МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«____»_____ 2023 р.

На правах рукопису
УДК 004.056.5:510.22(043.3)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний модуль автентифікації користувачів з використанням непрозорих токенів

Виконавець: Євгеній ЯКИМЧУК
Керівник: к.т.н., доцент Анна ІЛЬЄНКО
Консультант розділу «Охорона навколишнього середовища»: к.т.н., доцент Тетяна ДМИТРУХА
Нормоконтролер: к.т.н., доцент Анна ІЛЬЄНКО

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Магістр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«__» _____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
здобувача вищої освіти Якимчука Євгенія Анатолійовича

1. Тема: *Програмний модуль автентифікації користувачів з використанням непрозорих токенів*
затверджена наказом ректора від «15» вересня 2023 р. № 1814/ст.
2. Термін виконання: з 16.10.2023 р. по 31.12.2023 р.
3. Вихідні дані: реалізувати і описати реалізацію програмного модуля автентифікації користувачів з використанням непрозорих токенів; проаналізувати та дослідити загальні понять щодо автентифікації користувачів в інформаційних мережах, проблеми автентифікації користувачів; описати інтеграцію програмного модулю автентифікації з використанням непрозорих токенів; реалізувати та описати програмний модуль автентифікації з використанням непрозорих токенів;

4. Зміст пояснювальної записки: аналіз та дослідження загальних понять щодо автентифікації користувачів в інформаційних мережах, проблеми автентифікації користувачів; опис інтеграції програмного модулю автентифікації з використанням непрозорих токенів; реалізація і детальний опис реалізації програмного модуля автентифікації користувачів з використанням непрозорих токенів.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

№ з/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	16.10.2023	*Виконано*
2.	Аналіз літературних джерел	20.10.2023	*Виконано*
3.	Обґрунтування вибору рішення	22.10.2023	*Виконано*
4.	Збір інформації	24.10.2023	*Виконано*
5.	Аналіз та дослідження сучасних загальних понять щодо автентифікації користувачів в інформаційних мережах	27.11.2023	*Виконано*
6.	Аналіз загальних проблем автентифікації та їх рішень, опис інтеграції програмного модуля автентифікації користувачів з використанням непрозорих токенів	10.11.2023	*Виконано*
7.	Розробка програмного модуля автентифікації користувачів з використанням непрозорих токенів	19.11.2023	*Виконано*
8.	Оформлення і друк пояснювальної записки	10.12.2023	*Виконано*
9.	Оформлення презентації	11.12.2023	*Виконано*
10.	Отримання рецензій від рецензента	22.12.2023	*Виконано*

6. Консультанти з окремих розділів

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Охорона навколишнього середовища	Дмитруха Т.І.		

7. Дата видачі завдання: «16» жовтня 2023 р.

Здобувач вищої освіти Євгеній ЯКИМЧУК
(підпис, дата)

Керівник кваліфікаційної роботи Анна ІЛЬЄНКО
(підпис, дата)

РЕФЕРАТ

Кваліфікаційна робота на тему: «Програмний модуль автентифікації користувачів з використанням непрозорих токенів» складається зі вступу, основної частини, що містить 3 розділи, 3 висновки до кожного розділу, загального висновку, 3 додатків та списку використаної літератури. Загальний обсяг роботи – 120 сторінок. Список використаних джерел включає 25 джерел.

Метою кваліфікаційної роботи є програмна реалізація програмного модулю автентифікації користувачів з використанням непрозорих токенів.

У кваліфікаційній роботі розглянуті питання щодо загальних понять, проблем та їх вирішення щодо автентифікації користувачів в інформаційних мережах,.

Реалізація власного програмного модулю базується на використанні непрозорих токенів, надійному шифруванню, використанні ролей для кожного користувача та фільтрів, які контролюють вхідний трафік.

Розроблений програмний модуль автентифікації, заснований на використанні непрозорих токенів, представляє сучасний та безпечний метод забезпечення доступу до інформаційних ресурсів. Виявлені переваги даного підходу роблять його важливим компонентом для будь-якої інформаційної системи, яка прагне забезпечити найвищий рівень безпеки та зручності для своїх користувачів.

Ключові слова: МЕРЕЖА, АВТЕНТИФІКАЦІЯ, НЕПРОЗОРІ ТОКЕНИ, БЕЗПЕКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

З М І С Т

ВСТУП

Актуальність теми. У світі стрімкого розвитку інформаційних технологій та зростання обсягів цифрової активності, питання забезпечення безпеки та конфіденційності стає все більш актуальним. Особливу увагу привертає проблема надійності автентифікації користувачів в інформаційних системах. В умовах зростання кількості кіберзагроз та ризиків, пов'язаних з несанкціонованим доступом, розробка нових підходів до автентифікації стає необхідною.

Мета роботи – реалізація програмного модулю автентифікації користувачів з використанням непрозорих токенів.

Об’єкт дослідження – процес автентифікації користувачів в інформаційних системах. Основна увага спрямована на аналіз та вдосконалення методів, що використовуються для перевірки ідентичності користувача.

Предмет дослідження – інформаційні мережі, непрозорі токени, програмний модуль автентифікації. Виходячи з поставленої мети завданнями кваліфікаційної роботи є:

- аналіз та дослідження загальних понять щодо автентифікації користувачів в інформаційних мережах, проблеми автентифікації користувачів;
- опис інтеграції програмного модулю автентифікації з використанням непрозорих токенів;
- реалізація і оцінка доцільності реалізації програмного модуля автентифікації користувачів з використанням непрозорих токенів.

Методи. Проведені дослідження базуються на сучасних методах впровадження автентифікації користувачів, різноманітних фільтрів, які керують трафіком, шифруванням даних та розподіленням ролей доступу між користувачами.

Наукова новизна. Наукова новизна полягає у розробці та практичному впровадженні авторського програмного модулю автентифікації користувачів з

використанням непрозорих токенів, що дозволяє забезпечити стійкість та конфіденційність, а також використання непрозорих токенів дозволяє уникнути атак перехоплення.

