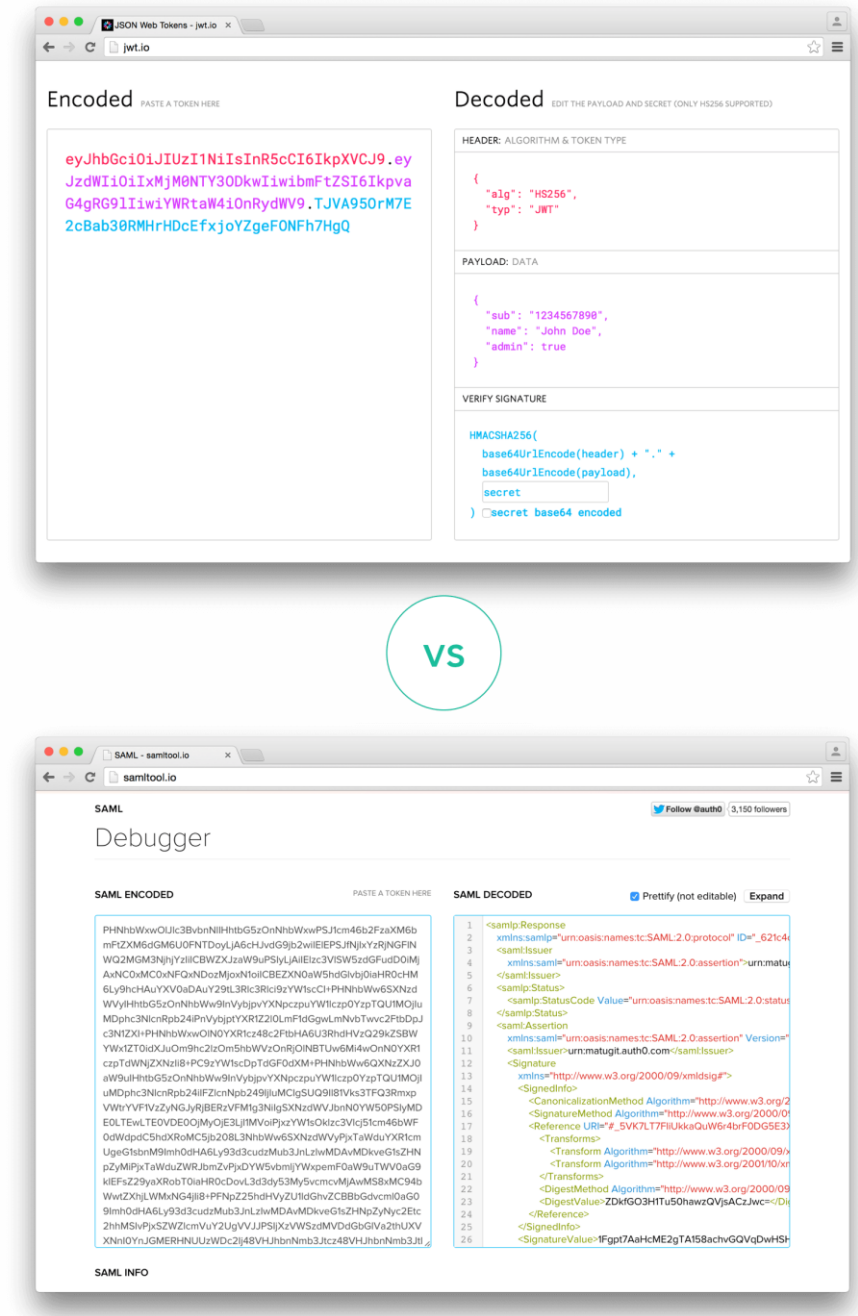

Рисунок 2.7 – Різниця розмірів JWT та SAML

2.7 Недоліки JWT Token

Найбільша проблема з JWT – це проблема відкликання токенів. Оскільки він продовжує діяти до закінчення терміну, сервер не має простого способу його відкликати.

2.7.1 Приклади, де це є проблемою

- ***Вихід із системи насправді не виходить із системи!***

Для прикладу можна взяти вихід з Twitter після твітів. Ми вважаємо, що вийшли з сервера, але це не так. Оскільки JWT є автономним і продовжуватиме працювати до закінчення терміну його дії. Це може бути 5 хвилин, 30 хвилин або будь-який інший час, встановлений для токена. Отже, якщо хтось отримає доступ до цього токена протягом цього часу, він зможе продовжувати доступ до сервера до закінчення терміну дії токена. [11]

- ***Блокування користувачів не блокує їх негайно.***

Наприклад, адміністратор якоїсь онлайн гри в реальному часі, де реальні користувачі користуються системою. Як модератор, він хочете швидко заблокувати когось за зловживання системою. Він не можете, знову ж таки з тієї ж причини. Навіть після блокування користувач продовжуватиме мати доступ до сервера до закінчення терміну дії токена. [11]

- ***Можливість застарілих даних***

Тобто, користувач є адміністратором і був понижений до звичайного користувача з меншими правами. Знову ж таки, це не набуде чинності негайно, і користувач продовжуватиме бути адміністратором до закінчення терміну дії токена. [11]

JWT часто не шифруються, тому будь-хто, хто може здійснити атаку "людина посередині" (MITM) і перехопити JWT, отримає ваші облікові дані аутентифікації. Це спрощується тим, що для здійснення MITM-атаки потрібно лише перехопити з'єднання між сервером і клієнтом.

- ***Довжина токенів***

У багатьох складних реальних додатках може знадобитися зберігати велику кількість різної інформації. І зберігання її в JWT може перевищити дозволену довжину URL або cookie, що спричиняє проблеми. Крім того, ви потенційно надсилаєте великий обсяг даних у кожному запиті. [11]

Рішенням може бути – зберігати список "відкликаних токенів" у базі даних і перевіряти його для кожного виклику. І якщо токен є у цьому списку відкликаних, то блокувати користувача від подальших дій. Але тоді ви робите цей додатковий виклик до бази даних, щоб перевірити, чи відкликаний токен, і це обманює мету JWT взагалі.

2.8 Атаки на JWT

Розглянемо різні аспекти безпеки, пов’язані з використанням JWT, а також проаналізуємо основні ризики, які можуть виникнути при використанні цього стандарту. Дослідимо потенційні атаки, які можуть бути спрямовані на JWT, та шляхи їх запобігання та вирішення. Тож наступні ризики:

- **Перехоплення:** Зі зростанням кількості атак типу "людина посередині" (MitM), незашифровані JWT становлять ризик. [7]
- **Алгоритми та ключі:** Гнучкість JWT у дозволі різних алгоритмів - це перевага, але слабкі алгоритми або ключі призводять до вразливостей. [7]
- **Зберігання токенів:** Те, як і де ви зберігаєте JWT, може викликати вразливості. Популярний localStorage часто є спокусливим, але небезпечним вибором. [8]
- **Відсутність можливості скасування доступу**: JWT не може скасувати доступ користувача, навіть якщо його обліковий запис було припинено або видалено. [8]

Приклад наступного коду є ризиковим

```
const token = jwt.sign({ user: ‘Alice’ }, ‘simpleKey’);

localStorage.setItem(‘userToken’, token);
```

Цей код показує токен, який легко отримати за допомогою інструментів браузера. Будь-який зловмисний скрипт може використовувати ці токени, збережені таким чином.

