Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ПОЯСНЮВАЛЬНА ЗАПИСКА

Дипломної роботи
магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформацімайні технології*
(шифр і назва галузі знань)
спеціальність *125 Кібербезпека*
(код і назва спеціальності)
освітній ступень *магістр*
(назва освітньої програми)
освітньо-наукова програма *кібербезпека*

на тему: «Підвищення ефективності авторизації за рахунок використання веб-токену JSON»

Виконавець: студент ІІ курсу, групи КБм-21

(підпис) **Лінецький Артем Павлович** (прізвище ім'я по-батькові)

	Прізвище, ініціали	Оцінка	Підпис
Науковий керівник	Наконечний В. С.		

Рецензент	Сайко В.Г.		

Нормоконтроль	Даков С.Ю.		

Київ 2022

Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
«__» _____ 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

студента *КБм-21* (група) *Лінецького Артема Павловича* (прізвище ім'я по-батькові)

Тема дипломної роботи *Підвищення ефективності авторизації за рахунок використання веб-токену JSON*

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 29.10.2021

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес авторизації з використанням веб-токену JSON.
Предмет досліджень	Методи авторизації з використанням веб-токену JSON.
Мета	Розробка методів підвищення ефективності авторизації з використанням веб-токену JSON.
Вихідні дані для проведення роботи	методів підвищення ефективності авторизації з використанням веб-токену JSON

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна удосконалення методу ефективності авторизації з використанням веб-токену JSON

Практична цінність покращення системи авторизації з використанням веб-токену JSON.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Розробка плану для досягнення мети роботи	29.10.2021 – 23.01.2022
Аналіз літературних джерел	24.01.2022 – 14.02.2022
Розробка методу авторизації з використанням веб-токену JSON	15.02.2022 – 24.04.2022
Оформлення і друк пояснювальної записки	25.04.2022 – 19.05.2022

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект Зниження збитків через викрадення даних

Соціальний ефект Покращення захисту даних при використанні в інформаційних системах

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ _____Наконечний В. С._
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ ___Лінецький А.П._____
(підпис) (прізвище, ініціали)

Дата видачі завдання: _____
Термін подання дипломної роботи до ЕК _____

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Підвищення ефективності авторизації за рахунок використання вебтокену JSON»: 54 сторінки, 12 рисунків, 1 додаток, 2 таблиці, 53 літературних джерела.

Об’єкт дослідження – процес авторизації з використанням вебтокену JSON.

Предмет дослідження: методи авторизації з використанням вебтокену JSON.

Мета роботи – розробка методу підвищення ефективності авторизації з використанням вебтокену JSON.

Методи дослідження – системний підхід, методи порівняння, структурний аналіз, спостереження та вимірювання.

У роботі досліджено сучасні методи автентифікації та авторизації. Проведено аналіз вразливостей авторизації з використанням вебтокену JSON. Запропоновано метод підвищення ефективності захисту авторизації з використанням вебтокену JSON. Побудовано систему авторизації з використанням вебтокену JSON з додатковим сервісом управління доступом за допомогою Redis. Запропонований метод імплементовано в проект «Spilno».

Наукова новизна: удосконалено метод захисту авторизації з використанням вебтокену JSON за рахунок додаткового сервісу управління доступом за допомогою Redis. Токен доступу зберігається в базі даних Redis.

Практичне значення роботи полягає у покращенні системи авторизації з використанням вебтокену JSON при розробці проекту «Spilno».

Результати даного дослідження можуть бути використані для імплементації системи авторизації в різних інформаційних системах.

Ключові слова: автентифікація, авторизація, керування доступом, JWT, JSON Web Tokens.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application programming interface

CPU – Central processing unit

CSRF – Cross-site request forgery

DNS – Domain Name System

FIM – Federated Identity Management

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Token

OAuth – Open Authorization

OIDC – OpenID Connect

OS – Operating system

SAML – Security Access Markup Language

SSO – Single SignOn

XSS – Cross-Site Scripting

ВСТУП

У веб-застосунках, як правило, присутні елементи, доступ до яких потрібно обмежити. Приклад таких елементів може бути особистий кабінет користувача або функціональність, доступна тільки після оплати. Щоб отримати доступ до подібних ресурсів, користувачеві необхідно пройти перевірку прав, тобто процедуру авторизації. При цьому програма повинна переконатися в автентичності користувача, якому надається доступ, тобто здійснити автентифікацію.

Один із способів автентифікації у веб-застосунках полягає у використанні сесій. У цьому випадку після успішного входу користувача в систему сервер генерує унікальний ідентифікатор, зберігає його в базі даних у прив'язці до даних користувача і відсилає його на клієнтську частину програми. Надалі при зверненні до сервера клієнт прикріплює цей ідентифікатор до запиту. Отримавши запит із зазначенням сесійного ідентифікатора, сервер витягує з бази даних відомості про контекст роботи користувача та обробляє запит, що надійшов з урахуванням цієї інформації. Такий спосіб має ряд недоліків. Насамперед, необхідність здійснення операцій з базою даних накладає додаткове навантаження на сервер та ускладнює його архітектуру. Крім того, у разі сервісної розподіленої архітектури системи ведення такої бази даних може обмежувати можливості масштабування та збільшувати кількість мережевих запитів. Однією з альтернатив використання сесій для реалізації автентифікації у веб-застосунку є застосування токенів доступу.

У сфері безпеки веб-додатків веб-токени JSON (JWT) відіграють все більш значущу роль. Вони дуже добре підходять для застосування в розподілених системах, оскільки їх перевірка може бути виконана службою-клієнтом без необхідності центрального доступу до сервера. Однак ця властивість також означає, що немає простого способу відкликати токен після його видачі [1].

Автентифікація на основі токенів має низку важливих переваг, особливо в системі із сервісною розподіленою архітектурою [2]. Насамперед, оскільки у токені може міститися вся необхідна інформація для авторизації користувача у сервісі,

немає потреби у додаткових зверненнях до сервера автентифікації. Токен, у свою чергу, містить цифровий підпис цієї інформації. Звіряючи цифровий підпис токена, сервіс надійно визначає автентичність інформації про користувача.

Сервер автентифікації в простому варіанті використання може не зберігати інформацію про видані токени, за рахунок чого зменшується розмір програмного коду і спрощується архітектура бази даних. При автентифікації такого роду робота з базою даних включає переважно операції на читання, що сприяє підвищенню швидкості роботи і розширює можливості для масштабування архітектури.

Крім того, токен, виданий довіреним сервером, може бути прийнятий іншими сервісами та доменами. Таким чином, користувач, що має подібний токен, отримує доступ одночасно до цілого ряду ресурсів, розподілених серед різних серверів, що приймають цей токен.

При проектуванні клієнт-серверної системи, що застосовує JWT-автентифікацію, важливо врахувати низку особливостей, що впливають на безпеку. JSON-токени можуть зберігатися в браузері двома способами [3]: в сховищі DOM або в кукі. У першому випадку система може бути схильна до XSS-атаки, так як JavaScript має доступ до DOM-сховища і зловмисник може витягти звідти токен для подальшого використання від імені користувача.

Тому тема дослідження «Підвищення ефективності авторизації за рахунок використання веб-токену JSON» є актуальною.

Об'єкт дослідження: процес авторизації з використанням веб-токену JSON.

Предмет дослідження: методи авторизації з використанням веб-токену JSON.

Мета кваліфікаційної магістерської роботи: розробка методів підвищення ефективності авторизації з використанням веб-токену JSON

Для досягнення поставленої в роботі мети необхідно здійснити виконання наступних завдань:

- провести теоретичний аналіз засад автентифікації та авторизації користувачів в сучасних інформаційних системах;

- дослідити та проаналізувати рівень захищеності систем авторизації з використанням веб-токену JSON;

- дослідити та проаналізувати методи захисту авторизації з використанням веб-токену JSON;
- побудувати та реалізувати систему авторизації на основі запропонованого методу;
- провести тестування побудованої системи авторизації.

Наукова новизна: удосконалено метод захисту авторизації з використанням вебтокену JSON за рахунок додаткового сервісу управління сеансами за допомогою Redis. Токен доступу зберігається в базі даних (Redis). Він допомагає керувати доступом запитів клієнтської частини до core-сервісів.

Апробація результатів роботи: основні наукові положення і результати роботи доповідалися та обговорювалися на V Міжнародній науково-практичній конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем” (PCSITS)” (Київ, 2022). Основні положення дипломної роботи викладені у матеріалах наукової конференції.

РОЗДІЛ 1
АНАЛІЗ ТЕХНОЛОГІЙ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ

1.1 Сучасні методи автентифікації та авторизації

Реалізація авторизації при розробці різних веб і корпоративних додатків є досить комплексним і відповідальним завданням від якого залежить безпека даних в системі.

У роботі M. B. Jones йдеться мова про появу нового набору протоколів відкритої ідентифікації, який використовує представлення даних JSON та прості шаблони зв'язку на основі REST. Ці протоколи та формати даних спеціально розроблені для простоти використання у браузерах та сучасних середовищах веб-розробки [4].

Jánoky L. V., Ekler P., Levendovszky зазначають, що веб-токени JSON забезпечують масштабований розподілений спосіб контролю доступу користувачів для сучасних веб-систем. Але після видачі токена JWT немає тривіального способу відкликати токен [5].

У роботі M. B. Jones, B. Campbell та C. Mortimore йдеться про специфікацію draft-ietf-oauth-jwt-bearer-05. Ця специфікація визначає використання токена-носія JSON Web Token (JWT) як засіб для запиту токена доступу OAuth 2.0, а також для використання як засіб автентифікації клієнта [6].

У статті P. Solapurkar розглядається OAuth 2.0 [7]. Цей стандарт є делегованим середовищем авторизації, що дозволяє здійснювати безпечну авторизацію додатків, що працюють на різних платформах.

У сфері медичних послуг OAuth дозволяє пацієнту (власнику ресурсу), який звертається за медичною допомогою в режимі реального часу, авторизувати автоматичні щомісячні платежі зі свого банківського рахунку (сервера ресурсів), при цьому пацієнту не потрібно надавати свої облікові дані до клініки (клієнтська програма). OAuth 2.0 досягає цього за допомогою токенів, виданих сервером

авторизації, який забезпечує перевірений доступ до захищеного ресурсу. Для забезпечення безпеки токени доступу мають певний термін дії та недовговічні. Таким чином, клінічна програма може використовувати токен оновлення для отримання нового токена доступу для щомісячної оплати готівкою за надання медичних послуг у режимі реального часу [8].

Токенам оновлення необхідно захищене сховище для того, щоб вони не були схильні до викрадення, оскільки будь-який зловмисник може використовувати їх для отримання нового доступу та оновлення токенів [9].

Оскільки OAuth 2.0 відкидає підписи і повністю покладається на SSL/TLS, він уразливий для фішинг-атак при доступі до сумісних з Application programming interface (API) інтерфейсів. У цій статті розроблено підхід, який поєднує веб-токен JSON (JWT) з OAuth 2.0 для запиту токена доступу OAuth із сервера авторизації, коли клієнт бажає використовувати попередню автентифікацію та авторизацію. Експериментальна оцінка підтверджує, що запропонована схема є практично ефективною, усуває необхідність у безпечному сховищі, усуваючи необхідність мати або зберігаючи токен оновлення, використовує сигнатуру та запобігає різноманітним атакам безпеки, що дуже бажано у службах охорони здоров'я [10].

Хмарна індустрія стала ключовою технологією, яка розкриває потенціал великих даних, Інтернету речей, мобільних та веб-додатків та інших пов'язаних технологій, але це також пов'язано зі своїми проблемами, такими як управління, безпека та конфіденційність. Ця стаття присвячена проблемам безпеки та конфіденційності хмарних обчислень з особливим акцентом на автентифікацію користувачів та керування доступом для хмарних програм SaaS. Пропонована модель використовує платформу, яка використовує JWT без збереження стану і безпеки для автентифікації клієнтів та керування сеансами. Крім того, авторизований доступ до захищених хмарних ресурсів SaaS ефективно управляється. Відповідно, були введені компонент Gate Match Policy та компонент Policy Activity Monitor (PAM). Крім того, інші підкомпоненти, такі як Блок перевірки політики (PVU) та БД проксі-сервера політики (PPDB) також були створені для оптимізованої доставки послуг. Теоретичний аналіз запропонованої

моделі зображує систему, яка є безпечною, легкою та легко масштабованою для підвищення безпеки хмарних ресурсів та управління ними [3].

Аналіз літературних та інформаційних джерел [1-53] дозволив виокремити основні методи автентифікації та авторизації.

Існують різні підходи до автентифікації користувачів:

– автентифікація по паролю;

– автентифікація по сертифікатам;

– автентифікація по ключам доступу;

– автентифікація по токенах.