Практична цінність. Практична цінність роботи полягає у створенні власного програмного модулю автентифікації користувачів з використанням непрозорих токенів, з використанням мова програмування Java. Розробка програмного модуля автентифікації з використанням непрозорих токенів має практичне значення для різноманітних інформаційних систем, сервісів та платформ. Впровадження цього методу дозволить підвищити безпеку облікових записів користувачів та зменшити ризик несанкціонованого доступу. Проведено тестування з використанням Burp Suit, а саме моніторинг трафіку та перевірку перехоплення і безпечність використання непрозорого токену.

Апробація результатів. 1. Ільєнко А., Якимчук Є. А., Кравчук І. А. Програмний модуль автентифікації користувачів з використанням непрозорих токенів // Modern problems of science, education and society: VIII Міжнародна науково-практична конференція, 9-11 жовтня 2023 р.: тези доповіді. – К., 2023. – С.358-360.

РОЗДІЛ 1

ЗАГАЛЬНІ ПОНЯТТЯ ЩОДО АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ В ІНФОРМАЦІЙНИХ МЕРЕЖАХ

1.1. Сучасний стан та проблеми щодо забезпечення безпеки інформаційних мереж

Мережева безпека відноситься до заходів, які приймаються будь-яким підприємством або організацією для захисту своєї комп'ютерної мережі і даних за допомогою апаратних і програмних систем. Це спрямовано на забезпечення конфіденційності та доступності даних і мережі. Кожна компанія або організація, яка обробляє великий обсяг даних, має певний рівень заходів проти різних кіберзагроз.

Найпростішим прикладом мережевої безпеки є захист паролем, який користувач мережі вибирає самостійно. У останні часи мережева безпека стала центральною темою кібербезпеки, і багато організацій запрошують осіб, які мають навички в цій галузі. Рішення з мережевої безпеки захищають різні вразливості комп'ютерних систем, такі як:

- Користувачі
- Розташування
- Дані
- Пристрої
- Додатки

Переваги безпеки мережі

1. Мережева безпека допомагає захистити інформацію та дані клієнтів, та забезпечує надійний доступ і допомагає захистити дані від кіберзагроз.

2. Безпека мережі захищає організацію від великих втрат, які могли статися через втрату даних або будь-який інцидент безпеки.

3. Захищає репутацію організації, оскільки захищає дані та конфіденційні елементи.

Робота над мережевою безпекою

Основний принцип мережевої безпеки - це захист великих об'ємів збережених даних та мереж шляхом розкладання правил та регуляцій, які повинні бути визнані перед виконанням будь-якої дії з даними. Ці рівні включають:

- Фізичну мережеву безпеку
- Технічну мережеву безпеку
- Адміністративну мережеву безпеку

Фізична мережева безпека: Це найбільш базовий рівень, який включає в себе захист даних та мережі від несанкціонованих осіб, які можуть намагатися отримати контроль над конфіденційністю мережі. Сюди входять зовнішні пристрої та маршрутизатори, які можуть використовуватися для підключення кабелів. Також можна досягти цього за допомогою пристроїв, таких як біометричні системи.

Технічна мережева безпека: Основний акцент робиться на захисті даних, збережених в мережі або даних, які переходять через мережу. Цей тип мережевої безпеки має дві мети. Перша - захист від несанкціонованих користувачів, а друга - захист від зловмисних дій.

Адміністративна мережева безпека: Цей рівень мережевої безпеки захищає поведінку користувачів, таку як надання дозволів та процес авторизації. Він також забезпечує рівень складності, який може бути потрібний мережі для захисту від усіх видів атак. Цей рівень також рекомендує необхідні зміни, які мають бути внесені до інфраструктури.

Типи мережевої безпеки

Контроль доступу: Не кожна особа повинна мати повний доступ до мережі або її даних. Один зі способів перевірки цього полягає в аналізі даних

кожної особи. Це робиться за допомогою системи контролю доступу до мережі, яка забезпечує, що лише обмежена кількість санкціонованих осіб може працювати з допущеною кількістю ресурсів.

Програмне забезпечення антивірусу та програмне забезпечення для захисту від шкідливих програм: Цей тип мережевої безпеки забезпечує, що зловмисне програмне забезпечення не потрапляє в мережу і не ставить під загрозу безпеку даних. Зловмисне програмне забезпечення, таке як віруси, трояни і черви, обробляється цим типом захисту. Це забезпечує не лише захист від потрапляння шкідливого програмного забезпечення, але і готовність системи до боротьби з ним після потрапляння.

Хмарна безпека: В сучасний час багато організацій використовують хмарні технології, де великий обсяг важливих даних зберігається в Інтернеті. Це дуже вразливо перед можливими зловживаннями, які можуть вчинити несанкціоновані користувачі. Дані повинні бути захищені, і це повинно бути гарантовано безпекою. Багато бізнесів використовують програми обліку послуг (SaaS) для надання деяким співробітникам доступу до даних, збережених в хмарі. Цей тип безпеки гарантує створення бар'єрів у видимості даних.

Безпека електронної пошти: Безпека електронної пошти включає в себе послуги та продукти, призначені для захисту облікового запису електронної пошти та його вмісту від зовнішніх загроз. Наприклад, шахрайські листи зазвичай автоматично відправляються у папку "Спам", оскільки більшість постачальників електронної пошти мають вбудовані функції для захисту вмісту.

Брандмауери: Брандмауер - це пристрій мережевої безпеки, який моніторить весь вхідний і вихідний трафік і, відповідно до визначеного набору правил безпеки, приймає, відхиляє або відбиває цей конкретний трафік. Перед введенням брандмауерів мережеву безпеку забезпечували списки контролю доступу (ACL), розташовані на маршрутизаторах.

Безпека додатків: Безпека додатків означає заходи мережевої безпеки, які використовуються на рівні додатків для запобігання витяганню або захопленню даних або коду в додатку. Це також включає заходи безпеки, призначені для захисту додатків під час розробки та проектування, а також техніки і методи для захисту додатків в майбутньому.