Ось як можна його дістати:

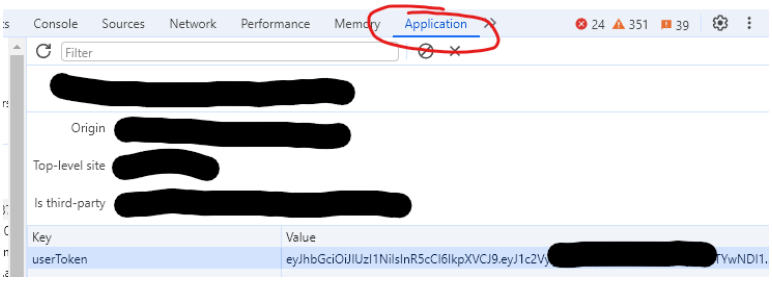

Рисунок 2.8 – Приклад збереження токена в localstorage

Копіюємо даний токен і вже за допомогою сервісу jwt.io ми можемо подивитися на всі дані які містив токен. І якщо там буде якась конфіденційна інформація, це може стати вразливістю:

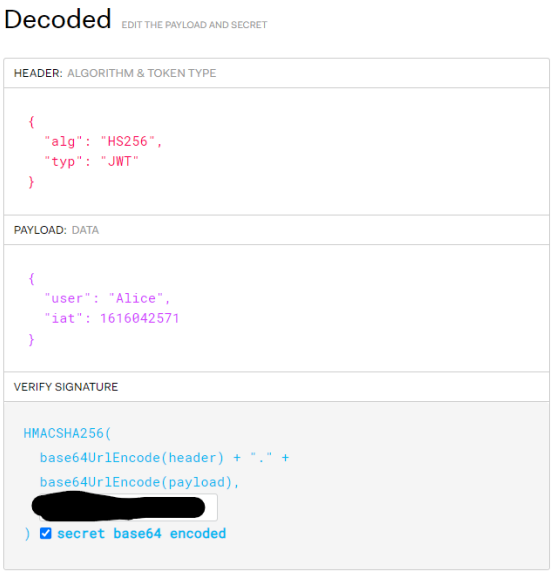

Рисунок 2.9 - Демонстрація наповнення jwt

Висновок до розділу 2

JWT (JSON Web Token) - це стандарт передачі інформації у форматі JSON, який забезпечує безпеку та автентифікацію. Він використовується для авторизації та обміну інформацією між сторонами. JWT може бути підписаний секретним ключем або за допомогою пари публічного/приватного ключа. Для авторизації користувачів кожен запит містить JWT у заголовку авторизації. Існують різні алгоритми шифрування, такі як HS256 і RS256, проте рекомендується використовувати RS256 за його перевагами. Незважаючи на переваги, JWT має свої ризики, такі як можливість перехоплення, використання слабких алгоритмів та проблеми зберігання токенів, які потребують уважного обходу для забезпечення безпеки.

3 Вдосконалення механізмів авторизації на основі JWT

3.1 Забезпечення відкликання токенів

Звичайна схема авторизації по токену є наступною:

1. Логінація за логіном + пароль
2. Автентифікація на сервері
3. Повернення JWT для подальших запитів
4. Запит для збереження «допису»
5. Перевірка валідності токена
6. Збереження «допису» в базі даних
7. Відповідь про успішне завершення

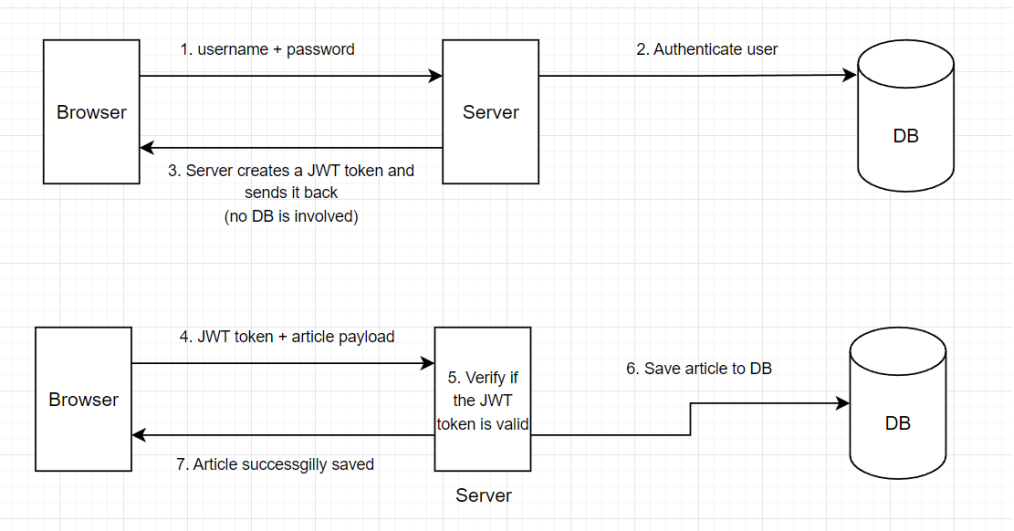

Рисунок 3.1 – Стандартний механізм авторизації та створення допису

Тут є проблемою те, що якщо користувача вже не існує, або від вийшов із системи, його token все ще працює, бо його час «життя» ще не вийшов, і можна повторно використовувати, якщо мати до нього доступ. Треба створити колекцію в базі даних з відкликаними токенами і коли користувач виходить із системи або видаляється, то його токен додається до колекції відкликаних токенів.

І після четвертого пункту система має перевірити чи даний токен не відкликаний, щоб не давати доступ до ресурсу у разі, якщо токен було відкликано.

Наступна діаграма говорить, що якщо користувач вийшов із системи, а хтось хоче використати його токен для створення допису від його іменні, то йде додаткова перевірка на те, чи був відкликаний токен:

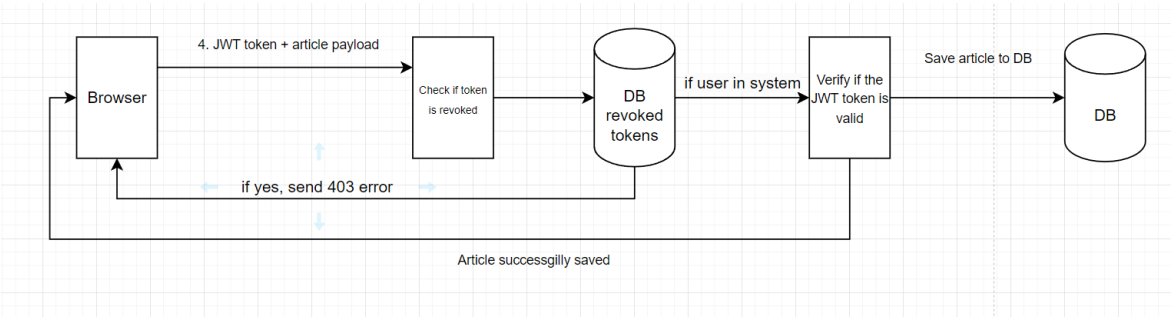

Рисунок 3.2 – Демонстрація перевірки на відкликаний токен

Тобто наш алгоритм буде наступним?