Автентифікація по паролю заснована на тому, що користувач повинен надати логін і пароль для успішної ідентифікації та автентифікації в системі [24]. Пара логін/пароль встановлюється користувачем при реєстрації в системі. Існує кілька стандартних протоколів для автентифікації пароля [31]: HTTP, Forms, URL-запити, тіло запиту, HTTP-заголовки.

Автентифікація по сертифікатам. Сертифікат – це набір атрибутів, які ідентифікують власника. Кореневий центр сертифікації виступає як посередник, який забезпечує справжність сертифіката. Сертифікат також пов'язується із закритим ключем за допомогою шифрування. який зберігається у власника сертифіката і дозволяє наочно підтвердити факт володіння сертифікатом на стороні клієнта.

Сертифікат разом із закритим ключем може зберігатися в операційній системі. У браузері, у файлі, на окремому фізичному пристрої. Закриті ключі часто додатково захищені паролем або PIN-кодом. Автентифікація за сертифікатом відбувається при підключенні до сервера. Цей механізм добре підтримується браузерами. Це дозволяє користувачеві вибрати та використовувати сертифікат.

Якщо веб-сайт дозволяє цей метод автентифікації. Під час автентифікації сервер перевіряє сертифікат за такими правилами [52]:

1) сертифікат повинен бути підписаний довіреним центром сертифікації ключів (аудит ланцюга сертифікатів);

2) сертифікат має бути дійсним на поточну дату;

3) сертифікат не повинен бути відкликаний відповідним центром сертифікації (перевірити список виключень).

Після успішної автентифікації Веб-додатки можуть авторизувати запити на основі інформації про сертифікат, наприклад, ім’я власника, емітент сертифіката, серійний номер або відбиток пальця.

Використання сертифіката для автентифікації є більш безпечним методом, ніж автентифікація паролем. Це робиться шляхом створення цифрового підпису в процесі автентифікації. Це підтверджує реальність використання приватного ключа в конкретних ситуаціях, але проблеми з розповсюдженням та підтримкою сертифікатів роблять цей метод автентифікації широко недоступним [53].

Автентифікація за допомогою ключів доступу. Цей метод найчастіше використовується для автентифікації пристроїв, служб або інших програм під час доступу до веб-сервісів. Тут секретною інформацією є ключі доступу - довгі унікальні рядки, які містять довільний набір символів, по суті змінюючи комбінацію логін/пароль.

У більшості випадків сервер генерує ключі доступу до запитів користувачів, які потім зберігають ці ключі в клієнтських програмах. При створенні ключа також можна обмежити рівень доступності та дії, які отримує клієнт при її перевірці. Хорошим прикладом використання ключової автентифікації є Amazon Web Services Cloud [20].

Особливість HTTP протоколу у тому, що не вміє зберігати стан користувача. Кожен новий запит до сервера ніяк не залежить від попередніх і в такому разі ми не маємо можливості визначити, хто саме зробив запит. Впоратися з цією проблемою допомагає сесія браузера (сеанс) – механізм, який дозволяє відстежити запити від одного браузера та зберегти деякі змінні під час переходів на сторінки сайту [20].

Авторизація на основі сесій означає, що зберігається стан користувача на сервері. У простішому варіанті сесія зберігається в оперативній пам'яті програми. У складнішому варіанті в базі даних. Коли здійснюється вхід на сайт, сервер здійснює перевірку облікових даних у базі. Після успішної перевірки сервер створює сесію з деяким ідентифікатором (session_id), зберігає її в пам'яті і відправляє користувачеві.

На стороні клієнта отриманий session_id зберігається у cookie браузеру. Коли здійснюється запит на захищений ресурс, cookie міститься у заголовку запиту до сервера. Далі сервер шукає session_id у пам'яті (in-memory, база даних) і якщо така сесія існує, то дозволяє відвідати захищений ресурс (Рисунок 1.1) [11, 13, 14].

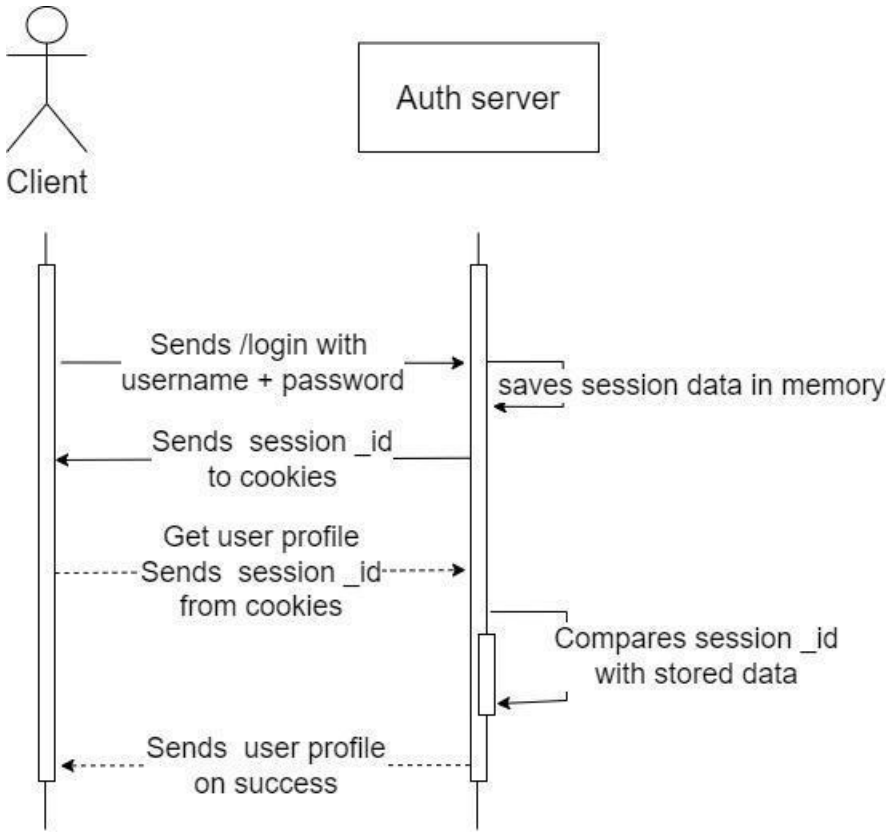


Рисунок 1.1 − Session-based автентифікація

Авторизація на основі токенів позбавляє сервер від менеджменту сесіями, що дозволить суттєво знизити витрати на пам'ять. Досягається це за рахунок зберігання всієї необхідної інформації про авторизацію на клієнті.

1.2 Автентифікація на основі токенів

Автентифікація на основі токенів найчастіше застосовується при побудові розподілених систем Single SignOn (SSO), де один додаток (resource server) делегує функцію автентифікації користувачів іншому додатку (authentication

service). Типовий приклад цього способу - вхід в додаток через обліковий запис в соціальних мережах. Тут соціальні мережі є сервісами автентифікації, а додаток довіряє функцію автентифікації соціальним мережам [12].

Токен - це набір інформації або даних, що передається з однієї системи до іншої в процесі виконання SSO [22]. Дані можуть бути просто email адресою та інформацією про систему, що відправила токен. Токени повинні мати цифровий підпис для одержувача, щоб підтвердити, що він надійшов з надійного джерела. Сертифікат для електронного підпису надається під час початкового етапу налаштування.

Токени це засіб авторизації для кожного запиту від клієнта до сервера. Токени (і відповідно сигнатура токена) генеруються на сервері базуючись на секретному ключі (котрий зберігається на сервері). Токен в результаті зберігається на клієнті (може також зберігається в куки) і використовується за необхідності авторизації будь-якого запиту [23].

Існує кілька поширених форматів токенів для веб-додатків [23]:

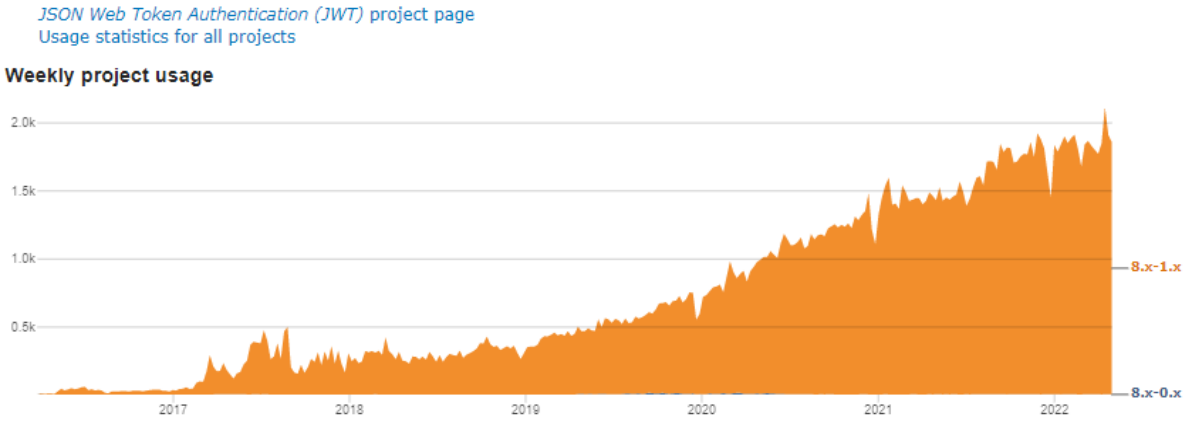
1) Simple Web Token (SWT) – найпростіший формат, що є будь-якою парою ключів/значень у форматі HTML. Цей стандарт визначає кілька імен резервних копій: Distributor, Audible, ExpiresOn та HMACSHA256. Токен підписано симетричним ключем, тому постачальник сертифіката та захищене джерело повинні мати цей ключ.

2) Security Access Markup Language (SAML) – визначає токен у форматі XML (аргументи SAML), яка включає набір додаткових тверджень, що стосуються емітенту, теми, необхідних умов перевірки та користувачеві. Тег SAML підписується несиметричним шифруванням. Крім того, на відміну від попередніх форматів, тег SAML включає механізм підтвердження володіння токеном, який запобігає блокуванню токена при атаці посередині при використанні незахищеного посилання.

3) JSON Web Token (JWT) – містить три розділи, розділених точками: заголовок, поле та підпис. Перші два розділи представлені у форматі JSON та закодовані у додаткових 64 форматах. До набору компонентів входять

необов'язкові пари ключ/значення, крім того, стандарт JWT визначає кілька запасних імен (розподіл, перевірка, перевірка тощо). Підписи генеруються шляхом обчислення симетричного та несиметричного електронного підпису.

У сучасних інформаційних системах перевага надається методам автентифікації за допомогою веб-токенів JSON. Це підтверджують дані з сайту drupal.org, представлені у вигляді графіка (Рисунок 1.2) [15].



Week	8.x-0.x	8.x-1.x	Total
May 1, 2022	2	1,852	1,854
April 24, 2022	2	1,906	1,908
April 17, 2022	2	2,110	2,112
April 10, 2022	1	1,845	1,846
April 3, 2022	2	1,766	1,768

Рисунок 1.2 − Кількість нових проектів, що використовують JWT авторизацію

JSON Web Token (JWT) − це JSON об'єкт, визначений у відкритому стандарті RFC 7519 [36]. Найпоширеніший випадок використання цього токена при авторизації. Коли користувач входить до системи, кожен наступний HTTP-запит включає JWT, який дає йому доступ до захищених служб та ресурсів веб-застосунків. Це зручно, тому що його легко налаштувати та можна використовувати різні домени

JWT використовується для перевірки автентифікації користувача наступним чином. Спочатку користувач заходить на сервер авторизації за допомогою авторотаційного ключа (це може бути пара логін/пароль, або Facebook ключ, або Google ключ, або ключ від іншого облікового запису). Потім сервер авторизації створює JWT та відправляє його користувачеві. Коли користувач робить запит до програми API, він додає до нього отриманий раніше JWT. Коли користувач робить

API запит, програма може перевірити за переданим із запитом JWT що це за користувач, його права, а також рівень його доступу. На рисунку 1.3 зображена загальна схема використання JWT для авторизації [25].