Система запобігання вторгнення (IPS): Система запобігання вторгнення, також відома як система виявлення і запобігання вторгненню, є застосунком мережевої безпеки, який моніторить активність мережі або системи на предмет зловмисної діяльності. Головні функції систем запобігання вторгненню полягають в ідентифікації зловмисної діяльності, зборі інформації про цю діяльність, її реєстрації та спроби блокувати або припинити її.

Виклики в області мережевої безпеки інформації

В сучасному цифровому світі мережева безпека інформації стикається з багатьма викликами, з якими повинні боротися організації. Ці виклики значно впливають на конфіденційність, цілісність та доступність даних і ресурсів мережі. Розглянемо деякі ключові виклики:

Загрози змінюються

Загрози постійно змінюються, а кіберпротивники розвивають нові вектори атак і техніки. Ця швидка еволюція ускладнює передбачення та ефективний захист від нових загроз. Серед типових загроз можна виділити:

- **Малікод**: Зловмисне програмне забезпечення, включаючи віруси, черви, трояни та розшифрувальне програмне забезпечення (ransomware), постійно розвивається, що робить виявлення і запобігання постійним викликом.
- **Вразливості нульового дня**: Атаки використовують не вирішенні вразливості, відомі як вразливості нульового дня, які становлять значний ризик, доки їх інженери не впровадять патчі для них.
- **Продвинуті стійкі загрози (Advanced Persistent Threats - APTs)**: APT - це високоступіневі і добре фінансовані зловмисники, які націлені на конкретні організації протягом тривалих періодів, часто залишаючись нерозкритими протягом довгих періодів.

Помилки людини

Незважаючи на технологічний прогрес, помилки людини залишаються значущим чинником випадків порушення безпеки. Загальні виклики безпеки, пов'язані з людьми, включають:

- **Слабкі паролі:** Багато користувачів все ще використовують легко вгадувані паролі, використовують їх на кількох облікових записах або не оновлюють їх регулярно.
- **Атаки фішингу:** Атаки фішингу через електронну пошту і тактики соціальної інженерії обманюють осіб, ведучи до розкриття чутливої інформації або переходу за шкідливими посиланнями.
- **Навмисна витікаюча інформація:** Працівники можуть ненавмисно викладати чутливі дані, неправильно налаштовуючи дозволи або недбало обробляючи інформацію.

Системи застарілих стандартів

Організації часто покладаються на системи застарілих стандартів, які не були розроблені з урахуванням сучасних загроз безпеці. Ці системи можуть бути позбавлені важливих функцій безпеки та отримувати обмежені або ж жодні оновлення безпеки, що робить їх вразливими для атак.

Внутрішні загрози

Внутрішні загрози становлять унікальний виклик, оскільки вони походять з середини організації. Ці загрози можуть включати співробітників, підрядників або бізнес-партнерів, які зловживають своїми привілегіями для зловживання або ненавмисного розкриття чутливої інформації.

Відповідність і регулювання

Організації повинні орієнтуватися в складних умовах вимог регулювання та стандартів відповідності. Ці стандарти, хоча і важливі для захисту даних і приватності, можуть бути складні для впровадження і підтримки в різних галузях та регіонах.

Для вирішення цих викликів необхідний комплексний підхід, який включає в себе технологічні рішення, навчання працівників та зобов'язання до постійного вдосконалення.

1.2. Підходи щодо автентифікації користувачів на базі токенів

Аутентифікація на основі токенів - це протокол, який дозволяє користувачам перевіряти свою ідентичність і отримувати унікальний токен доступу. Протягом часу життя токену користувачі отримують доступ до веб-сайту або додатку, для яких був виданий токен, замість того, щоб перевводити облікові дані кожен раз, коли вони повертаються на ту саму веб-сторінку, додаток або будь-який ресурс, захищений цим самим токеном.

Аутентифікаційні токени працюють подібно до квитка. Користувач має доступ, доки токен залишається дійсним. Якщо користувач вийшов із системи або закрив додаток, токен стає недійсним.

Аутентифікація на основі токенів відрізняється від традиційних методів аутентифікації на основі пароля або сервера. Токени надають другий рівень безпеки, і адміністратори мають детальний контроль над кожною дією та транзакцією.

Аутентифікація та авторизація - це насправді різні, але взаємозалежні поняття в області інформаційної безпеки. До появи аутентифікаційних токенів, для керування доступом, використовували традиційні методи, такі як паролі та аутентифікація на сервері. Проте ці методи часто супроводжувалися своєрідними викликами.

Зазвичай паролі включають в себе:

- **Створення користувачем**. Користувач придумує комбінацію літер, цифр та символів.

• **Запам'ятовування**. Людина повинна запам'ятати цю унікальну комбінацію.

• **Повторення**. Кожного разу, коли користувач потребує доступ до чогось, йому потрібно вводити пароль.

Крадіжка паролів - це звичайне явище. Фактично, однією з перших задокументованих випадків крадіжки паролів сталася ще в 1962 році. Люди не можуть запам'ятати всі свої паролі, тому вони вдаються до різних хитрощів, таких як:

• Запис їх на папері. Аркуші паперу, де записані паролі, є справжніми кошмарами з точки зору безпеки.

• Повторення їх. Люди часто використовують однаковий пароль на кількох ресурсах. Якщо один пароль виявляється, багато облікових записів може бути уразливими.

• Легка зміна. Люди змінюють одну літеру чи цифру, коли їм пропонують змінити пароль.

Паролі також вимагають аутентифікації на стороні сервера. Кожен раз, коли користувач входить у систему, комп'ютер створює запис про транзакцію, що призводить до збільшення обсягу пам'яті на сервері.

Аутентифікація на основі токенів пропонує інший підхід:

У випадку аутентифікації на основі токенів, додатковий сервіс (зазвичай, сервер аутентифікації або постачальник ідентичності) перевіряє запит користувача на доступ до ресурсу на сервері.

Після успішної перевірки сервер видаватиме користувачеві токен, який служить обліковими даними і дає доступ до певних ресурсів.

Токени є більш безпечними і менш схильними до крадіжки в порівнянні з традиційними паролями, оскільки вони, як правило, обмежені за часом або можуть бути використані лише один раз і не вимагають постійного введення паролю користувачем.