1. Логінація за логіном + пароль
2. Авторизація на сервері
3. Повернення JWT для подальших запитів
4. Запит для збереження «допису»
5. **Перевірка чи токен був відкликаний**
6. Перевірка валідності токена
7. Збереження «допису» в базі даних
8. Відповідь про успішне завершення

3.2 Зменшення розмірності токенів

Замість зберігання імен усіх файлів у JWT, можна зберігати тільки ідентифікатори користувача, а повний список файлів отримувати з бази даних або іншого сховища при кожному запиті.

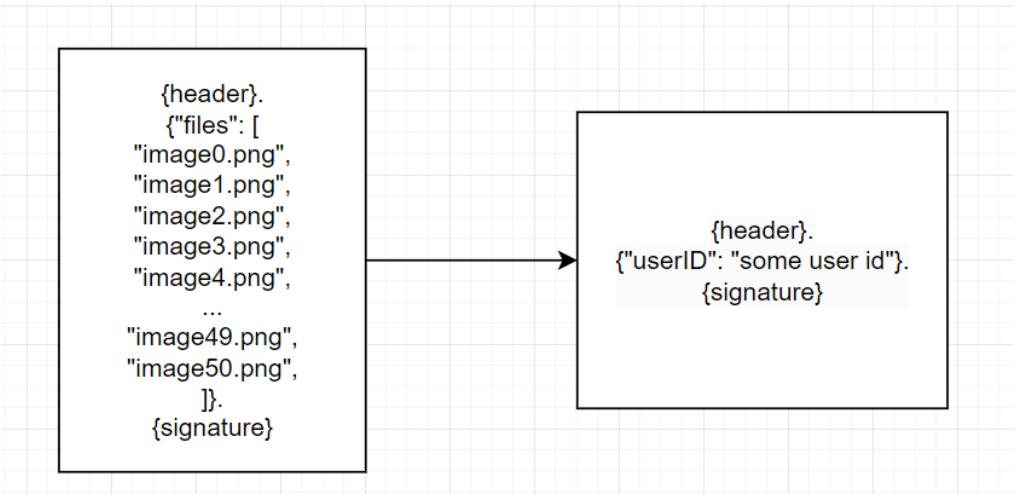

Рисунок 3.3 – Варіант заміни payload

3.3 Попередження підробки ключів

При використані асиметричних алгоритмів шифрування, коли є секретний та приватний ключі проблемою є те, що перевірка токенів спирається на те, який алгоритм шифрування користувач поклав у заголовок токену, спираючись на те, що він буде перевіряти за допомогою публічного ключа.

Потрібно щоб був один алгоритм шифрування і функція перевірки не спиралася на той алгоритм, який прийшов в заголовку, а на той який був раніше визначений при розробці, адже є ймовірність, що зловмисник використає симетричний алгоритм, який зашифрує за допомогою публічного ключа як секретним, і при перевірці за допомогою публічного – перевірка буде вдала.

Для захисту від цієї проблеми мають бути наступні кроки:

- Завчасно вибрати один асиметричний алгоритм, яким буде шифруватися та розшифровуватися;
- Не звертати уваги при перевірці токена на той алгоритм, який прийшов в header(це може бути підробка зловмисника), а розшифровувати

завчасно домовленим алгоритмом та використовувати публічний та секретний ключі.

3.4 Передача матриці доступу у токені

Дану практичну частину я буду розробляти в середовище розробки intellij idea на NodaJs використовуючи JavaScript. Токени генерувалися завдяки бібліотеці jsonwebtoken.

Наша матриця доступу буде наступною:

Табляця 3.1 – Матриця доступу

Операція/Роль	user	admin
Read	article	article
Create	article	article
Update	own_article	any_article
Delete	own_article	any_article

Тобто, роль визначає рівень доступу користувача (наприклад, user, або admin).

Операція - це дія, яку може здійснювати користувач (наприклад, read, create, update, delete).

Ресурс - це об'єкт або сутність, до якої здійснюється доступ (наприклад, article або own_article, де own_article означає статтю, яка належить користувачеві).

Тобто, моя матриця доступу зберігається в окремому файлі, і коли буде запит, він буде отримувати назву сервісу та дію яку хочуть зробити, й виглядає наступним чином:

```
const accessMatrix = {
    articlesService: {
        user: {
            read: true,
            create: true,
            updateOwn: true,
            deleteOwn: true,
        },
        admin: {
            read: true,
            create: true,
            updateOwn: true,
            updateAny: true,
            deleteOwn: true,
            deleteAny: true,
        }
    },
    commentsService: {
        user: {
            read: true,
            create: true,
            updateOwn: true,
            deleteOwn: true
        },
        admin: {
            read: true,
            create: true,
            updateOwn: true,
            updateAny: true,
            deleteOwn: true,
```

```
        deleteAny: true
      }
    }
};
```

Ось приклад користувачів в базі даних:

Рисунок 3.4 – Приклад користувачів у хмарній базі даних mongo

При логінації нам видається токен, який містить деякі дані про користувача та головне його роль, яка дає йому певні можливості на нашому веб сайті. Переглянути розшифрований токен ми можемо завдяки сервісу https://jwt.io/ :

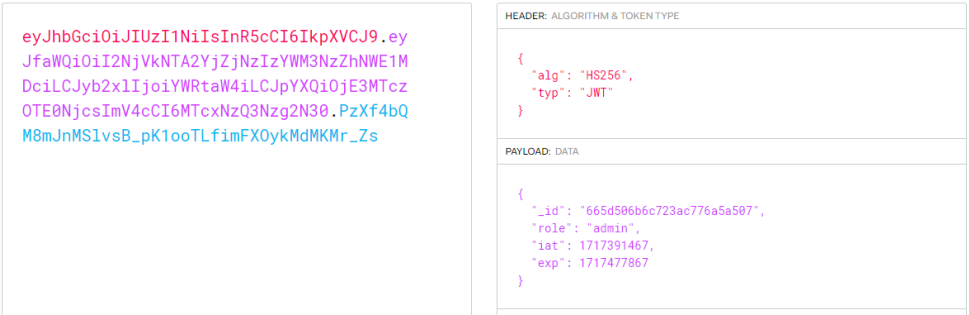

Рисунок 3.5 – Розшифровка отриманого токена при логінації

Тут видно наше корисне навантаження(payload), що роль в даного користувача user. Тож, він може створювати собі статті та переглядати всі, але не може редагувати чужі.