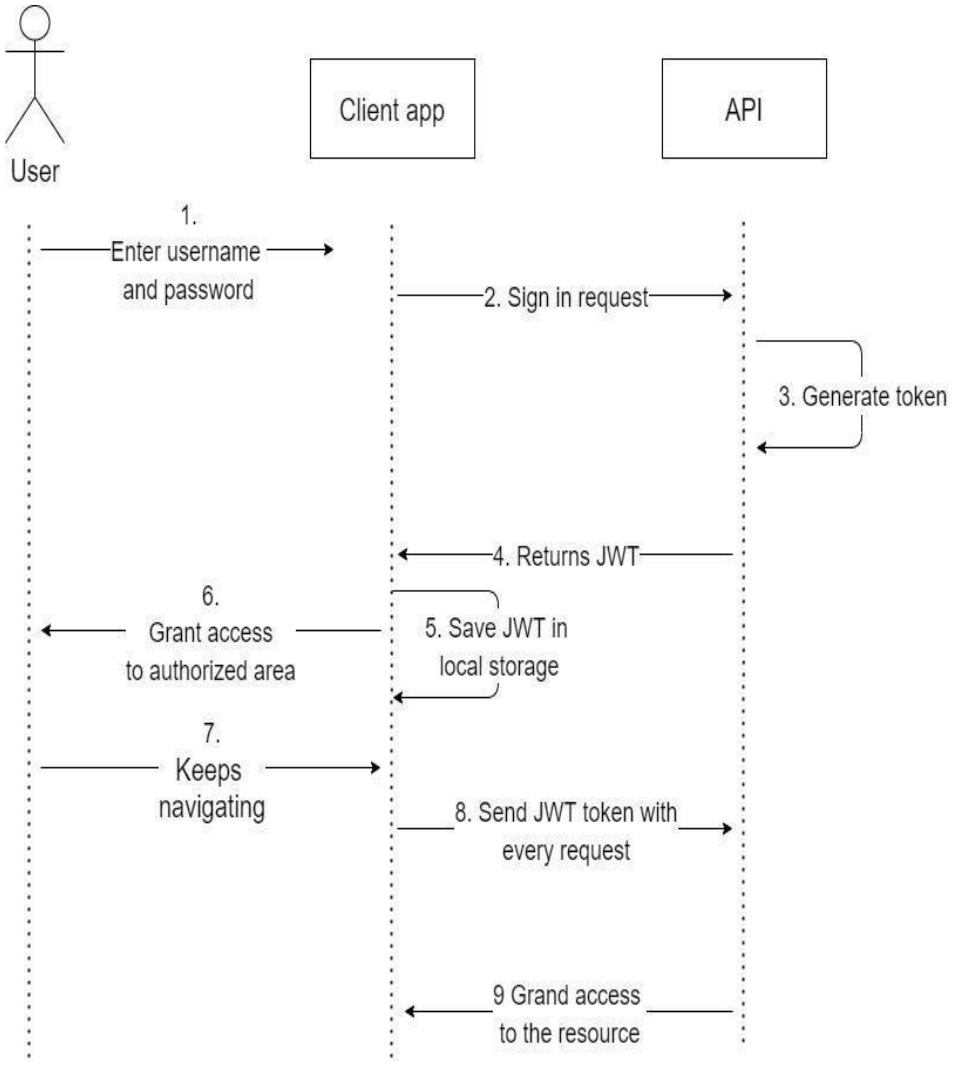


Рисунок 1.3 − Автентифікація на основі токенів

Поширені випадки використання JWT [46]:

1. Основним випадком використання JWT буде як токену доступу в потоці OAuth, тобто токен доступу в потоці OAuth може мати формат JWT. Оскільки JWT є автономним, клієнту, який має токен доступу у форматі JWT, не потрібно буде викликати сервер авторизації для перевірки токена доступу. Це чудово з точки зору продуктивності та масштабованості, а властивості безпеки JWT також гарантують, що JWT можна довіряти.

2. На підприємстві, коли у вас є окремі програми, що виконують автентифікацію та довіряють програми, JWT можуть бути дуже корисними. У цьому випадку система автентифікації може видати JWT клієнтській програмі, а клієнтська програма може представити його додатку-споживачу (або додатку, що довіряє). Перевага використання JWT у цьому випадку полягає в тому, що застосунок-споживач не повинен звертатися до програми автентифікації для перевірки JWT. Він може підтвердити підпис JWT і мати впевненість, що JWT дійсний і не підроблений.

3. JWT дуже корисні, коли ви розробляєте мікросервіси і маєте різні мікросервіси для різних частин архітектури додатків, які надають різні послуги. У цьому випадку всі мікрослужби програми-споживачі можуть покладатися на JWT, видані мікросервісами програми автентифікації, які видають JWT. Таким чином, мікросервіси програм-споживачів повинні надавати послугу, яку вони надають, і не турбуватися про надання послуг автентифікації. Використання JWT має ідеальний сенс у світі мікросервісів.

Аналіз джерел [11-46] дозволив здійснити порівняння поширених методів автентифікації: на основі токенів та на основі сесій.

Дані аналізу методів автентифікації дозволяють зазначити, що автентифікація на основі токенів має велику кількість переваг: легше масштабувати; простіший у використанні; вбудована функція закінчення терміну дії; не потрібно запитувати у користувачів "згоду на використання файлів cookie"; запобігає CSRF; працює краще на мобільному; працює для користувачів, які блокують файли cookie.

Таблиця 1.1

Порівняння автентифікації на основі токенів та на основі сесій

	На основі сесій	На основі токенів
Масштабованість	Чим більше зростає навантаження на сервер, тим більше пам'яті потрібно виділяти для зберігання інформації про сесію	Вся інформація про авторизацію лежить на клієнті, тому серверу не потрібно виділяти пам'ять під сесії та очищати її
Робота з нативними програмами	Немає нативної підтримки	Є нативна підтримка
Підтримка мікросервісної	Cookie не мають крос-доменної підтримки	Підтримує мікросервісну архітектуру

	На основі сесій	На основі токенів
архітектури		
Підтримка API	Не підтримує	Підтримує
Навантаження на сервер	При кожному запиті необхідно зробити запит у БД за даними користувача	Позбавляє запиту в БД за даними користувача, т.к. у токені вже є дані
Вразливість атак з використанням міжсайтового скриптингу (XSS)	Не мають надійного захисту	Не мають надійного захисту
Вразливість підробки міжсайтових запитів (CSRF)	Не мають надійного захисту	Мають захист

Разом з тим, слід зазначити, що автентифікація на основі токенів також не є ідеальною та має вразливості.

Висновок до першого розділу

У даному розділі було розглянуто теоретичні аспекти автентифікації та авторизації, технологія автентифікації за допомогою токенів. Проведено аналіз поширених методів автентифікації: на основі токенів та на основі сесій.

Наявність надійного механізму автентифікація є важливою вимогою при побудові веб-додатків, що працюють із захищеними даними.

Використання токенів для організації доступу дозволяє спростити архітектуру сервера, підвищити продуктивність та дає можливість одночасної роботи з різними сервісами та доменами. Веб-токени JSON забезпечують масштабований розподілений спосіб контролю доступу користувачів для сучасних веб-систем.

Дані аналізу методів автентифікації дозволяють зазначити, що автентифікація на основі токенів має велику кількість переваг.

JWT − це непрозорий автономний токен, який призначається користувачеві після успішного входу до постачальника ідентифікаційних даних. На відміну від непрозорих токенів сеансу, які вимагають від додатків-споживачів викликати постійну базу даних, щоб дізнатися особу користувача, який володіє токеном, JWT є

самостійним токеном, яким володіє клієнт та може бути представлений додатку-споживаючу. Додаток-споживач може перевіряти та прочитати вміст токена, не звертаючись до постачальника ідентифікаційних даних.

Разом з тим, слід зазначити, що автентифікація на основі токенів також не є ідеальною та має вразливості.

РОЗДІЛ 2
АНАЛІЗ ВРАЗЛИВОСТЕЙ АВТЕНТИФІКАЦІЇ З ВИКОРИСТАННЯМ JWT ТА СПОСОБІВ ЇХ УСУНЕННЯ

2.1 Структура JWT

JSON Web Token − це JSON об'єкт, визначений у відкритому стандарті RFC 7519 [33]. Коли користувач входить до системи, кожен наступний HTTP-запит включає JWT, який дає йому доступ до захищених служб та ресурсів веб-застосунків.

JWT токен розділений на три частини [4]:

Header – заголовок.

Payload – корисні дані.

Signature – підпис.

Заголовок (Header) містить у собі інформацію про те, як повинен обчислюватися JWT підпис. Заголовок передається у вигляді JSON об'єкта і виглядає так [16]:

header = {"alg": "HS256", "type": "JWT"}

Поле alg визначає алгоритм хешування. Цей алгоритм використовуватиметься під час створення підпису. Наприклад, у випадку HS256, для його обчислення потрібен лише один таємний ключ. Також можливе використання і іншого алгоритму, а саме RS256 – на відміну від попереднього, він є асиметричним та створює пару ключів: приватний та публічний.

Приватний ключ потрібен для створення підпису, а публічний ключ використовується для перевірки справжності цього підпису, це ще один, додатковий захід безпеки.

У полі type вказується, що це JWT токен.

Таблиця 2.1

Алгоритми, що підтримують токени JWT

Параметр alg	Значення Цифровий підпис або алгоритм MAC
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm
HS512	HMAC using SHA-512 hash algorithm
RS256	RSASSA-PKCS1-v1_5 using SHA-256 hash algorithm
RS384	RSASSA-PKCS1-v1_5 using SHA-384 hash algorithm
RS512	RSASSA-PKCS1-v1_5 using SHA-512 hash algorithm
PS256	RSASSA-PSS using SHA-256 hash algorithm (only node ^6.12.0 OR >=8.0.0)
PS384	RSASSA-PSS using SHA-384 hash algorithm (only node ^6.12.0 OR >=8.0.0)
PS512	RSASSA-PSS using SHA-512 hash algorithm (only node ^6.12.0 OR >=8.0.0)
ES256	ECDSA using P-256 curve and SHA-256 hash algorithm
ES384	ECDSA using P-384 curve and SHA-384 hash algorithm
ES512	ECDSA using P-521 curve and SHA-512 hash algorithm
none	No digital signature or MAC value included

Корисні дані (PayLoad) – це корисні дані, що зберігаються всередині токена JWT. Також ці дані називають JWT claims (заявки). У цьому прикладі розглядається сервер аутенфікації, який створює JWT з інформацією про користувача – userId.

payload = { "userId": "b08f86af-35da-48f2-8fab-cef3904660bd" }

У цьому прикладі використовується лише один claim, але їх може бути не обмежена кількість. Так само існує деякий стандартний список claim'ів.

iss (issuer) - визначає програму, з якої відправляється токен.

sub (subject) - визначає тему токена.

exp (expiration time) – час життя токена.

Audience (aud) – Аудиторія (aud).

Expiration time (exp) – Термін придатності (exp).

Not before (nbf) – Не раніше (nbf).

Issued at (iat) – Видано за адресою (iat).

JWT ID (jti) – Ідентифікатор токену.

Ці поля не є обов'язковими. Чим більше claim'ів створюється, тим більше за розміром виходить сам токен, що може негативно вплинути на продуктивність та викликати небажані затримки під час взаємодії із сервером.

Залишається створити підпис:

```
const SECRET_KEY = 'cAtwa1kkEy'
const unsignedToken = base64urlEncode(header) + '.' +
base64urlEncode(payload)
const signature = HMAC-SHA256(unsignedToken, SECRET_KEY)
```

Алгоритм base64url кодує заголовок та корисні дані. Алгоритм з'єднує закодовані дані через точку.

Далі отриманий рядок хешується алгоритмом, заданим у хедері на основі нашого секретного ключа.

```
// header eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
// payload eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1j
ZWYzOTA0NjYwYmQifQ
// signature -xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

На останньому етапі відбувається об'єднання всіх в токен JWT.

```
const token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) +
'.' + encodeBase64Url(signature)
// JWT Token
//eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0
zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.-
xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

Само собою використання JWT не приховує і не маскує дані автоматично. Сенс використання JWT – це перевірка, що надіслані дані були передані з

авторизованого джерела. Дані були закодовані та підписані. Мета кодування – перетворення структури.

Користувач, який хоче ввійти до надійної сторони, переходить на сервер автентифікації для входу. Сервер автентифікації перевіряє облікові дані користувача та створює JWT. У JWT сервер автентифікації включає такі твердження, як ім’я користувача, ім’я користувача, прізвище користувача, емітент токена, тема токена, дата випуску, а також дата закінчення терміну дії токена та дозволи користувача (якщо застосовно) — це стає корисним навантаженням JWT.

Потім сервер автентифікації створює заголовок, який говорить, який тип токена (type=JWT) і який алгоритм підпису, і поміщає це значення в заголовок токена. Потім створюється підпис JWT. Це найважливіший аспект безпеки токена.

Існує два методи створення підпису JWT [16].

Першим методом створення підпису є використання загального секретного криптографічного ключа для хешування заголовка та корисного навантаження JWT. Це робиться за допомогою алгоритму HMAC-SHA256 або HMACSHA512, який приймає криптографічний ключ і хешує заголовок та корисне навантаження JWT.

Сервер додатків, який споживає JWT, отримає цей JWT, і він також візьме той самий заголовок і корисне навантаження та використовуватиме раніше спільний криптографічний ключ і алгоритм HMACSHA256 або HMACSHA512 (незалежно від заголовка), щоб створити хеш і порівняти хеш, який включено в підпис JWT. Якщо обидва збігаються, це означає, що кінцевий користувач або людина посередині не втрутилися в JWT. І сервер додатків може довіряти JWT, оскільки ніхто, крім сервера автентифікації, не мав доступу до криптографічного ключа. Відповідно до цього припущення, користувачу надається доступ до ресурсу, якщо він має відповідні привілеї для доступу до цього ресурсу.