Токен-базові сесії також споживають менше ресурсів сервера, оскільки не потрібно зберігати обширні записи кожної сесії на сервері.

Загалом, аутентифікація на основі токенів підвищує безпеку, зменшує залежність від статичних паролів і спрощує процес аутентифікації. Вона пропонує більш надійний і зручний підхід до контролю доступу та аутентифікації, що робить її цінним інструментом в сучасній інформаційній мережевій безпеці.

Всі аутентифікаційні токени дозволяють отримувати доступ, але кожен тип працює трохи по-різному. Існують три загальних типи аутентифікаційних токенів:

Підключені (Connected): Ключі, диски, пристрої та інші фізичні предмети підключаються до системи для отримання доступу. Якщо ви коли-небудь використовували USB-пристрій чи смарт-карту для входу в систему, ви використовували підключений токен.

Безконтактні (Contactless): Пристрій знаходиться достатньо близько до сервера, щоб взаємодіяти з ним, але він не підключений фізично. Прикладом цього типу токена може бути так званий "магічний кільце" від Microsoft.

Відокремлені (Disconnected): Пристрій може взаємодіяти з сервером на великі відстані, навіть якщо він ніколи не контактує з іншим пристроєм. Якщо ви коли-небудь використовували свій телефон для процесу двофакторної аутентифікації, ви використовували цей тип токена.

У всіх цих сценаріях користувач повинен виконати деякі дії для початку процесу. Це може включати введення пароля або відповідь на питання. Але навіть після успішного виконання цих попередніх кроків користувач не може отримати доступ без допомоги токена доступу.

Використовуючи систему аутентифікації на основі токенів, відвідувачі мають перевірити облікові дані лише один раз. Відповідно, вони отримають токен, який надає доступ протягом тимчасового періоду, який ви визначаєте.

Процес працює наступним чином:

• Запит (Request): Користувач просить доступ до сервера чи захищеного ресурсу. Це може включати в себе вхід за допомогою пароля або інший процес, який ви визначаєте.

• Перевірка (Verification): Сервер визначає, чи має користувач доступ. Це може включати в себе перевірку пароля відносно імені користувача або інший процес, який ви визначаєте.

• Токени (Tokens): Сервер взаємодіє з пристроєм аутентифікації, таким як кільце, ключ, телефон чи схожий пристрій. Після перевірки сервер видаватиме токен і передасть його користувачу.

• Зберігання (Storage): Токен знаходиться в браузері користувача, поки триває робота.

Якщо користувач спробує відвідати іншу частину сервера, токен знову взаємодіє з сервером. Доступ надається або відмовляється на підставі токена.

Адміністратори встановлюють обмеження на токени. Ви можете дозволити одноразовий токен, який негайно знищується, коли користувач виходить із системи. Або ви можете налаштувати токен на самознищення в кінці визначеного часового періоду.

Аутентифікаційні токени корисні для адміністраторів систем, які:

Часто надають тимчасовий доступ

Ваша база користувачів змінюється в залежності від дати, часу чи спеціальної події. Повторне надання та скасування доступу є втомлюючими. Токени можуть бути корисними в такому випадку.

Вимагають деталізованого доступу

Ваш сервер надає доступ на основі конкретних властивостей документів, а не властивостей користувача. Паролі не дозволяють такого рівня деталізації. Наприклад, якщо ви керуєте онлайн-журналом і хочете, щоб кожен міг читати та коментувати лише один документ, а не інші, то токени можуть це дозволити.

Становлять привабливі цілі для взлому

Ваш сервер містить конфіденційні документи, які можуть завдати серйозної шкоди вашій компанії, якщо їх розголосити. Простий пароль не надає достатньої захисту. Фізичний пристрій може допомогти значно краще.

Аутентифікаційні токени призначені для підвищення безпеки ваших протоколів та забезпечення безпеки сервера. Але вони не будуть працювати ефективно, якщо не дотримуватися правил:

Приватність (Private): Користувачі не можуть ділитися аутентифікаційними пристроями або передавати їх між відділами. Так само, як вони не повинні ділитися паролями, вони не повинні ділитися будь-якою іншою частиною вашої системи безпеки.

Захищеність (Secure): Комунікація між токеном і вашим сервером повинна бути захищеною за допомогою HTTPS-з'єднань. Шифрування є важливою частиною забезпечення безпеки токенів.

Перевіреність (Tested): Періодично проводьте тестування токенів, щоб переконатися, що ваша система захищена і працює належним чином.

Відповідність (Appropriate): Вибирайте правильний тип токена для вашого конкретного використання. Наприклад, JWT не є ідеальними для сесійних токенів. Вони можуть бути витратними, і ризики безпеки, пов'язані з перехопленням, неможливо повністю усунути.

Таблиця 1.1

Порівняння типів токенів

Тип токена	**Опис**	**Сфери використання**
Апаратні токени	Фізичні пристрої для генерації одноразового паролю	Безпечний доступ до корпоративних мереж
Програмні токени	Мобільні додатки або програми для генерації одноразового паролю	Онлайн-банкінг та віддалений доступ
Біометричні токени	Використання біометричних даних для автентифікації	Розблокуваня смартфонів та біометричних банкоматів

Продовження таблиці 1.1

Токени-носії	Токени, що використовуються для авторизації	Доступ на основі OAuth до веб-ресурсів

Двофакторна аутентифікація (2FA) і багатофакторна аутентифікація (MFA)

Двофакторна аутентифікація (2FA) і багатофакторна аутентифікація (MFA) - це методи аутентифікації, які виходять за межі традиційної однофакторної аутентифікації (зазвичай на основі чогось, що користувач знає, наприклад, пароля) для надання додаткового рівня безпеки. Вони вимагають від користувачів надавати кілька підтверджень для підтвердження своєї ідентичності.

Двофакторна аутентифікація (2FA)

Двофакторна аутентифікація - це процес безпеки, в якому користувач повинен представити два різні фактори аутентифікації для отримання доступу. Ці фактори зазвичай поділяються на три категорії:

- Щось, що ви знаєте: Це може бути пароль, ПІН-код або відповіді на запитання щодо безпеки.
- Щось, що у вас є: Зазвичай це токен, такий як апаратний або програмний генератор токенів або смарт-карта.
- Щось, що ви є: Сюди входять біометричні дані, такі як відбитки пальців, сканування сітківки очей або розпізнавання обличчя.