Створена стаття користувачем Anton

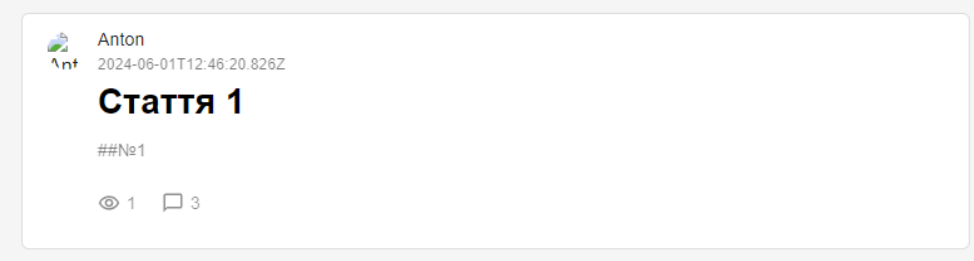

Рисунок 3.6 – Приклад створеної статті

Він може її редагувати або видаляти, маючи цю можливість завдяки ролі своїй

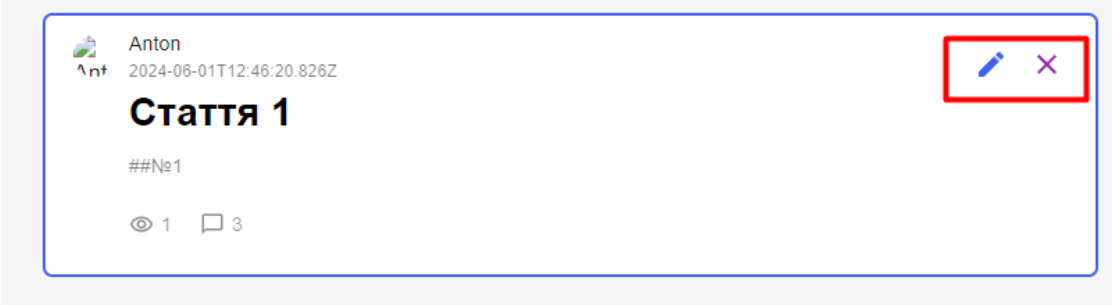

Рисунок 3.7 – Демонстрація можливості редагування власної статті

А ось приклад, коли інший користувач зайшов у систему і бачить цю статтю, в нього не буде можливості змінити чи видалити дану статтю

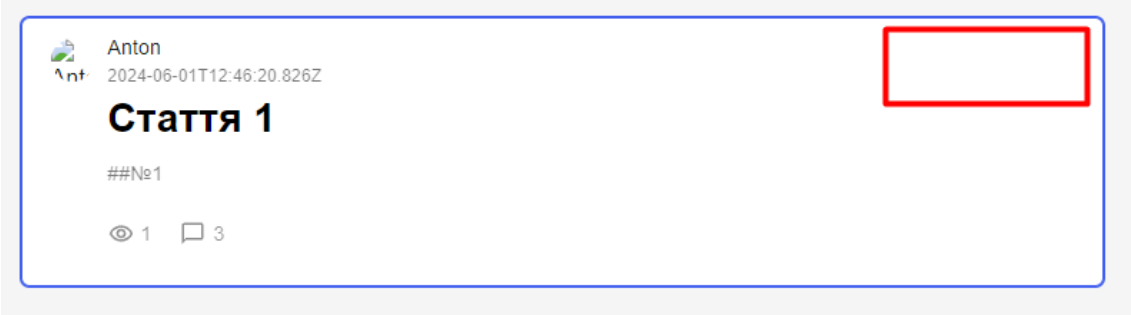

Рисунок 3.8 – Демонстрація відсутності редагування чужої статті

Наступний код показує як саме відбувається розмежування:

Функція на сервері для діставання токена з заголовка запита та інформації про користувача (user), який кладеться в req.user та в якому лежить роль:

```
// Middleware для перевірки токена та ролі користувача
function verifyToken(req, res, next) {
  const token = req.headers['authorization'];
  if (!token) return res.status(401).send('Потрібно авторизуватися');

  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).send('Невірний токен');
    req.user = user;
    next();
  });
}
```

checkAccess – це функція, яка перевіряє чи має користувач доступ до того чи іншого мікросервіса:

```
// Middleware для перевірки доступу користувача
function checkAccess(service, action) {
  return (req, res, next) => {
    const userRole = req.user.role;
    const serviceAccess = accessMatrix[service];

    if (!serviceAccess || !serviceAccess[userRole] || !serviceAccess[userRole][action]) {
      return res.status(403).send('У вас немає прав для виконання даної операції');
    }

    next();
  };
}
```

Наступна маршрутизація демонструє чи можна переглядати користувачеві всі статті:

```
app.get('/articles',
   verifyToken,
   checkAccess('articlesService', 'read'),
   postController.getAll);
```

Тут видно, що якщо роль в користувача user або admin, то далі виконуватиметься логіка повернення користувачу даних про всі статті. Як і показано на попередніх рисунках, що бачити статті можуть всі користувачі, а змінювати чи видаляти ні.

Наступна функція показує, як сервер перевіряє чи можна користувачу давати право на редагування тієї чи іншої статті, вибравши її:

```
// Захищений маршрут для редагування статті
app.put('/articles/:id',
   verifyToken,
   checkAccess('articlesService', 'updateOwn'),
   checkArticleAccess,
   postController.getById,
   );

// Функція перевірки доступу користувача до редагування статті
function checkArticleAccess(req, res, next) {
   const userRole = req.user.role; // Отримання ролі користувача з об'єкту req.user
   const articleOwnerId = getArticleOwnerId(req.params.id); // Отримання ID власника
статті за її ID (замініть на вашу логіку отримання ID власника)

   // Якщо користувач - адміністратор, він має право на редагування будь-якої статті
   if (userRole === 'admin') {
      next(); // Продовжити виконання запиту
   } else if (userRole === 'user') { // Якщо користувач - зареєстрований користувач
      if (req.user.id === articleOwnerId) { // Якщо користувач є власником статті
         next(); // Продовжити виконання запиту
      } else {
         res.status(403).send('Ви не можете редагувати цю статтю'); // Відмова у доступі,
```

```
якщо користувач не є власником статті
      }
   } else {
      res.status(403).send('У вас немає прав для редагування статті'); // Відмова у доступі для інших користувачів
   }
}
```

postController.getById – це та частина яка повертає нам конкретно вибрану статтю, якщо у користувача є можливість редагування після того як він пройшов попередні перевірки.