Другим методом створення підпису є використання алгоритму хешування для створення хешу заголовка та корисного навантаження, а потім використання приватного ключа для його шифрування, що призводить до підпису. Потім сторона-одержувач також бере заголовок і корисне навантаження і створює хеш, використовуючи той самий алгоритм хешування, а потім розшифровує підпис за

допомогою відкритого ключа відправника і порівнює хеш, надісланий відправником, з хешем, обчисленим одержувачем. Якщо вони збігаються, JWT може бути непідробним і дійсним, а користувачу надається доступ до ресурсу, якщо користувач має відповідні права доступу до цього ресурсу. JWT має вбудовані функції безпеки, завдяки яким він забезпечує автентифікацію відправника та захист цілісності.

2.2 Аналіз вразливостей автентифікації з використанням JWT

Однак у JWT також є ряд недоліків [21].

1. З точки зору безпеки, зберігаючи JWT у файлі cookie, він не відрізняється від будь-якого іншого ідентифікатора сеансу. Для токена великого розміру про файл cookie не може бути й мови.

2. Коли JWT зберігається в іншому місці, то з’являється новий клас атак. Локальне сховище не забезпечує жодного з тих самих механізмів безпеки, що й файли cookie. Локальне сховище, на відміну від файлів cookie, не надсилає вміст сховища даних із кожним запитом. Єдиний спосіб отримати дані з локального сховища – це використовувати JavaScript. Це означає, що будь-який зловмисник за допомогою JavaScript може отримати до нього доступ і вилучити.

3. Не можливо визнати недійсними окремі токени JWT. На відміну від сеансів, які сервер може визнати недійсними, коли захоче, окремі токени JWT без стану не можуть бути визнані недійсними до закінчення терміну їх дії. Це означає, що не можливо скасувати сеанс зловмисника після виявлення компромісу.

Маркери JWT без стану не можуть бути визнані недійсними або оновлені, що створює проблеми з розміром або з безпекою залежно від того, де вони зберігаються. Токени JWT із заповненим станом функціонально такі ж, як і файли cookie сеансу.

Виникають певні вразливості до таких загроз [21].

1. Перехоплення токена

Перехоплення токена може призвести до ряду неприємних наслідків. По-перше, так як JWT передається у відкритому вигляді, для отримання вихідних даних, що зберігаються в частині корисного навантаження, достатньо застосувати до цієї частини функцію base64UrlDecode. Тобто зловмисник, який перехопив токен, зможе витягти дані, що зберігаються в токені, про користувача. Відповідно до кращих практик, для запобігання такій загрозі рекомендується:

- використовувати під час передачі токенів захищене з'єднання;
- не передавати в токенах чутливі дані користувача, обмеживши знеособленими ідентифікаторами.

По-друге, зловмисник, який перехопив токен, зможе його використати та отримати доступ до програми від імені користувача, чий JWT був перехоплений. Тут рекомендації будуть наступні:

- як і першому випадку, використовувати захищене з'єднання під час передачі токенів;
- обмежити час життя JWT та використовувати механізм refresh tokens.

У сучасних схемах автентифікації, заснованих на JWT, після проходження автентифікації користувач отримує два токена:

- access token – JWT, на основі якого програма ідентифікує та авторизує користувача;
- refresh token – токен довільного формату, який слугує для оновлення access token.

Access token за такого підходу має дуже обмежений час життя (наприклад, одну хвилину).

Refresh token має тривалий час життя (день, тиждень, місяць), але він одноразовий і служить виключно для оновлення access token користувача.

Схема автентифікації у такому разі виглядає так:

користувач проходить процедуру автентифікації та отримує від сервера access token та refresh token;

- при зверненні до ресурсу користувач передає у запиті свій access token, на основі якого сервер ідентифікує та авторизує клієнта;

− при закінченні access token клієнт передає у запиті свій refresh token та отримує від сервера нові access token та refresh token;

− при закінченні refresh token користувач знову проходить процедуру автентифікації.

2. Підбір ключа симетричного алгоритму підпису

З використанням симетричних алгоритмів для підпису JWT (HS256, HS512 та інших.) зловмисник може спробувати підібрати ключову фразу. Підібравши її, зловмисник отримає можливість маніпулювати JWT-токенами так, як це робить сама програма, а отже зможе отримати доступ до системи від будь-якого зареєстрованого в ній користувача.

Рекомендації для захисту від атаки в цьому випадку такі:

− використовувати ключові фрази великої довжини, що складаються з великих і малих літер латинського алфавіту, цифр і спецсимволів, та зберігати їх у суворій конфіденційності;

− забезпечити періодичну зміну ключової фрази. Це знизить зручність використання для користувачів, але допоможе уникнути компрометації ключової інформації.

Використання алгоритму none

Використання в заголовку JWT алгоритму none вказує на те, що токен не був підписаний. У подібному токені відсутня частина з підписом, і встановити справжність стає неможливо. Щоб захиститися від такої атаки необхідно вести на стороні програми білий список дозволених алгоритмів підпису та відкидати токени, що використовують відмінні від наявних в списку алгоритми.

Для мінімізації ймовірності атак при використанні JWT потрібно дотримуватись таких рекомендацій [16, 17].

1. Токени бажано використовувати з коротким терміном дії. Вони мають бути дійсними лише кілька хвилин, щоб клієнт міг розпочати роботу.

2. Токен бажано використовувати лише один раз. Сервер програм видає новий токен для кожного завантаження.

3. Не рекомендується використовувати JWT для постійних, довготривалих даних.

4. Рекомендується використовувати під час передачі токенів захищене з'єднання та не передавати в токенах чутливі дані користувача, обмежившись знеособленими ідентифікаторами.

5. Використовувати ключові фрази великої довжини, що складаються з великих і малих літер латинського алфавіту, цифр і спецсимволів, та зберігати їх у суворій конфіденційності; забезпечити періодичну зміну ключової фрази.

6. Необхідно вести на стороні програми білий список дозволених алгоритмів підпису.

Але ці рекомендації не вирішують проблеми повторного використання викраденого токену.

2.3 Авторизація з використанням веб-токену JSON з додатковим сервісом управління доступом за допомогою Redis

Класична схема авторизації [26]:

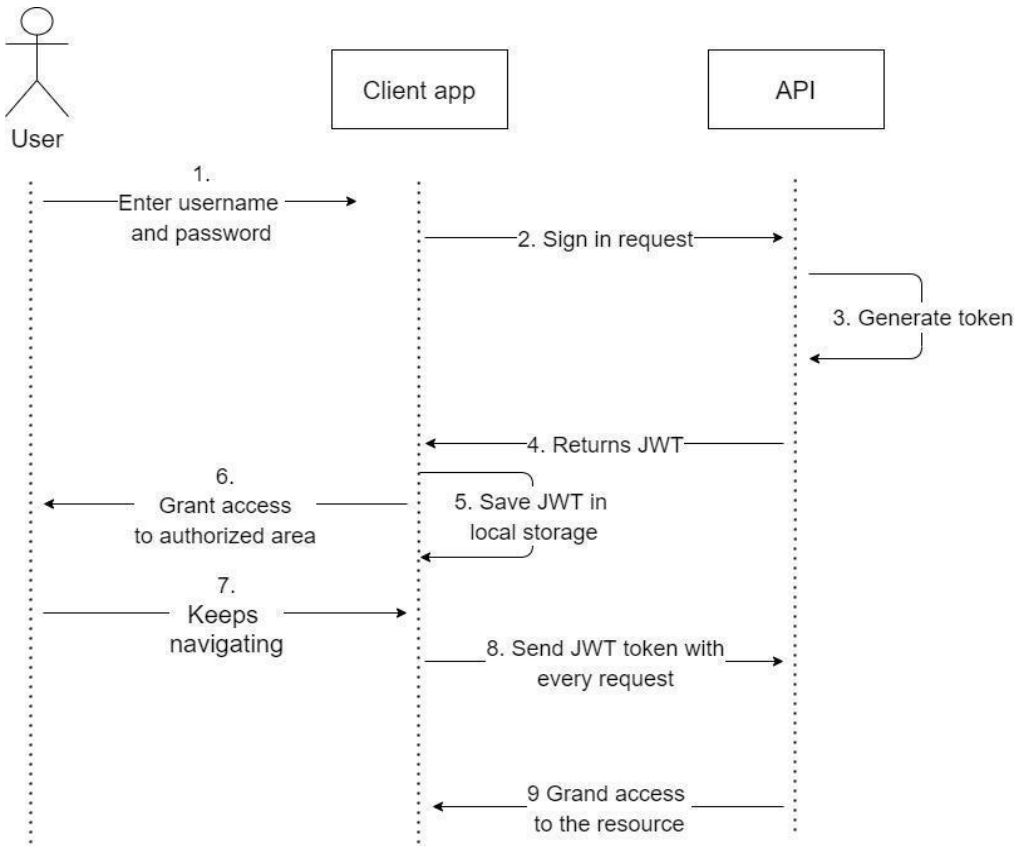


Рисунок 2.1 − Авторизація з використанням веб-токену JSON

Для підвищення безпеки авторизації автором пропонується зберегти токен у власній базі даних, щоб прив’язати його до користувача. Враховуючи, що пошук токена відбувається під час кожного запиту, гарною альтернативою є збереження токену в якомусь сховищі, наприклад Redis. У такому разі у токені можна зберігати більше корисної інформації для керування доступом.

Redis – резидентна система управління базами даних класу NoSQL з відкритим вихідним кодом, що працює зі структурами даних типу "ключ - значення". Використовується як для баз даних, так і для реалізації кешів, брокерів повідомлень [39].

Новий потік із вбудованим сховищем та підтримкою перевірки є таким:

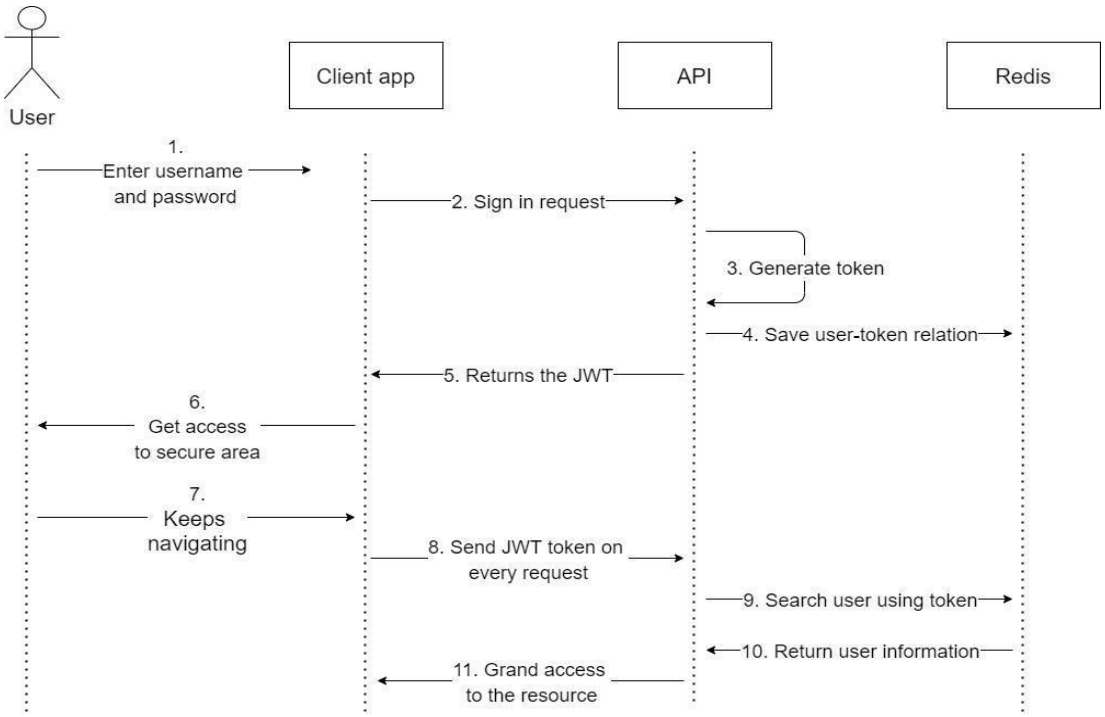


Рисунок 2.2 − Авторизація з використанням веб-токену JSON з додатковим сервісом управління доступом за допомогою Redis

Реалізація цього підходу:

1. Користувач здійснює вхід до програми, ввівши ім’я користувача та пароль.
2. Клієнтська частина надсилає запит до API.
3. Після надсилання сервер генерує токен JWT.