2FA надає додатковий рівень безпеки, забезпечуючи, що навіть якщо один фактор буде скомпрометований (наприклад, пароль буде вкрадений), зловмиснику все ще потрібний другий фактор для отримання доступу. Це широко використовується в онлайн-банкінгу, електронній пошті та інших чутливих додатках.

Багатофакторна аутентифікація (MFA)

Багатофакторна аутентифікація розширює концепцію 2FA, вимагаючи від користувачів надавати більше двох факторів аутентифікації. Додаткові фактори можуть включати:

- Щось, що ви є: Біометричні дані (наприклад, відбитки пальців, розпізнавання обличчя).
- Щось, що у вас є: Токени або смарт-карти.
- Щось, що ви знаєте: Паролі або ПІН-коди.

MFA значно підвищує безпеку, збільшуючи кількість факторів, необхідних для аутентифікації. Це часто використовується в дуже безпечних середовищах, таких як урядові агентства, фінансові установи та критична інфраструктура.

Переваги 2FA та MFA

- **Підвищена безпека**: 2FA та MFA значно зменшують ризик несанкціонованого доступу, оскільки зловмиснику потрібно скомпрометувати кілька факторів для отримання доступу.
- **Зменшення залежності від паролів**: MFA зменшує залежність від традиційних паролів, які часто вразливі до порушень і атак.
- **Вимоги щодо дотримання**: Багато регуляторних стандартів вимагають використання 2FA або MFA для захисту чутливих даних і систем.

Виклики впровадження 2FA та MFA

- **Зручність використання**: Збалансувати безпеку і зручність використання може бути складно. Складні процеси аутентифікації можуть дратувати користувачів.
- **Витрати**: Апаратні токени та біометричні системи можуть бути дорогими у впровадженні та обслуговуванні.
- **Підготовка користувачів**: Користувачам може знадобитися підготовка, щоб правильно розуміти та використовувати 2FA або MFA.

Протоколи аутентифікації на основі токенів

Протоколи аутентифікації на основі токенів є фундаментальними для забезпечення безпечного доступу до інформаційних мереж. Ці протоколи

використовують різні механізми для перевірки ідентичності користувача і надання доступу. Широко використовувані протоколи аутентифікації на основі токенів:

Одноразовий пароль на основі часу (TOTP)

Одноразовий пароль на основі часу - це широко використовуваний протокол аутентифікації на основі токенів, який використовує поточний час для генерації одноразових паролів.

Механізм TOTP:

Спільний секретний ключ: Користувач і сервер аутентифікації обмінюються спільним секретним ключем (зазвичай у формі криптографічного ключу).

Часовий компонент: Поточний час, зазвичай в секундах від певного початкового моменту (наприклад, Unix-час), служить динамічним компонентом.

Функція хешування: Функція хешування (зазвичай HMAC-SHA1, HMAC-SHA256 або HMAC-SHA512) комбінує спільний секрет і часовий компонент, щоб згенерувати унікальне значення хешу.

Генерація OTP: Зазвичай значення хешу обрізається, щоб створити коротший OTP. Довжина OTP може варіюватися (наприклад, 6 або 8 цифр). Результат - одноразовий пароль на основі часу.

Часове вікно: Як клієнт, так і сервер повинні залишатися синхронізованими щодо часового вікна, яке використовується для генерації OTP. Зазвичай часове вікно встановлено на короткий період, наприклад, 30 секунд.

Використання TOTP:

TOTP широко використовується в сценаріях, які вимагають двофакторної аутентифікації (2FA) або багатофакторної аутентифікації (MFA). Його використання включає:

Онлайн-сервіси: TOTP реалізований в різних онлайн-сервісах, таких як поштові служби, платформи соціальних мереж та фінансові установи, для підвищення безпеки під час входу.

Мобільні додатки: Багато мобільних додатків пропонують аутентифікацію на основі TOTP для збільшення безпеки.

Безпечний доступ до VPN: Організації використовують TOTP для безпечного віддаленого доступу до своїх мереж через віртуальні приватні мережі (VPN).

Менеджери паролів: Деякі додатки для управління паролями використовують TOTP для захисту доступу до збережених паролів.

Безпека TOTP:

Стійкість до атак повторного використання: TOTP стійкий до атак повторного використання, оскільки кожний OTP є часовим і дійсний лише протягом короткого періоду.

Спільний секретний ключ: Спільний секретний ключ між користувачем і сервером є криптографічним ключем, що ускладнює завдання атакуючих для отримання ключа.

Генерація OTP офлайн: Оскільки OTP TOTP можна генерувати офлайн за допомогою спільного секретного ключу та поточного часу, не потрібно постійного підключення до Інтернет, що зменшує ризик перехоплення.

Масштабованість: TOTP дуже масштабований і може використовуватися для великої кількості користувачів без значного навантаження на ресурси.

Однак безпека TOTP значно залежить від безпечного зберігання спільного секрету і підтримання точної синхронізації часу між клієнтом і сервером. Якщо спільний секрет порушено або відбувається значне відхилення часу, це може вплинути на безпеку.

TOTP широко використовується, де необхідна двофакторна аутентифікація (2FA) або багатофакторна аутентифікація (MFA). Його використання включає:

Онлайн-сервіси: TOTP реалізований в різноманітних онлайн-сервісах, таких як поштові провайдери, соціальні мережі та фінансові установи, для підвищення безпеки під час входу.

Мобільні додатки: Багато мобільних додатків пропонують аутентифікацію на основі TOTP для підвищення безпеки.

Безпечний доступ до VPN: Організації використовують TOTP для забезпечення безпечного віддаленого доступу до своїх мереж через віртуальні приватні мережі (VPN).

Менеджери паролів: Деякі додатки для управління паролями використовують TOTP для захисту доступу до збережених паролів.