3.5 Розробка прикладу авторизації з використанням JWT

3.5.1 Відсутність можливості скасування доступу

JSON Web Tokens (JWT) не визначають стандартного методу перевірки токенів, залишаючи реалізацію логіки перевірки на розсуд розробників. Такий підхід може призвести до невідповідностей та вразливостей, якщо валідація не виконується суворо. [10]

Розробники повинні забезпечити перевірку всіх аспектів JWT, включаючи його підпис, час закінчення терміну дії та вимогу емітента. Це необхідно для запобігання потенційним атакам, таким як повторне використання або підробка токена. Тільки ретельна перевірка всіх складових JWT може гарантувати безпеку системи автентифікації. [10]

Рішення, щоб зробити це безпечніше і покращити наш код наступним чином:

1. Додати поле isActive до об’єкта користувача, яке вказує на те, чи активний користувач

```
[
   {
      id: " 665d504bfec64947b21eb315",
```

```
    username: "Anton",
    email: "user@gmail.com",
    password:
"$2b$07$Md86LB.RrmzkEq40HK/vfeDEIFmENSEfnZHgY0m/RNOsgbeIOVZy2",
    role: "user",
    isActive: true
  },
  {
    id: " 665d506b6c723ac776a5a507",
    username: "user2",
    email: "strukalo@example.com",
    password:
"$2b$07$24exJuoNuTsrRQvPAB9xg.PiKl7VPL3FeraY1D7S.Mla4m95sg/qq",
    role: "admin",
    isActive: true
  }
]
```

2. Створити список відкликаних токенів, у якому будуть зберігатися відкликані токени
3. Змінити маршрут /logout, щоб тут була присутня деактивація користувача(розлогінація) та відкликання токена

```
// Маршрут для відкликання токена (деактивація користувача)
app.post('/logout', verifyToken, (req, res) => {
  const token = req.headers['authorization'];
  blacklistedTokens.push(token); // Додати токен до списку відкликаних токенів
  const user = req.body;
  user.isActive = false; // Деактивація користувача

  // Оновлення бази
  await Post.findByIdAndUpdate(
    {
      _id: user.id,
    },
```

```
        {
          ...user,
        },
        { returnDocument: "after" },
    )
    res.send('Ви вийшли з системи');
});
```

4. Доповнити нашу функцію verifyToken перевіркою чи наш токен не відкликаний

```
// Middleware для перевірки токена та ролі користувача
function verifyToken(req, res, next) {
    const token = req.headers['authorization']; // Отримання токена з заголовка authorization
    if (!token) return res.status(401).send('Потрібно авторизуватися');

    // Перевірка, чи токен не відкликаний
    if (blacklistedTokens.includes(token)) return res.status(403).send('Токен відкликаний');

    jwt.verify(token, SECRET_KEY, (err, user) => {
        if (err) return res.status(403).send('Невірний токен');

        // Перевірка, чи користувач активний
        const currentUser = users.find(u => u.id === user.id);
        if (!currentUser || !currentUser.isActive) return res.status(403).send('Користувач деактивований');

        req.user = user; // Зберігання інформації про користувача у властивості req
        next();
    });
}
```

Ось так зберігаються токени в базі даних і коли ми перевіряємо новий, ми звертаємося до бази даних, щоб порівняти даний токен з вже відкликаними:

Рисунок 3.9 – Приклад збереження токенів в базі даних

На розробленому додатку зі статтями ось ми отримали токен після логінації:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NjVkNTA2YjZjNzIzYWM3NzZhNWE1MDciLCJpYXQiOjE3MTgxMDcwNzAsImV4cCI6MTcxODE5MzQ3MH0.OxB1_vjSsCXTJ1vzdbWFcR2q1SroYgkXnAZKmxUy75U

Далі я вийшов із системи та припустимо, що мій токен зловмисник хоче використати коли я не в системі.

Ось мій токен вже серед відкликаних в базі даних mongoDB:

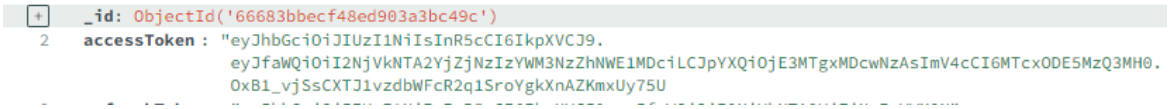

Рисунок 3.10 – Токен після виходу із системи в колекції відкликаних

Тепер спробуємо по цьому токену дістати дані, бачимо, що токен відкликаний.

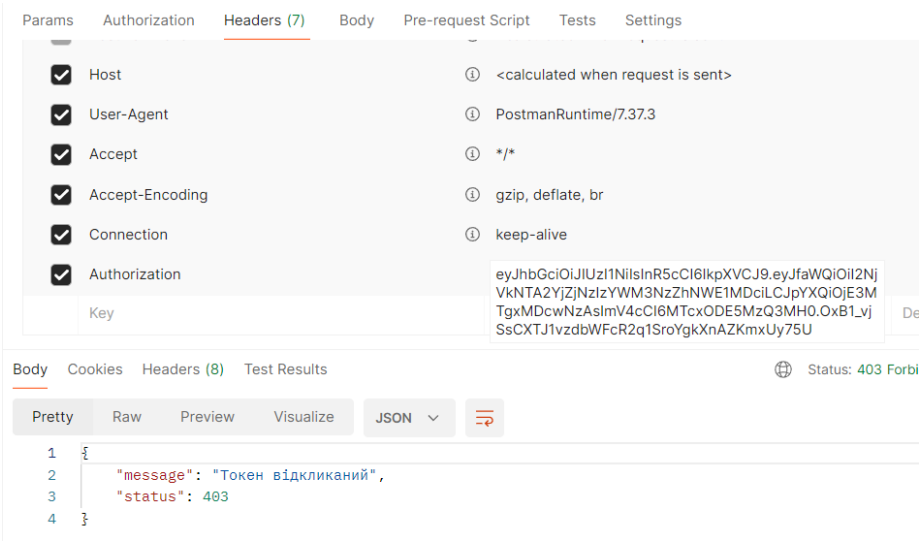

Рисунок 3.11 – Невдала спроба отримати дані

3.5.2 Підробка ключів в шифруванні токена

Статистика використання алгоритмів у сучасних реаліях

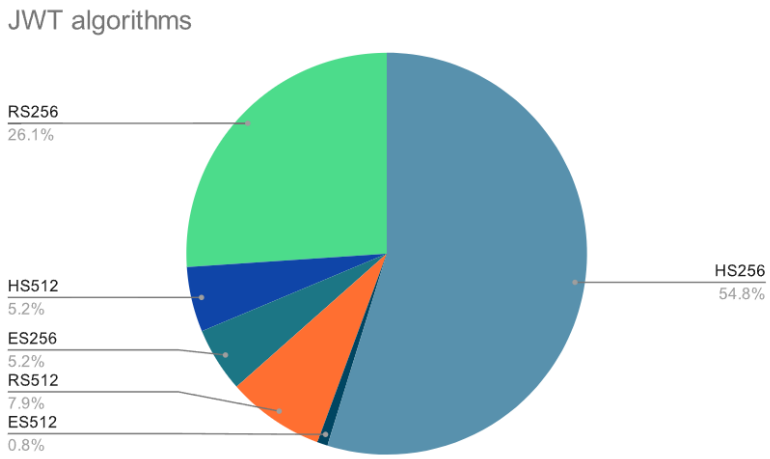

Рисунок 3.12 – Діаграма використання алгоритмів

Для зловмисника, симетричний алгоритм легше піддати грубій форсації, бо немає потреби збирати відкриті ключі, а час обчислення є максимально ефективним. [14]

З діаграми видно, що симетричні алгоритми використовуються ширше, ніж асиметричні алгоритми (рисунок 3.12). Симетричні алгоритми вимагають менше обчислень, тому у великому масштабі вони можуть мати кращу

продуктивність, ніж асиметричні алгоритми, але забезпечують меншу безпеку. [14]

Симуляція атаки підробки ключа

З асиметричними алгоритмами приватний ключ використовується для підробки JWT (частина підпису), а відкритий ключ використовується для перевірки JWT.