4. Токен JWT зберігається в сховищі Reids.
5. Токен JWT надсилається назад користувачеві.
6. Користувачу надається доступ за результатом перевірки токену JWT.
7. Користувач заходить до програми або здійснює навігацію в застосунку.
8. Токен JWT передається кожен раз при здійсненні запиту.
9. Сервер перевіряє токен JWT в сховище Redis.
10. Якщо для даного користувача вже існує токен, то старий видаляється та створюється новий токен користувача і буде здійснено перевірку додаткових даних користувача.
11. Користувач отримує доступ до ресурсів.

Така схема передбачає використання лише одного токену для одного користувача, що значно зменшує ймовірність використання токену зловмисником.

Висновок до другого розділу

У другому розділі було проведено аналіз рівню захищеності систем авторизації з використанням веб-токену JSON.

Аналіз літературних джерел та електронних ресурсів дозволив підсумувати, що використання JWT токенів для автентіфікації у веб-додатках має ряд переваг у порівнянні з варіантом, що передбачає зберігання на сервері сесій, особливо у випадку розподіленої сервісної архітектури.

Незважаючи на суттєві переваги JWT автентіфікації, для забезпечення безпеки системи в цілому необхідно врахувати: захист від XSS атак при реалізації клієнта; вибор надійних механізмів валідації та використання складних ключів шифрування при створенні та обробці токенів в серверній частині.

Проведений аналіз методів захисту авторизації з використанням веб-токену JSON дозволив виокремити такі рекомендації для мінімізації ймовірності атак при використанні JWT: токени бажано використовувати з коротким терміном дії; токени мають бути дійсними лише кілька хвилин; не рекомендується використовувати JWT для постійних, довготривалих даних; рекомендується використовувати під час

передачі токенів захищене з'єднання та не передавати в токенах чутливі дані користувача; використовувати ключові фрази великої довжини, що складаються з великих і малих літер латинського алфавіту, цифр і спецсимволів, та зберігати їх у суворій конфіденційності; забезпечити періодичну зміну ключової фрази; використовувати білий список дозволених алгоритмів підпису. Але ці рекомендації не вирішують проблеми повторного використання викраденого токену.

В роботі запропоновано метод авторизації з використанням веб-токену JSON з додатковим сервісом управління доступом за допомогою бази даних Redis. Даний метод дозволяє вирішити проблему повторного використання викраденого токену.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ МЕТОДУ АВТОРИЗАЦІЇ В ПРОЕКТІ «SPILNO»

3.1 Опис ключових варіантів використання в проекті «Spilno»

В сучасних реаліях нашої держави проект «Spilno» є затребуваним застосунком. Застосунок спрямований на покращення комунікації між особами, якім потрібна допомога та особами, які готові надати свої ресурси для вирішення нагальних потреб населення.

На рисунку 3.1 показано діаграму послідовності використання застосунку «Spilno» .

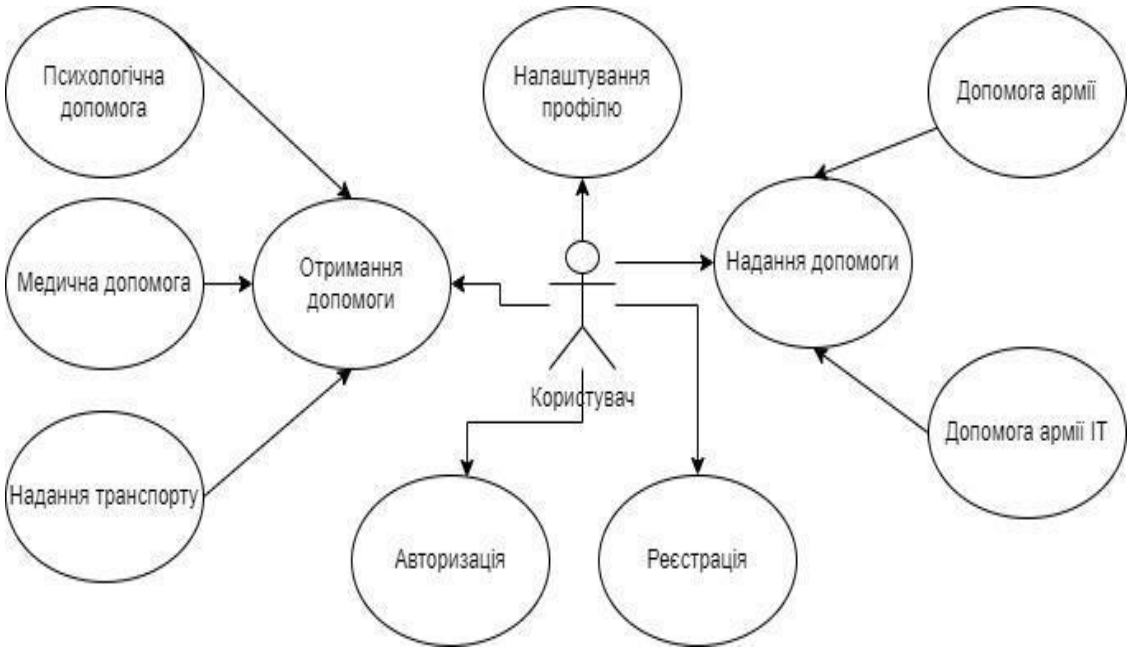


Рисунок 3.1 – Варіанти використання

Основними варіантами використання для користувача є «Авторизація», «Реєстрація», «Налаштування профіля», «Надання допомоги» та «Отримання допомоги».

«Отримання допомоги» в свою чергу розділяється на «Надання транспорту», «Медична допомога», «Психологічна допомога». Варіант використання «Надання допомоги» поділяється на «Допомога армії», «Допомога армії ІТ»

3.2 Інтерфейс системи

При запуску додатку відкривається сторінка з вкладкою авторизації в системі. В програмі передбачено можливість здійснювати двухфакторну автентифікацію та автентифікацію через соціальні мережі.

Приклади екранів авторизації у додатку показані на рисунку 3.2. У додатку передбачена можливість редагувати профіль користувача.

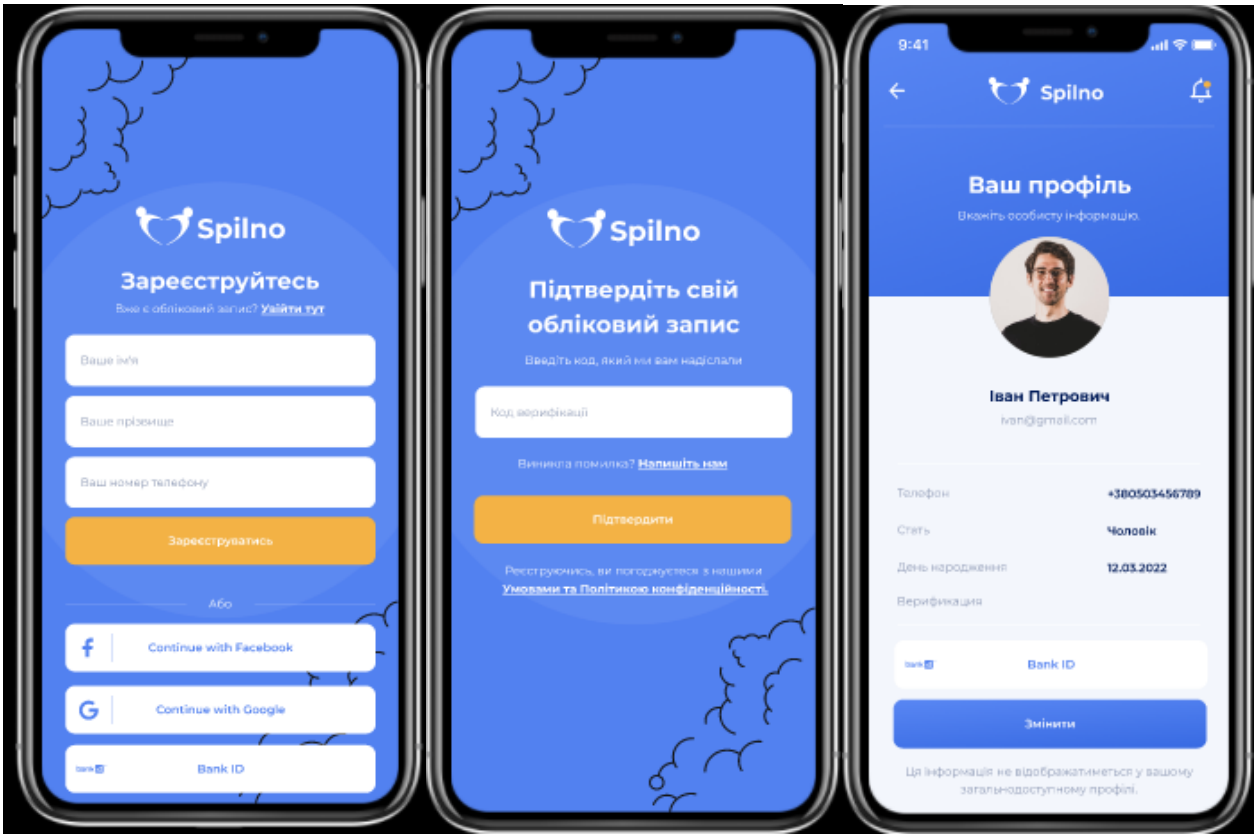


Рисунок 3.2 – Авторизація у додатку

Пройшовши автентифікацію в додатку буде відображено основні функції застосунку. Інтерфейс показано на рисунку 3.3 – 3.4.

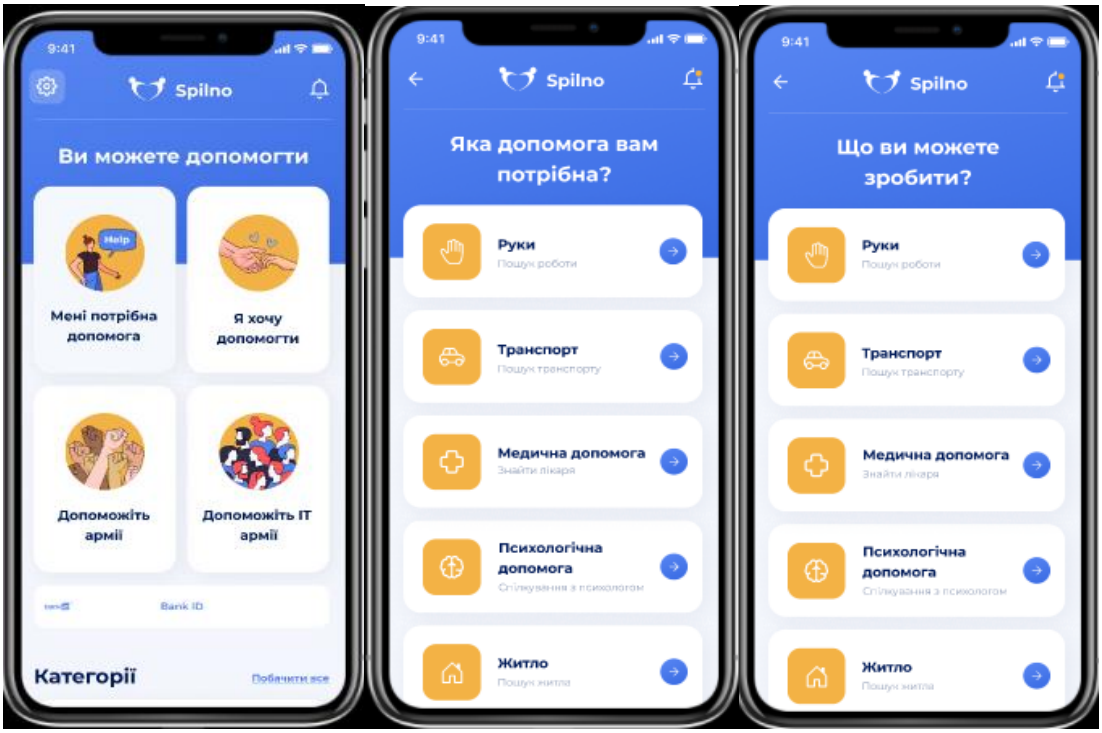


Рисунок 3.3 – Основний функціонал

Інтерфейс інших вкладок зображено на (рисунку 3.4.)