HMAC-based One-Time Password (HOTP)

HMAC-based One-Time Password - це протокол аутентифікації на основі токенів, який генерує одноразові паролі (OTP) на основі спільного секретного ключа та значення лічильника.

Спільний секретний ключ: Як користувач, так і сервер аутентифікації діляться спільним секретним ключем (зазвичай у вигляді криптографічного ключу). Цей секретний ключ є довгим, випадковим значенням, відомим тільки користувачу та серверу.

Лічильник: Як користувач, так і сервер підтримують значення лічильника. Лічильник спочатку встановлюється на певне значення і збільшується при кожній генерації OTP.

Функція хешування (HMAC): До спільного секретного ключа та значення лічильника застосовується функція аутентифікації повідомлень на основі хеш-функції (HMAC). Зазвичай в HOTP використовуються такі хеш-функції, як HMAC-SHA1, HMAC-SHA256 і HMAC-SHA512.

Обрізання: Отримане значення хешу часто обрізається для створення коротшого OTP. Кількість цифр в OTP може варіюватися (наприклад, 6 або 8 цифр).

Генерація OTP: Обрізане значення хешу стає OTP. OTP подається користувачу для використання при аутентифікації.

Використання HOTP:

HOTP широко використовується в сценаріях, де необхідна надійна аутентифікація за допомогою одноразових паролів. Використання HOTP включає:

Безпечна аутентифікація: HOTP використовується для підвищення безпеки аутентифікації в різних системах, включаючи віртуальні приватні мережі (VPN), онлайн-банківські системи та корпоративні мережі.

Генерація OTP офлайн: Оскільки OTP HOTP можна генерувати офлайн за допомогою спільного секрету та значення лічильника, їх можна використовувати навіть без підключення до Інтернету.

Токени апаратного забезпечення: Деякі апаратні токени безпеки, такі як токени RSA SecurID, використовують HOTP для генерації OTP.

OAuth 2.0

OAuth 2.0 - це відкритий стандарт для авторизації на основі токенів, що дозволяє користувачу надавати обмежений доступ стороннім додаткам до своїх ресурсів без відкриття своїх облікових даних.

Ролі: OAuth 2.0 визначає кілька ролей:

- **Власник ресурсу**: Користувач, який володіє захищеними ресурсами (наприклад, даними, фотографіями).
- **Клієнт**: Сторонній додаток, який намагається отримати доступ до ресурсів користувача.
- **Авторизаційний сервер**: Керує аутентифікацією та авторизацією користувача.
- **Сервер ресурсів**: Зберігає та обслуговує ресурси користувача.

Авторизація: OAuth 2.0 пропонує кілька потоків авторизації, включаючи:

- **Потік авторизаційного коду:** Зазвичай використовується для веб-додатків і включає отримання коду авторизації та обмін його на маркер доступу.
- **Неявний потік**: Підходить для односторінкових додатків і безпосередньо видає маркери доступу клієнту.

• **Потік авторизації клієнта**: Використовується для взаємодії між машинами, коли клієнт безпосередньо отримує маркер доступу.

• **Маркери доступу**: Після успішної авторизації OAuth 2.0 видає маркер доступу клієнту. Маркер доступу діє як маркер володаря та використовується для доступу до ресурсів користувача на сервері ресурсів.

Використання OAuth 2.0:

OAuth 2.0 широко використовується в різних сценаріях, включаючи:

Доступ до API: Він дозволяє стороннім додаткам безпечно отримувати доступ до API від імені користувачів. Соціальні медіа, хмарні сервіси та фінансові установи часто використовують OAuth 2.0 для авторизації API.

SSO – single sign on: OAuth 2.0 є основою для рішень сингл, які дозволяють користувачам увійти один раз і отримати доступ до кількох служб без повторного введення облікових даних.

Аутентифікація мобільного додатка: OAuth 2.0 використовується для захисту взаємодії між мобільними додатками та службами на стороні сервера, покращуючи захист даних користувача.

OpenID Connect (OIDC)

OpenID Connect - є автентифікаційним рівнем, побудованим на основі OAuth 2.0, призначеним для надання ідентифікації та автентифікації користувача. Він дозволяє програмам перевіряти ідентифікацію кінцевих користувачів та отримувати інформацію про них, використовуючи встановлені механізми OAuth 2.0.

Аутентифікація користувача: OIDC забезпечує аутентифікацію користувача через постачальника ідентифікації (IdP), часто використовуючи стандартні методи аутентифікації, такі як ім'я користувача та пароль або багатофакторну аутентифікацію.

ID-токен: Після успішної аутентифікації IdP видає ID-токен, який є токеном JSON Web Token (JWT). Цей токен містить заяви про аутентифікованого користувача і цифровий підпис IdP для забезпечення його цілісності.

Кінцева точка користувача: OIDC визначає кінцеву точку користувача, яка дозволяє клієнтам отримувати додаткову інформацію про користувача у стандартизованому форматі.