Зловмисник може змінити алгоритм, указаний у заголовку, на симетричний (наприклад, HS256) і створити підпис, використовуючи відкритий ключ, який законно використовується сервером для перевірки.

Приклад ситуації, коли неправильно реалізований Back-End використовуватиме відкритий ключ і запускатиме симетричний алгоритм (як зловмисник вказав у заголовку), щоб успішно перевірити JWT, підроблений зловмисником.

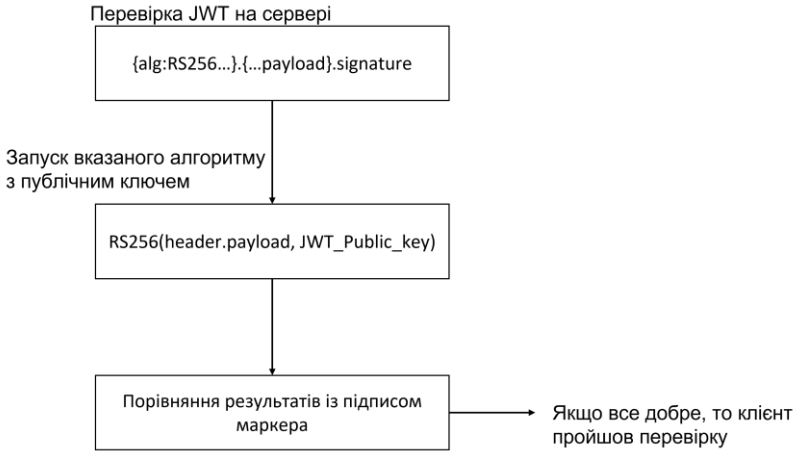

Рисунок 3.13 – Механізм перевірки токена

Помилка при перевірці може виникнути у двох випадках:

- Заголовок і корисне навантаження були відредаговані, але підпис – ні.
- Є неправильна конфігурація правильного алгоритму або правильного ключа.

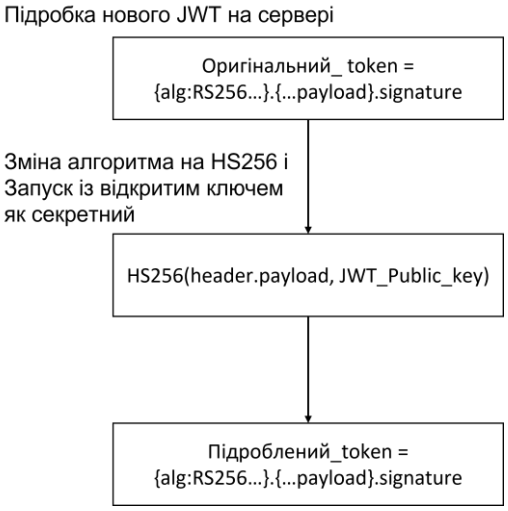

Рисунок 3.14 – План атаки

Останній крок: сервер перевірить маркер за допомогою процесу (рисунок 3.13), у якому сервер довіряє полю alg. Він перевірить токен, запустивши алгоритм HS256 із відкритим ключем як секретним, що саме хотів зловмисник.

Рішення: треба використовувати попередньо визначені алгоритми в процесі перевірки та не довіряйте введеним користувачами.

У створеному додатку спочатку функція просто перевіряла токен на валідність наступним чином: jwt.verify(token, publicKey), що означає, що воно буде використовувати алгоритм, той який переданий в заголовку.

Ось, коли зловмисник підмінив, ми маємо, що алгоритм буде використовуватися HS256, як зловмисник і хотів.

Рисунок 3.15 – Вдала підміна алгоритму

Якщо змінити перевірку на:

```
        jwt.verify(token, publicKey, { algorithms: ['RS256'] }, (err, user) => {
 // перевірка користувача
            …
 next();
});
```

Тут видно що перевірка буде виключно по RS256, який був завчасно домовлений при розробці. Використавши той самий токен у заголовку - маємо знову невдалий план отримати доступ, так як нам буде відмовлено:

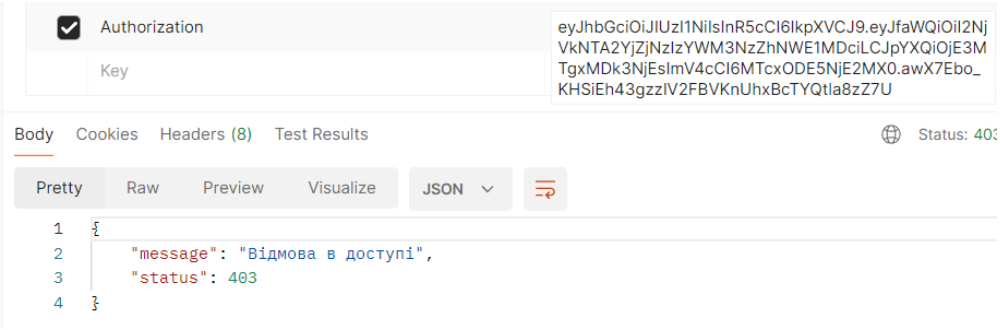

Рисунок 3.16 – Невдала спроба авторизації

3.5.3 Обмежена інформація в payload

JWT (JSON Web Tokens) мають обмеження на розмір даних, які вони можуть зберігати. Ця проблема стає особливо актуальною в ситуаціях, коли потрібно надати користувачеві доступ до великої кількості ресурсів або коли кожен ресурс має багато метаданих. Наприклад, якщо потрібно надати користувачеві доступ до 1000 файлів, і кожен файл має своє ім’я, довжина JWT може легко перевищити розумні межі.