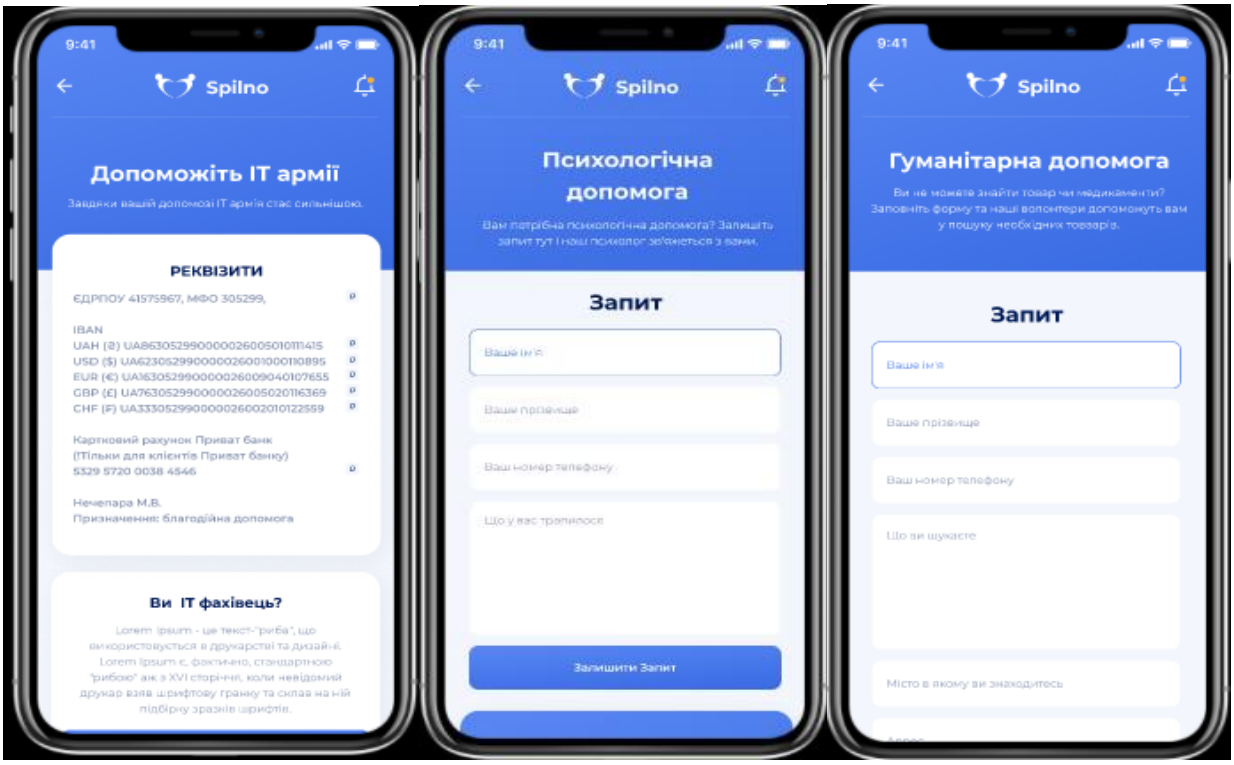


Рисунок 3.4 –Функціонал системи

Загалом, інтерфейс користувача відносно простий у використанні і дозволяє налаштувати систему з максимальною простотою та гнучкістю.

3.3 Реалізація методу авторизації в проекті «Spilno»

Розглянемо основні етапи реалізації запропонованого методу JWT автентифікації на прикладі інформаційної системи «Spilno».

В додатку використовується розподілена сервісна архітектура, основними елементами якої є сервер автентифікації, а також сервіси інших послуг. Застосування сервера автентифікації дозволяє знизити навантаження на послуги та підвищити цим швидкість їх роботи.

При зверненні до сервісу користувач може мати токен, що діє, з справжнім цифровим підписом. Перед обробкою запиту сервіс перенаправляє користувача на сервер автентифікації для здійснення входу в систему і отримання дійсного JWT токена.

Розглянемо докладніше елементи архітектури такої системи щодо реалізації JWT автентифікації на сервері і на клієнті.

У серверній частині програми повинні бути реалізовані як мінімум три функціональні блоки:

- блок реєстрації користувача;
- блок обробки входу користувача до системи;
- блок обробки запиту до захищеної інформації.

OAuth і JWT є двома найбільш широко використовуваними токен-фреймворками або стандартами для авторизації доступу до REST API.

OAuth дозволяє програмі отримувати обмежений доступ до служби HTTP. Хоча JWT є компактним, безпечним для URL-адреси засобом представлення претензій, які підлягають передачі між двома сторонами.

Щоб отримати токен, користувач повинен пройти автентифікацію або ввійти в систему. Типовий потік автентифікації показаний на схемі послідовності нижче.

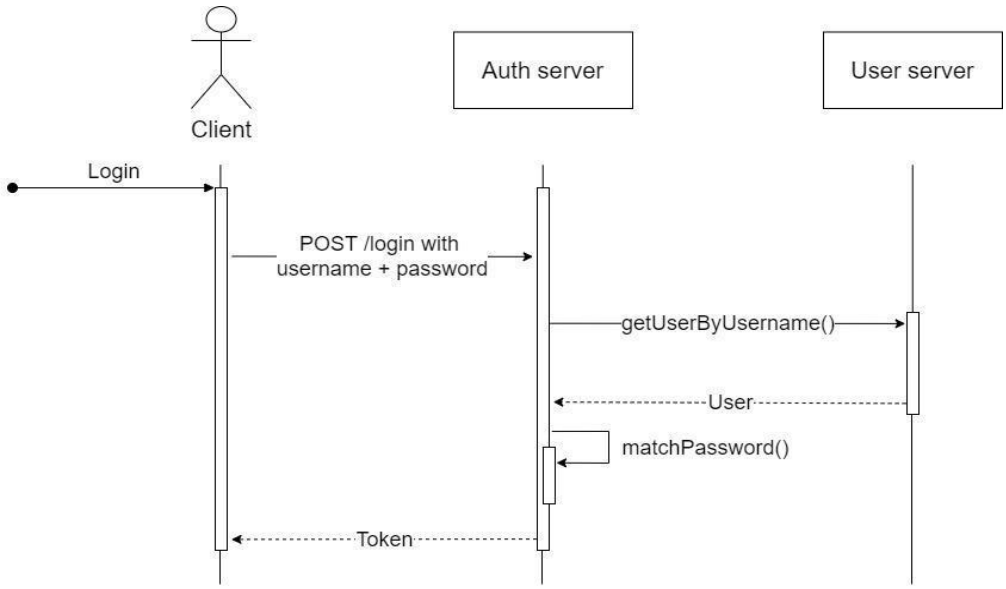


Рисунок 3.5 – Діаграма послідовності для OAuth

Користувач буде вводити своє ім’я користувача та пароль через клієнт (який може бути мобільним пристроєм або ПК), а в кінці процесу автентифікації користувачу буде надано токен. Потім клієнт буде включати токен у кожен наступний запит API до сервера ресурсів (наприклад, сервера користувачів).

Розглянемо приклад, коли здійснено запит API, для отримання автентифікованого користувача. Потік даних OAuth для отримання автентифікованого користувача з ідентифікатором 123 зазвичай виглядатиме як діаграма послідовності, наведена нижче.

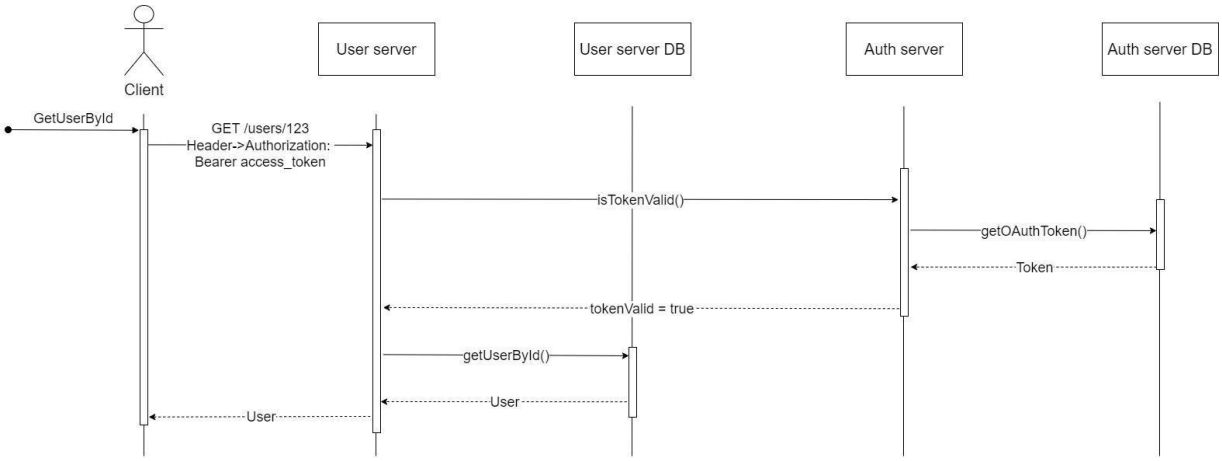


Рисунок 3.6 – Діаграма послідовності для OAuth

У той час як потік даних JWT для отримання автентифікованого користувача, ідентифікатор якого дорівнює 123, зазвичай буде виглядати як діаграма послідовності, наведена нижче.

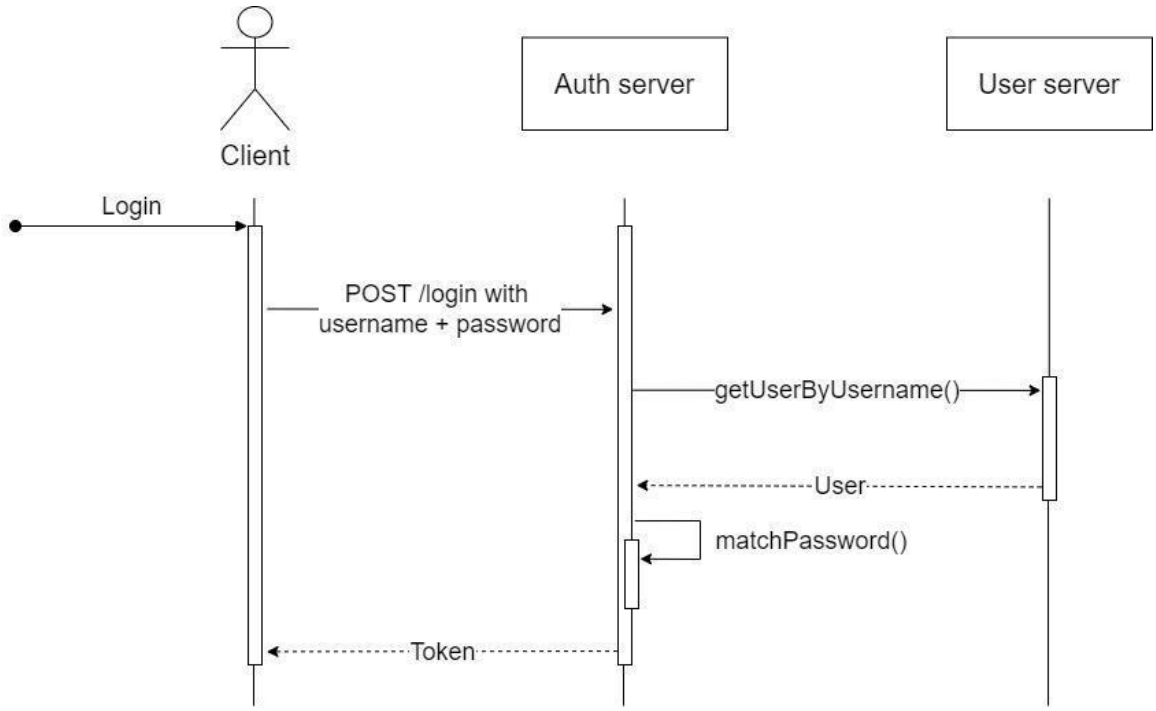


Рисунок 3.7 – Діаграма послідовності для JWT користувача

OAuth більш важкий у порівнянні з JWT. Це пояснюється тим, що OAuth вимагає, щоб сервер Auth перевіряв дійсність токена, а сервер Auth, у свою чергу, покладається на інформацію, яку він зберіг у базі даних, щоб зробити таке рішення.

Проте OAuth має перевагу перед JWT, оскільки токени можна легко відкликати. Це особливо хороша функція, якщо потрібно миттєво скасувати доступ.

Основна відповідь токена OAuth показана нижче.

```
{
   "access_token": "foo",
   "token_type": "приклад",
   "expires_in": 3600,
   "refresh_token": "бар"
}
```

Відповідні атрибути описані нижче.

- access_token Короткочасний токен, який зазвичай діє близько години.
- expires_in Тривалість життя токена доступу в секундах.
- refresh_token Довговічний токен, який використовується для отримання нових токенів доступу. Зазвичай діє протягом кількох днів або місяців.

Клієнт буде включати короткочасний JWT для кожного виклику до сервера ресурсів і використовуватиме довготривалий JWT для отримання нового токена доступу.

Короткочасний JWT перевіряється локально. Однак потрібно зберігати запис або чорний список відкликаних токенів оновлення до закінчення терміну їх дії. Цей чорний список буде перевірено лише тоді, коли клієнт забажає оновити токен OAuth.