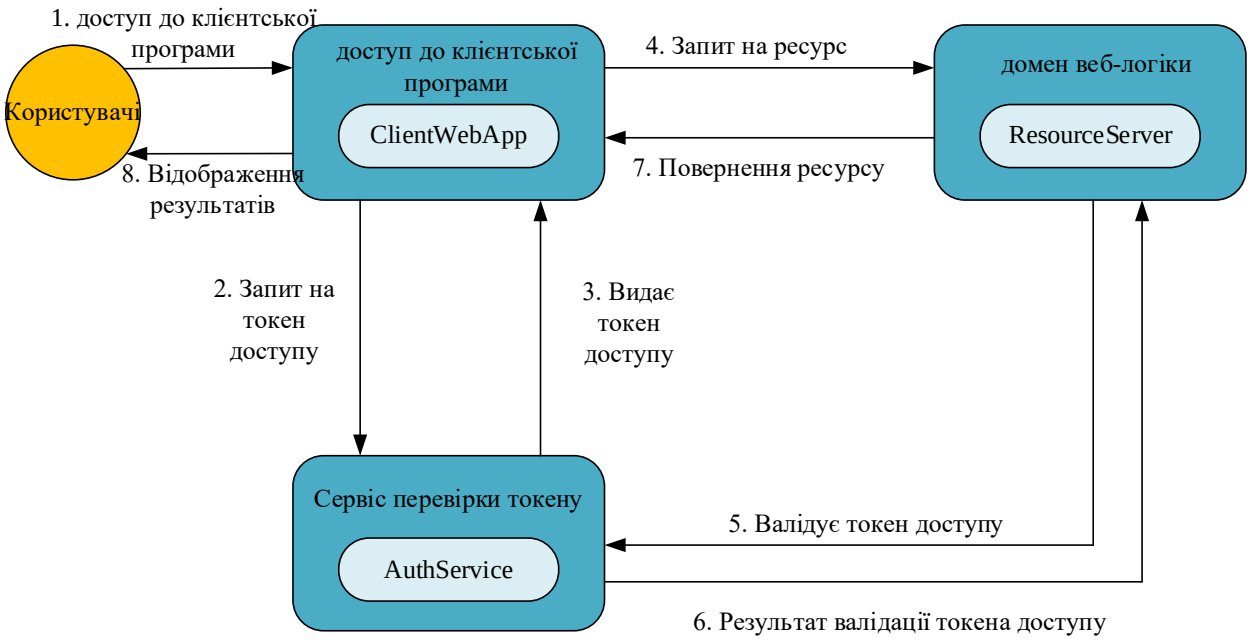


Рис. 2.1. Діаграма архітектури програмного модуля автентифікації з використанням OAuth2.0.

Використання OpenID Connect:

OpenID Connect переважно використовується для аутентифікації користувача в веб- та мобільних додатках. Використання включає:

SSO – single sign on: OIDC дозволяє реалізувати рішення для одного разу входу, що дозволяє користувачам увійти один раз і отримувати доступ до кількох додатків без повторного введення облікових даних.

Інтеграція профілю користувача: OIDC надає стандартизований спосіб отримання даних профілю користувача, включаючи атрибути, такі як ім'я, електронна адреса і фотографія профілю.

Аутентифікація для мобільних і веб-додатків: OIDC забезпечує безпечну аутентифікацію користувача в мобільних і веб-додатках, підвищуючи безпеку та зручність користувача.

Таблиця 1.2

Порівняння протоколів аутентифікації

Протокол аутентифікації	**Опис**	**Сфери використання**
TOTP (Часовий OTP)	Генерація OTP на основі поточного часу	Двофакторна аутентифікація (2FA)
HOTP (OTP на основі HMAC)	Генерація OTP на основі значення лічильника	Двофакторна аутентифікація (2FA)
OAuth 2.0	Протокол на основі токенів для авторизації	Один обліковий запис (SSO) та доступ до API
OpenID Connect	Прошарок аутентифікації, побудований на основі OAuth 2.0	Аутентифікація користувачів для веб-додатків

1.3. Дослідження класифікації токенів для віддаленої автентифікації

Класифікація токенів служить фундаментальною основою для розуміння і категоризації різних типів токенів, які використовуються у віддалених процесах аутентифікації. Важливість класифікації токенів можна пояснити кількома ключовими моментами:

Підвищення рівня безпеки: Класифікація токенів допомагає глибше розуміти відомості про безпеку і ризики, пов'язані з різними типами токенів. Ця інформація дозволяє розробляти більш надійні заходи забезпечення безпеки для захисту від загроз та вразливостей.

Оптимізація користувацького досвіду: Категоризація токенів на основі їх характеристик і використання дозволяє розробляти потоки аутентифікації, які не

тільки безпечні, але й зручні для користувачів. Це гарантує, що користувачі можуть легко та інтуїтивно отримувати доступ до віддалених ресурсів.

Масштабованість і сумісність: Розуміння класифікації токенів допомагає створювати масштабовані системи аутентифікації, які можуть підтримувати широкий спектр застосунків та послуг. Це також забезпечує сумісність між різними компонентами аутентифікації та стандартами.

Зменшення ризиків: Класифікація токенів дозволяє визначати потенційні ризики та вразливості, пов'язані з певними типами токенів. Ця інформація є важливою для розробки стратегій мінімізації ризиків та найкращих практик.

Обсяг класифікації токенів

Обсяг класифікації токенів охоплює широкий спектр типів токенів, призначених для різних цілей та сценаріїв використання. До них включаються:

а. **Токени аутентифікації**: Токени, які використовуються для підтвердження ідентичності користувачів під час процесу аутентифікації. Вони відіграють ключову роль у підтвердженні того, що сутність, яка запитує доступ, дійсно є законним користувачем.

б. **Токени авторизації**: Токени, які використовуються для надання доступу до конкретних ресурсів або послуг на основі заздалегідь визначених дозволів і ролей. Вони визначають, які дії може виконувати користувач або клієнт після аутентифікації.

в. **Токени сеансу**: Токени, які зберігають стан сеансу користувача протягом всього його взаємодії з віддаленою послугою. Ці токени важливі для збереження послідовності дій та контексту під час сеансу користувача.

г. **Токени оновлення**: Токени, які використовуються для отримання нових токенів доступу без потреби повторно вводити дані для аутентифікації. Вони полегшують життя користувачів і спрощують процес аутентифікації.

д. **Токени випадковості**: Токени, використовувані для запобігання атакам типу "повторення" у процесах аутентифікації. Вони гарантують, що запити на аутентифікацію не будуть перехоплені та зловживані.

Механізми аутентифікаційних токенів

Аутентифікаційні токени працюють на основі кількох ключових механізмів:

а. **Ідентифікація користувача**: Аутентифікаційні токени містять інформацію, специфічну для користувача, або заяви, такі як ідентифікатор користувача або ім'я користувача, що дозволяють системі ідентифікувати користувача.

б. **Криптографічні підписи**: Багато аутентифікаційних токенів цифрово підписуються владою (наприклад, сервером аутентифікації), щоб забезпечити їх автентичність та цілісність.

в. **Спільні секрети**: Деякі токени ґрунтуються на спільних секретах між користувачем і сервером. Ці секрети використовуються при створенні токенів та їх перевірці.