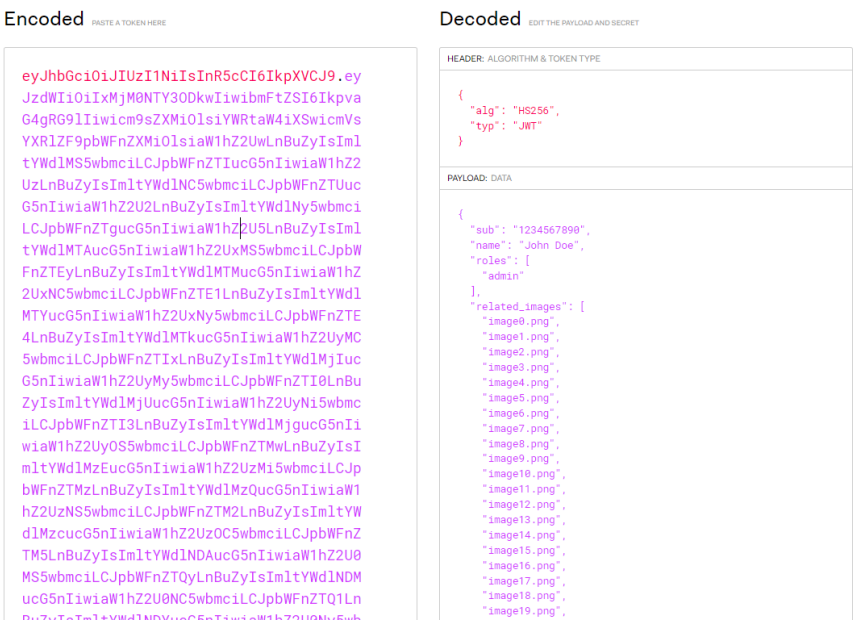

Рисунок 3.17 – Фрагмент токену більш, ніж на 20 файлів

Наприклад, уявімо, що кожен файл має ім’я з 20 символів. Якщо у нас є 1000 файлів, це означає, що нам потрібно зберегти 20 000 символів тільки для імен файлів. Додаючи сюди структуру JWT і інші необхідні поля, ми отримуємо JWT розміром понад 20 057 символів. Це суттєво перевищує оптимальний розмір для передачі через мережу і може призвести до проблем з продуктивністю та затримками.

Проблеми які можуть буди пов’язані з великими JWT:

- Підвищені витрати на передачу даних;
- Проблеми з продуктивністю;
- Обмеження на розмір заголовків.

Замість зберігання імен усіх файлів у JWT, будемо зберігати тільки ідентифікатори користувача, а повний список файлів отримувати з бази даних або іншого сховища при кожному запиті.

```
{
 "_id": "665d506b6c723ac776a5a507",
 "role" : "user",
```

"iat": 1718109761,

"exp": 1718196161

}

Для такого payload у нашому токені всього 191.

А для прикладу коли ми зберігаємо 1000 файлів з кожним іменем у нас буде понад 20 057 символів. То зекономлений час ми можемо використовувати для пошуку файлів в базі даних, якщо це буде по конкретному файлу, то ми швидко отримаємо або запитуваний файл, або відмову, якщо не маємо до нього доступу. І це буде набагато швидше ніж перебирати токен, вміст якого 20 000 символів.

Висновок до розділу 3

Реалізація та використання JWT (JSON Web Tokens) вимагає ретельного підходу до перевірки та валідації токенів для забезпечення безпеки системи.

Додаткові заходи безпеки включають додавання поля isActive до об’єкта користувача для позначення активності користувача. Також слід створити список відкликаних токенів для зберігання відкликаних токенів, що дозволить блокувати доступ користувачів із недійсними токенами.

Функція перевірки токена має доповнюватися перевіркою відкликаних токенів та активності користувача перед наданням доступу до ресурсів.

Для захисту від підробки ключів у шифруванні токена слід використовувати асиметричні алгоритми, такі як RS256, замість симетричних алгоритмів, які легше піддаються атакам. Визначення алгоритмів у процесі перевірки і недовіра введеним користувачами допоможуть запобігти підміні алгоритму.

Замість зберігання великої кількості даних у JWT, краще зберігати лише необхідну інформацію, наприклад, ідентифікатор користувача, і отримувати

додаткові дані з бази даних або іншого сховища при кожному запиті. Це дозволить уникнути проблем з продуктивністю та обмеженням на розмір заголовків.

ВИСНОВКИ

Аналіз авторизації за допомогою JWT:

JWT зменшує частоту звернень до бази даних, оскільки містить всю необхідну інформацію для авторизації, що підвищує продуктивність і спрощує масштабування. Формат JSON полегшує інтеграцію з різними платформами та мовами програмування, роблячи JWT універсальним інструментом.

Було розроблено модельний приклад авторизації з використанням JWT. Створено невеликий веб-застосунок, у якому реалізовано базову авторизацію за допомогою JWT. На основі цього прикладу відбувалися дослідження та вдосконалення методів, спрямовані на підвищення безпеки системи. Це дозволило наочно продемонструвати, як покращені методи можуть забезпечити ефективніший захист від типових атак на JWT, тим самим підтверджуючи їхню ефективність у реальних умовах.

Розробка механізмів покращення безпеки доступу:

- **Відкликання токенів**: запропонований метод відкликання токенів дозволяє миттєво скасувати дійсність токена у разі його компрометації або завершення сесії користувача. Це забезпечує додатковий рівень безпеки, особливо в розподілених системах, де може бути важко миттєво синхронізувати стан сесій між різними сервісами.
- **Підміна ключів при підписанні токенів**: було запропоновано, щоб алгоритм підписання визначався завчасно під час розробки і не звертав увагу на те, який алгоритм приходить в заголовку токена при перевірці, а використовував завчасно домовлений. Це запобігає можливим атакам, пов'язаним з підміною алгоритму підпису, що є важливим для підтримки безпеки в масштабованих розподілених системах.
- **Розмірність**: запропонований варіант зменшення розміру JWT дозволяє швидше передавати токен та зекономлений час використати на запит до

бази даних до конкретного файлу і вже відразу отримати або доступ, або відмову. Це підвищує ефективність роботи системи, особливо коли мова йде про обробку великої кількості запитів в розподілених системах.

Підсумок запропонованих методів покращення:

Запропоновані методи покращення безпеки доступу за допомогою JWT забезпечують більш ефективне та безпечне управління авторизацією, що особливо важливо для розподілених систем. Використання механізмів відкликання токенів, фіксація алгоритму підписання та оптимізація розміру токенів дозволяють не тільки підвищити продуктивність і безпеку, але й забезпечити більш надійну і гнучку роботу розподілених систем, що стикаються з високим навантаженням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1 – Про розподілені системи [Електронний ресурс]. – Режим доступу до ресурсу: http://surl.li/ucstv

2 - What Is Authorization [Електронний ресурс]. – Режим доступу до ресурсу: https://www.netsuite.com/portal/resource/articles/erp/authorization.shtml

3 - Аутентифікація і авторизація [Електронний ресурс]. – Режим доступу до ресурсу: https://qagroup.com.ua/publications/autentyfikatciia-i-avtoryzatciia/

4 - Popular Authentication Methods [Електронний ресурс]. – Режим доступу до ресурсу: http://surl.li/uctsu

5 - Introduction to JSON Web Tokens [Електронний ресурс]. – Режим доступу до ресурсу: https://jwt.io/introduction

6 - RS256 vs HS256 [Електронний ресурс]. – Режим доступу до ресурсу: https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/

7 - Handling JWT Safely: Mitigating Common Security Risks [Електронний ресурс]. – Режим доступу до ресурсу: https://qwiet.ai/handling-jwt-safely-mitigating-common-security-risks/