Новий токен доступу не буде надано, якщо токен оновлення знайдено в чорному списку. В ідеалі чорний список повинен містити токени оновлення, пов’язані з користувачами, які вийшли з системи, і користувачами, обліковий запис яких було вимкнено. Хоча токен доступу не скасовується відразу, він має бути короткочасним токеном.

Можуть бути сценарії, коли така поведінка є небажаною, але в кінцевому підсумку це залежить від вимог системи. Цей підхід дозволяє серверу ресурсів перевіряти доступ OAuth.

Node.js - це крос-платформне середовище виконання JavaScript з відкритим вихідним кодом, яке виконує код JavaScript поза браузером. В основному JavaScript використовується для створення сценаріїв на стороні клієнта, які вбудовані в HTML-код сторінки веб-сайту та запускаються движком JavaScript у браузері [3].

За допомогою Node.js було використовано JavaScript для отримання інструментів командного рядка. Таким чином, Node.js уособлює концепцію “JavaScript скрізь”, дозволяючи розробляти веб-додатки однією мовою, як для сценаріїв з боку сервера, так і з боку клієнта.

Класичний протокол передачі гіпертексту є інструментом без стану. Це означає, що кожен запит, надісланий від одного клієнта, інтерпретується веб-сервером незалежно і не пов’язаний з будь-яким іншим запитом. Немає вбудованого

механізму для сервера, щоб запам’ятати певного користувача з різних запитів, що також унеможливлює для сервера знати, чи кожен запит походить від одного користувача.

Відстеження сеансів дає змогу відстежувати прогрес користувача на кількох серверах або HTML-сторінках, які за своєю природою не мають стану. Сеанс визначається як серія пов’язаних запитів браузера, які надходять від одного клієнта протягом певного періоду часу. Тут використовується пам’ять (постійне сховище з одним сервером, що не реплікується) для керування сеансами. Коли використовується сховище на основі пам’яті, вся інформація про сеанс зберігається в пам’яті та втрачається під час зупинки та перезапуску. Краще використовувати зовнішнє постійне сховище.

Можна використовувати проміжне програмне забезпечення експрес-сеансу для керування сеансами в Node.js. Сеанс зберігається на самому експрес-сервері. Сховище сеансів на стороні сервера за замовчуванням, MemoryStore, навмисне не призначене для виробничого середовища. Він не масштабується і призначений для налагодження та розробки.

Щоб керувати кількома сеансами для кількох користувачів, необхідно створити глобальну карту та помістити на неї кожен об’єкт сеансу. Глобальні змінні в Node. js споживають пам’ять і можуть виявитися жахливими дірами в безпеці в проектах виробничого рівня. Це можна вирішити за допомогою зовнішнього сховища сеансів. Необхідно зберігати кожен сеанс у сховище, щоб кожен належав лише одному користувачеві. Побудувати таке сховище можливо за допомогою Redis.

Для підвищення безпеки авторизації автором пропонується зберегти токен у власній базі даних, щоб прив’язати його до користувача. Враховуючи, що пошук токена відбувається під час кожного запиту, гарною альтернативою є збереження токену в якомусь сховищі, наприклад Redis. У такому разі у токені можна зберігати більше корисної інформації для керування доступом. Новий потік із вбудованим сховищем та підтримкою перевірки зображено на рисунку 3.8:

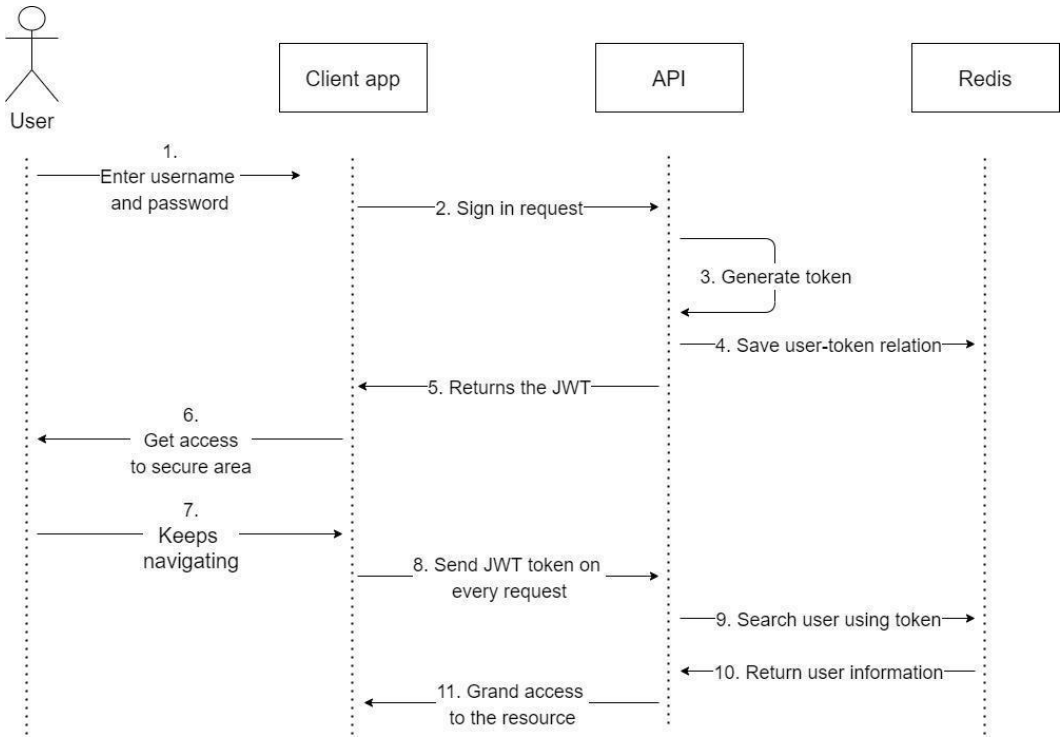


Рисунок 3.8 − Авторизація з використанням веб-токену JSON з додатковим сервісом управління доступом за допомогою Redis

Реалізація цього підходу: Користувач здійснює вхід до програми, ввівши ім’я користувача та пароль. Клієнтська частина надсилає запит до API або до серверу Auth. Після надсилання сервер генерує токен JWT. Токен JWT зберігається в сховищі Reids. Токен JWT надсилається назад користувачеві. Користувачу надається доступ за результатом перевірки токену JWT. Користувач заходить до програми або здійснює навігацію в застосунку. Токен JWT передається кожен раз при здійсненні запиту. Сервер перевіряє токен JWT в сховище Redis. Якщо для даного користувача вже існує токен, то старий видаляється та створюється новий токен користувача і буде здійснено перевірку додаткових даних користувача. Користувач отримує доступ до ресурсів.

Така схема передбачає використання лише одного токену для одного користувача, що значно зменшує ймовірність використання токену зловмисником.

В основі JWT лежать криптографічні механізми, які визначаються веб-підписом JSON (JWS) [33], веб-шифруванням JSON (JWE) [34] і веб-алгоритмами JSON (JWA) [35].

Реалізація серверної частини JWT автентифікації доступна у багатьох мовах програмування та фреймворках. Як компонент сервера автентифікації було використано веб-сервер, написаний мовою JavaScript з використанням фреймворку Express [13] та бібліотеки jsonwebtoken [16].

При реєстрації нового користувача сервер запам'ятовує його дані та генерує на їх основі токен:

```
const jwt = require('jsonwebtoken')
…
app.post('/register', (req, res) => {
…
const token = jwt.sign({user}, process.env.SECRET)
res.json({
token,
…})
…})
```

Тут у змінній user зберігається посилання на об'єкт, що містить дані користувача, введені під час реєстрації, а змінної оточення SECRET сервер зберігає секретний ключ та використовує його при генерації JWT токенів.

Отриманий таким чином JWT токен відразу може бути переданий на клієнтську частину, якщо передбачено автоматичну автентифікацію нових користувачів. Якщо користувач здійснює вхід у додаток, то на стороні сервера виконується схожий сценарій, з тією різницею, що для генерації токена використовується зчитана з бази даних інформація про користувача, що відповідає даним, введеним при вході в систему.

```
app.post('/login', (req, res) => {
…
const token = jwt.sign({userInfo}, process.env.SECRET)
res.json({
token,
…})
```

```
…})
```

Тут у змінній userInfo зберігається посилання на об'єкт, що містить дані користувача, знайдені в базі даних за ключовою інформацією, введеною користувачем при вході в систему. При надходженні запиту до захищених даних сервісу він зчитує із заголовка http запиту з ім'ям Authorization значення JWT токена:

Потім проводиться верифікація токена та отримання з нього корисної інформації у розшифрованому вигляді.

```
jwt.verify(bearerToken, process.env.SECRET,
(err, decoded) => {
…})
…})
```

У змінній decoded у разі успішної валідації повертається посилання на розшифровану корисну інформацію. На основі даних формується відповідь сервера. Якщо перевірка JWT токена і корисної інформації, що міститься в ньому, пройдена успішно, сервіс пересилає клієнту запитані захищені дані. В іншому випадку користувач перенаправляється на сервер автентифікації для входу в систему.

Розглянемо, що має бути реалізовано в клієнтській частині програми, що використовує JWT автентифікацію та веб-браузер. Після успішного входу в систему клієнт отримує відповідь від сервера автентифікації JWT токен.

Цей токен повинен бути збережений в клієнтському оточенні для подальшого використання при API запитах до захищених даних сервісів.

Можна використовувати різні варіанти зберігання такого роду даних користувача. Наприклад, токен може бути збережений у даних самої програми:

```
state.user = userData // {name, token}
```

При такому варіанті зберігання токен буде доступний у контексті (state) поточного стану програми до закриття. Якщо потрібно зберегти токен у проміжках між запусками програми, можна використовувати сховище браузера local storage:

```
localStorage.setItem('user',
JSON.stringify(userData))
```

У цьому випадку при повторному запуску програми можна здійснити автоматичний вхід користувача з використанням даних, прочитаних зі сховища:

userData = localStorage.getItem('user')

За наявності даних клієнтської частини системи JWT токена, отриманого або з сервера, або зі сховища браузера, у всіх запитах до захищених даних сервера необхідно встановлювати http заголовок Authorization. Значення цього заголовка буде рядок, що містить сам токен і ключове слово Bearer.

Якщо для здійснення http-запитів використовується поширена бібліотека для здійснення асинхронної клієнт-серверної взаємодії axios, встановити необхідний заголовок можна так: axios.defaults.headers.common['Authorization'] = `Bearer ${userData.token}`

Важливо передбачити можливість видалення інформації користувача з локального сховища браузера. Така функціональність буде потрібна для обробки виходу користувача з системи, а також при отриманні повідомлення від сервера про те, що токен не є дійсним.

Очистка інформації користувача в локальному сховищі браузера може бути проведена таким чином:

localStorage.removeItem('user').

Код авторизації для застосунку «Spilno» наведено у додатку А.

Висновок до третього розділу

Розроблено частину сервісів у застосунку «Spilno». Застосунок «Spilno» є колективною розробкою, яка спрямована на покращення комунікації між особами, якім потрібна допомога та особами, які готові надати свої ресурси для вирішення нагальних потреб населення та військових у період повномасштабної війни в Україні.

Розроблено діаграми варіантів використання застосунку «Spilno», інтерфейс застосунку та сервіс авторизації.

Побудовано та реалізовано систему авторизації в застосунку «Spilno» на основі запропонованого методу авторизації з використанням вебтокену JSON з додатковим сервісом управління доступом за допомогою Redis. Система авторизації не дозволяє повторного використання викраденого токену і є більш захищеною ніж класична.

Система авторизації може модернізуватися та може бути використана для механізму надання прав доступу для конкретних користувачів чи груп користувачів, що призведе до ще більшого підвищення захисту програмної системи.

ВИСНОВКИ

Відповідальним завданням при розробці різних веб і корпоративних додатків від якого залежить безпека даних є реалізація авторизації. Одним з найпопулярніших методів авторизації в різних клієнт-серверних програмах є токени авторизації. Наразі в інформаційних системах використовуються такі токени: Simple Web Token (SWT), Security Access Markup Language (SAML), JSON Web Token (JWT).

Веб-токен Json (JWT) забезпечує дуже ефективний і масштабований в Інтернеті спосіб визначення ідентичності користувачів між різними сторонами. Це допомагає сторонам, які довіряють, передавати перевірку ідентифікаційних даних користувача на аутсорсинг постачальнику ідентифікаційних даних, щоб довірена сторона могла зосередитися на наданні бажаної послуги і не турбуватися про надання автентифікації користувача. Це ідеально підходить у світі мікросервісів, де робота з автентифікації передана мікрослужбі перевірки ідентичності. Переваги JWT полягають в тому, що він не має приналежності, і, отже, довірена сторона може перевірити токен JWT, не звертаючись до постачальника ідентифікаційних даних. Це пов’язано з тим, що JWT має вбудовані функції безпеки, завдяки яким він забезпечує автентифікацію відправника та захист цілісності. Однак у JWT також є багато недоліків. Нажаль, після видачі токена JWT немає тривіального способу відкликати токен.