8 - JSON Web Token (JWT) Weaknesses [Електронний ресурс]. – Режим доступу до ресурсу: https://www.secopsolution.com/blog/jwt-weaknesses

9 - JWT header parameter injections [Електронний ресурс]. – Режим доступу до ресурсу: https://portswigger.net/web-security/jwt#jwt-header-parameter-injections

10 - JSON Web Token (JWT) Weaknesses [Електронний ресурс]. – Режим доступу до ресурсу: https://www.secopsolution.com/blog/jwt-weaknesses

11 - JSON Web Tokens (JWT) are Dangerous for User Sessions [Електронний ресурс]. – Режим доступу до ресурсу: https://redis.io/blog/json-web-tokens-jwt-are-dangerous-for-user-sessions/

12 - JWTs Aren’t Made for Authorization [Електронний ресурс]. – Режим доступу до ресурсу: https://www.permit.io/blog/jwts-arent-made-for-authorization

13 – Documentation of jwtwebtoken library [Електронний ресурс]. – Режим доступу до ресурсу: https://www.npmjs.com/package/jsonwebtoken

14 – Analyzing Broken User Authentication Threats to JSON Web Tokens [Електронний ресурс]. – Режим доступу до ресурсу: https://www.akamai.com/blog/security-research/owasp-authentication-threats-for-json-web-token

ДОДАТОК А

ОСНОВНІ ФУНКЦІЇ ДЛЯ АВТОРИЗАЦІЇ ТА МАРШРУТИЗАЦІЇ ПО BACK-END ЧАСТИНІ

```
import { ApiError } from "../errors";
import { IPost } from "../interfaces";
import { Post } from "../models";

class PostService {
  public async getAll(): Promise<IPost[]> {
    try {
      return (await Post.find().populate("user").exec()) as unknown as IPost[];
    } catch (e) {
      throw new ApiError(e.message, e.status);
    }
  }

  public async getLastTags(): Promise<string[]> {
    try {
      const posts = await Post.find().limit(5).exec();
      const tags = posts
        .map((post) => post.tags)
        .flat()
        .slice(0, 5);

      return tags;
    } catch (e) {
      throw new ApiError(e.message, e.status);
    }
  }

  public async getById(postId: string): Promise<IPost> {
    try {
      const post = (await Post.findByIdAndUpdate(
        { _id: postId },
        { $inc: { viewsCount: 1 } },
        { returnDocument: "after" },
```

```
    ).populate("user")) as unknown as IPost;

    if (!post) {
      throw new ApiError("Post is not found", 422);
    }

    return post;
  } catch (e) {
    throw new ApiError(e.message, e.status);
  }
}

public async deleteById(postId: string): Promise<void> {
  try {
    await Post.findByIdAndDelete({ _id: postId });
  } catch (e) {
    throw new ApiError(e.message, e.status);
  }
}

public async create(dataPost: IPost, userId: string): Promise<IPost> {
  try {
    const post = (await Post.create({
      ...dataPost,
      user: userId,
    })) as unknown as IPost;
    return post;
  } catch (e) {
    throw new ApiError(e.message, e.status);
  }
}

public async updateById(updateData: IPost, postId: string): Promise<IPost> {
  try {
    const updatedPost = (await Post.findByIdAndUpdate(
      {
        _id: postId,
      },
```

```
        {
          ...updateData,
        },
        { returnDocument: "after" },
      )) as unknown as IPost;
      return updatedPost;
    } catch (e) {
      throw new ApiError(e.message, e.status);
    }
  }
}

export const postService = new PostService();


// Middleware для перевірки токена та ролі користувача
function verifyToken(req, res, next) {
    const token = req.headers['authorization'];
    if (!token) return res.status(401).send('Потрібно авторизуватися');

    jwt.verify(token, SECRET_KEY, (err, user) => {
        if (err) return res.status(403).send('Невірний токен');
        req.user = user;
        next();
    });
}

// Middleware для перевірки доступу користувача
function checkAccess(service, action) {
    return (req, res, next) => {
        const userRole = req.user.role;
        const serviceAccess = accessMatrix[service];

        if (!serviceAccess || !serviceAccess[userRole] || !serviceAccess[userRole][action]) {
            return res.status(403).send('У вас немає прав для виконання даної операції');
        }

        next();
```

```
    };
}

// Middleware для перевірки доступу користувача
function checkAccess(service, action) {
    return (req, res, next) => {
        const userRole = req.user.role;
        const serviceAccess = accessMatrix[service];

        if (!serviceAccess || !serviceAccess[userRole] || !serviceAccess[userRole][action]) {
            return res.status(403).send('У вас немає прав для виконання даної операції');
        }
        next();
    };
}

app.get('/articles',
    verifyToken,
    checkAccess('articlesService', 'read'),
    postController.getAll);

// Функція перевірки доступу користувача до редагування статті

function checkArticleAccess(req, res, next) {

    const userRole = req.user.role; // Отримання ролі користувача з об'єкту req.user

    const articleOwnerId = getArticleOwnerId(req.params.id); // Отримання ID власника статті за її
ID (замініть на вашу логіку отримання ID власника)
    // Якщо користувач - адміністратор, він має право на редагування будь-якої статті
    if (userRole === 'admin')
        next(); // Продовжити виконання запиту
    } else if (userRole === 'user') { // Якщо користувач - зареєстрований користувач
        if (req.user.id === articleOwnerId) { // Якщо користувач є власником статті
            next(); // Продовжити виконання запиту
        } else {
```

```
            res.status(403).send('Ви не можете редагувати цю статтю'); // Відмова у доступі, якщо користувач не є власником статті
        }
    } else {
        res.status(403).send('У вас немає прав для редагування статті'); // Відмова у доступі для інших користувачів
    }
}


// Маршрут для відкликання токена (деактивація користувача)
app.post('/logout', verifyToken, (req, res) => {
    const token = req.headers['authorization'];
    blacklistedTokens.push(token); // Додати токен до списку відкликаних токенів
    const user = req.body;
    user.isActive = false; // Деактивація користувача

    // Оновлення бази
    await Post.findByIdAndUpdate(
        {
            _id: user.id,
        },
        {
            ...user,
        },
        { returnDocument: "after" },
    )
    res.send('Ви вийшли з системи');
});


// Middleware для перевірки токена та ролі користувача
function verifyToken(req, res, next)
    const token = req.headers['authorization']; // Отримання токена з заголовка authorization
    if (!token) return res.status(401).send('Потрібно авторизуватися');

    // Перевірка, чи токен не відкликаний
    if (blacklistedTokens.includes(token)) return res.status(403).send('Токен відкликаний');
    jwt.verify(token, SECRET_KEY, (err, user) => {
```

```
        if (err) return res.status(403).send('Невірний токен');
        // Перевірка, чи користувач активний
        const currentUser = users.find(u => u.id === user.id);

        if (!currentUser || !currentUser.isActive) return res.status(403).send('Користувач деактивований');
        req.user = user; // Зберігання інформації про користувача у властивості req

        next();
    });
}
```