Проведений аналіз методів захисту авторизації з використанням веб-токену JSON дозволив виокремити такі рекомендації для мінімізації ймовірності атак при використанні JWT: токени бажано використовувати з коротким терміном дії; токени мають бути дійсними лише кілька хвилин; не рекомендується використовувати JWT для постійних, довготривалих даних; рекомендується використовувати під час передачі токенів захищене з'єднання та не передавати в токенах чутливі дані користувача; використовувати ключові фрази великої довжини, що складаються з великих і малих літер латинського алфавіту, цифр і спецсимволів, та зберігати їх у суворій конфіденційності; забезпечити періодичну зміну ключової фрази;

використовувати білий список дозволених алгоритмів підпису. Але ці рекомендації не вирішують проблеми повторного використання викраденого токену.

В роботі запропоновано метод авторизації з використанням веб-токену JSON з додатковим сервісом управління доступом за допомогою бази даних Redis. Даний метод дозволяє вирішити проблему повторного використання викраденого токену.

Побудовано та реалізовано систему авторизації в застосунку «Spilno» на основі запропонованого методу авторизації з використанням вебтокену JSON з додатковим сервісом управління доступом за допомогою Redis. Система авторизації не припукає повторного використання викраденого токену і є більш захищеною ніж класична.

Система авторизації може модернізуватися та може бути використана для механізму надання прав доступу для конкретних користувачів чи груп користувачів, що призведе до ще більшого підвищення захисту програмної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. dWTV. Learn how to revoke JSON Web Tokens, July 2017. – [Електронний ресурс] Режим доступу https: //developer.ibm.com/tv/learn-how-to-revoke-json-web-tokens/.

2. Бакунова О. М. и др. Stateless авторизация с использованием JWT //Web of Scholar. – 2018. – Т. 1. – №. 6. – С. 10-13.

3. Ethelbert O. et al. A JSON token-based authentication and access management schema for Cloud SaaS applications //2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, 2017. С. 47-53.

4. Jones M. B. The emerging JSON-based identity protocol suite //W3C workshop on identity in the browser. – 2017. – С. 1-3.

5. Jánoky L. V., Ekler P., Levendovszky J. Evaluating the Performance of Novel JWT Revocation Strategy //Acta Cybernetica. – 2021. – Т. 25. – №. 2. – С. 307-318

6. Jones M. B. et al. JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants. – 2016.

7. Solapurkar P. Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario //2016 2nd International Conference on Contemporary Computing and Informatics (IC3I). – IEEE, 2016. С. 99-104.

8. Hardt, Dick and Jones, Michael. The OAuth 2.0 authorization framework: Bearer token usage. – [Електронний ресурс] Режим доступу https://tools.ietf.org/html/rfc6750.

9. Singh J., Chaudhary N. K. OAuth 2.0: Architectural design augmentation for mitigation of common security vulnerabilities //Journal of Information Security and Applications. – 2022. – Т. 65. – С. 103091.

10. Morkonda S. G., Chiasson S., van Oorschot P. C. Empirical Analysis and Privacy Implications in OAuth-based Single Sign-On Systems //Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society. – 2021. – С. 195-208.

11. Arlitt M. Characterizing web user sessions. ACM SIGMETRICS Performance Evaluation Review, 28(2):50-63, 2000. DOI: 10.1145/362883.362920.

12. Auth0 Inc. Revoke tokens. – [Електронний ресурс] Режим доступу https://auth0.com/docs/.

13. Chamith M. Session Management in Nodejs Using Redis as Session Store. //Medium. – 2020. – [Електронний ресурс] Режим доступу https://medium.com/swlh/session-management-in-nodejs-using-redis-assession-store-64186112aa9

14. Goel A. Access Control and Authorization Techniques wrt Client Applications //Data Intelligence and Cognitive Informatics. – Springer, Singapore, 2022. – С. 23-44.

15. J´anoky, L´aszl´o Viktor, Levendovszky, J´anos, and Ekler, P´eter. An analysis on the revoking mechanisms for JSON Web Tokens. International Journal of Distributed Sensor Networks, 14(9), September 2018. DOI: 10.1177/1550147718801535.

16. JsonWebToken implementation for node.js. [Електронний ресурс] – https://github.com/auth0/node-jsonwebtoken.

17. KirstenS. Cross Site Scripting (XSS) Software Attack. “OWASP Foundation” [Електронний ресурс] – https://owasp.org/www-community/attacks/xss/

18. LeBlanc J. Identity and Data Security for Web Development: Best Practices / Jonathan LeBlanc., 2016. – 204 с. – ("O'Reilly Media, Inc."). – (ISBN- 10 : 1491937017).

19. M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian , J. Nazario, “Automated Classification and Analysis of Internet Malware,” In Proceedings of the 10th international conference on Recent advances in intrusion detection (RAID 2007), pp. 178-197, September 2007.

20. Martin T. Identification vs Authentication Authorization [Електронний ресурс] / Thoma Martin. – 2020. – Режим доступу до ресурсу: https://medium.com/plain-and-simple/identification-vs-authentication-vs- authorization-e1f03a0ca885.

21. McLean T. Critical vulnerabilities in JSON Web Token libraries. [Електронний ресурс] – https://www.chosenplaintext.ca/2015/03/31/jwt-algorithm-confusion.html

22. Melton R. Securing a Cloud-Native C2 Architecture Using SSO and JWT //2021 IEEE Aerospace Conference (50100). – IEEE, 2021. – С. 1-8.

23. Okta Inc. Self-encoded access tokens. https://www.oauth.com/oauth2-servers/access-tokens/self-encoded-access-tokens/.

24. Password Authentication for Web and Mobile Apps: The Developer's Guide To Building Secure User Authentication, 2020. – 142 с. – (Independently published). – (ISBN 979-8649303095).

25. Pontarelli, Brian. Revoking JWTs & JWT expiration. https://fusionauth.io/learn/expert-advice/tokens/revoking-jwts/.

26. Rasyada N. SHA-512 Algorithm on Json Web Token for Restful Web Service-Based Authentication //Journal of Applied Data Sciences. – 2022. – Т. 3. – №. 1. – С. 33-43

27. RFC 2906 AAA Authorization Requirements. [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc2906

28. RFC 2989. Criteria for Evaluating AAA Protocols for Network Access. [Електронний ресурс] – Режим доступу: https://www.rfc-editor.org/rfc/pdfrfc/rfc2989.txt.pdf

29. RFC 3539 AAA Transport Profile. [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc3539

30. RFC 4648 The Base16, Base32, and Base64 Data Encodings. [Електронний ресурс] – Режим доступу: https://www.ietf.org/rfc/rfc4648.txt

31. RFC 4962. Guidance for Authenticaton, Authorization, and Accounting (AAA) Key Management. [Електронний ресурс] – https://tools.ietf.org/html/rfc4962

32. RFC 6749. The OAuth 2.0 Authorization Framework. [Електронний ресурс] – https://datatracker.ietf.org/doc/html/rfc6749

33. RFC 7515. JSON Web Signature (JWS). [Електронний ресурс] – https://datatracker.ietf.org/doc/html/rfc7515

34. RFC 7516. JSON Web Encryption (JWE). [Електронний ресурс] – https://tools.ietf.org/html/rfc7516

35. RFC 7518. JSON Web Algorithms (JWA). [Електронний ресурс] – https://datatracker.ietf.org/doc/html/rfc7518

36. RFC 7519. JSON Web Token (JWT). [Електронний ресурс] – https://tools.ietf.org/html/rfc7519

37. RFC 8725. JSON Web Token Best Current Practices. [Електронний ресурс] – https://datatracker.ietf.org/doc/html/rfc8725

38. Sakimura, Natsuhiko, Bradley, John, Jones, Mike, De Medeiros, Breno, and Mortimore, Chuck. OpenID connect core 1.0. The OpenID Foundation, page S3, 2014.

39. Session Management in Nodejs Using Redis as Session Store. – [Електронний ресурс] – Режим доступу: https://medium.com/swlh/session-management-in-nodejs-using-redis-as-session-store-64186112aa9

40. Slipachuk L., Toliupa S., Nakonechnyi V. The Process of the Critical Infrastructure Cyber Security Management using the Integrated System of the National Cyber Security Sector Management in Ukraine //2019 3rd International Conference on Advanced Information and Communications Technologies (AICT). – IEEE, 2019. – С. 451-454

41. Toliupa S. et al. Building an Intrusion Detection System in Critically Important Information Networks with Application of Data Mining Methods //2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET). – IEEE, 2022. – С. 128-133.

42. Wang R., Chen S., Wang X. F. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services //2012 IEEE Symposium on Security and Privacy. – IEEE, 2012. – С. 365-379.

43. ДСТУ ISO/IEC TR 13335-3:2003 «Інформаційні технології. Настанови з керування безпекою інформаційних технологій (IT). Частина 3. Методи керування захистом IT» (ISO/IEC TR 13335- 3:1998, IDT) – [Електронний ресурс] – Режим доступу: http://metrology.com.ua/download/iso-iec-ohsas-i-dr/61-iso/290-dstu-isoiec-tr-13335-4-2005.

44. Класифікація автоматизованих систем та стандартні функціональні профілі захищеності від НСД [Електронний ресурс] – Режим доступу: http://dstszi.kmu.gov.ua/dstszi/control/uk/publish/.
article?showHidden=1&art_id=101870&cat_id=89734&ctime=13445010 89407

45. Лінецький А. Підвищення ефективності авторизації за рахунок використання веб-токену JSON // Матеріали V Міжнародної науково-практичної

конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем” (PCSITS)” 14 - 15 КВІТНЯ 2022, КИЇВ, Україна

46. Макаров Д. А. Механизм авторизации с использованием технологии JWT //Теория и практика современной науки. – 2020. – №. 1. – С. 474-476.

47. Матенко А., Современные методы защиты от киберугроз: Взгляд интегратора [Електронний ресурс].– Режим доступу: http://www.itbiz.ua/prezentaczii.html.

48. Міжнародний стандарт ISO/IEC 27037:2012 «Інформаційні технології. Методи забезпечення безпеки. Настанови щодо ідентифікації, збору, придбання і збереження цифрових даних» [Електронний ресурс]. – Режим доступу: http://www.iso.org/iso/catalogue_detail?csnumber=44381.

49. Офіційний сайт Державної служби спеціальних телекомунікаційних систем і захисту інфор-мації. Електронний ресурс: «Біла книга Держспецзв’язку». — http://www.dstszi.gov.ua/dstszi/ — 47 с.

50. Приложение. Net Core Web API с использованием JWT-токенов для авторизации //Постулат. – 2020. – №. 1.

51. Прокопович-Ткаченко Д. І. Удосконалення методу авторизації та автентифікації безпроводового доступу для підвищення безпеки телекомунікаційних систем та мереж. Системи озброєння і військова техніка : наук. журн.;2013, № 2, С. 141-145

52. Яциковська У.О. Модель захищеної архітектури клієнт–сервер / У.О.Яциковська, І.В.Васильцов, М.П.Карпінський // Вісник Східноукраїнського національного університету імені Володимира Даля. – 2010. – № 9 [151]. – С. 74–79. – ISSN 1998–7927.

53. Яциковська У.О. Удосконалена система захисту комп’ютерної мережі на підставі асиметричного шифрування / У.О.Яциковська, М.М.Касянчук, Р.Б.Трембач // Вісник Східноукраїнського національного університету імені Володимира Даля. – 2009. – № 6 [136], Частина 1. – С. 45-48. – ISSN 1998–7927.

ДОДАТОК А

Лістинг програми

service.ts

Injectable() export class AuthService {

constructor(

private userService: UserService,

private jwtService: JwtService,

private static cypherPassword(password: string, salt: string): string {

return pbkdf2Sync(password, salt, 10000, 64, 'sha256').toString('hex');

* Hash password using KDF with sha256 and random salt

* @param password password to hash

* @return array pair of hashed password and salt

hashPassword(password: string): [string, string] {

const salt = randomBytes(16).toString('hex');

return [AuthService.cypherPassword(password, salt), salt];

* Authenticate user by its' e-mail and password

* @param email email of user

* @param password password of user

* @return number id of authenticated user

async authenticateUser({ email, password }: AuthDto): Promise<User> {

const user = await this.userService.findOneByEmail(email); if (!user) {

throw new UserNotFoundException();

```
const hashedPassword = AuthService.cypherPassword(
password,
user.password_salt,
;
if (hashedPassword !== user.password_hash) {
throw new InvalidPasswordException();

return user;
```

ДОДАТОК Б

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ

Тези наукових доповідей

1. Наконечний В.С., Лінецький А.П. Підвищення ефективності авторизації з використанням веб-токену JSON. V Міжнародна науково-практична конференція “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”, PCSITS 2022, Київ.