г. **Тривалість токенів**: Аутентифікаційні токени часто мають обмежений строк дії для зменшення ризику недозволеного використання. Тривалість токенів може варіюватися в залежності від використовуваних сценаріїв.

Сценарії використання аутентифікаційних токенів

а. **Вхід користувачів**: Вони використовуються під час процесу входу користувачів для підтвердження їхньої ідентичності і надання доступу до захищених облікових записів або платформ.

б. **Аутентифікація API**: Аутентифікаційні токени є необхідними для захисту кінцевих точок API, забезпечуючи доступ до даних або послуг тільки для авторизованих клієнтів чи користувачів.

в. **Одноразовий вхід (Single Sign-On, SSO)**: У рішеннях з одноразовим входом аутентифікаційні токени полегшують безперервний доступ до кількох послуг після єдиного успішного входу.

г. **Аутентифікація з багатьма факторами (Multi-Factor Authentication, MFA)**: Аутентифікаційні токени можуть використовуватися в поєднанні з іншими факторами (наприклад, паролі та біометрія) для підвищення рівня безпеки.

Аспекти аутентифікаційних токенів

а. **Захист даних користувача**: Аутентифікаційні токени часто містять заяви або дані користувача. Забезпечення їх безпечної передачі та зберігання є важливим для захисту чутливої інформації.

б. **Зменшення ризика викрадення токенів**: Потрібно вживати надійні заходи для запобігання викраденню токенів. Важливими є належне зберігання, шифрування та механізми скасування.

в. **Перевірка токенів**: Перевірка токенів на стороні сервера є важливою для забезпечення їхньої подлинності та захисту від втручань.

г. **Термін дії токенів**: Тривалість дії токенів повинна бути належно налаштована для балансу між безпекою та зручністю використання. Коротший термін дії токенів обмежує їхню вразливість у разі втрати.

Найбільш популярним рішенням віддаленої аутентифікації користувачів є JWT та Bearer токени.

1. **Bearer токен**. Є одним з найбільш часто використовуваних типів. Це прості самодостатні токени, які надають доступ без будь-якого подальшого підтвердження особи. Однак вони вразливі, якщо їх перехопити, оскільки кожен, хто володіє токеном, може ним скористатися.

2. **JWT**. Особливий тип токену, який складається з трьох частин: заголовка, корисного навантаження та підпису. Вони мають цифровий підпис для забезпечення їх цілісності та можуть містити інформацію про користувача чи організацію.

3. **Непрозорі токени**. На відміну від JWT токену вони не розкривають жодної інформації про користувача чи сеанс у самому токені. Вони служать посиланням на дані, що зберігають на сервері, забезпечуючи підвищену безпеку та конфіденційність.

Ключові переваги токенів аутентифікації:

1. **Покращена безпека**. Токени зменшують ризик пов’язаний з передачею конфіденційних даних через мережу. Токени можна безпечно

передавати, а їхній вміст часто захищається за допомогою шифрування або цифрових підписів.

2. **Зменшена залежність від пароля**. Токени зменшують залежність від традиційних паролів, які вразливі до різноманітних атак, зокрема підбору та фішингу.

3. **Масштабованість**. Токени можна використовувати як у малих, так і у великих системах, що робить їх адаптованими до різних додатків, від рішень єдиного входу (SSO) до архітектури мікросервісів.

4. **Покращена взаємодія з користувачем**. Аутентифікація на основі токенів часто забезпечує більш зручну роботу користувача, оскільки користувачам не потрібно повторно вводити логін та пароль.

Таблиця 1.3

Порівняння непрозорих і JWT токенів

Особливість	**Непрозорі токени**	**JWT токени**
Структура	Непрозорі токени мають внутрішню структуру, приховану від клієнта, що забезпечує максимальну безпеку. Зазвичай це довгі випадкові рядки.	JWT є самодостатніми та можуть розкривати інформацію користувача, потенційно створюючи загрозу безпеці, якщо вони не зашифровані.
Безпека	Непрозорі токени надзвичайно безпечні, оскільки вони не розкривають дані користувача та стійкі до атак. Вони є кращим вибором для суворих вимог безпеки.	JWT можуть бути безпечними, але вони повинні бути ретельно підписані та зашифровані для належного захисту конфіденційних даних.

Продовження таблиці 1.3

Stateless vs Stateful	Непрозорі токени є кращими в системах автентифікації з урахуванням стану, пропонуючи детальний контроль і моніторинг сеансів користувачів для підвищення безпеки.	JWT зазвичай використовуються в системах без стану, що може створити проблеми для ефективного керування станами сеансу та відкликання.
Витрати на валідацію	Перевірка непрозорих токенів, хоча й відбувається трохи повільніше через перевірки бази даних, забезпечує ретельну перевірку безпеки, що робить їх придатними для додатків із високим рівнем безпеки.	JWT пропонують швидшу перевірку, але їм може бракувати надійності непрозорих токенів, особливо коли йдеться про запобігання неправильному використанню токенів.
Розмір	Непрозорі токени можуть бути більшими, але це невеликий компроміс із покращеною безпекою, яку вони забезпечують, що робить їх ідеальними для безпечних критичних програм.	JWT компактні, але можуть не забезпечувати такий самий рівень гарантії безпеки, як непрозорі токени, особливо в середовищах з високим рівнем безпеки.

Продовження таблиці 1.3

Відкликання	Непрозорі токени можна ефективно відкликати на стороні сервера, зробивши посилання на токен недійсним, забезпечуючи швидку реакцію на загрози безпеці.	Для JWT потрібні складніші механізми, такі як чорні списки токенів або короткочасні токени, що потенційно може призвести до затримок у обробці відкликання.
Керування	Непрозорі токени спрощують керування, оскільки закінчення терміну дії токена, оновлення або керування версіями легко контролюються на стороні сервера, забезпечуючи кращу гнучкість і контроль.	JWT вимагають ретельного керування, що часто потребує повторної видачі токенів, що може бути громіздким у системах автентифікації без стану.
Використання	Непрозорі токени чудово себе показують у додатках, які вимагають найвищого рівня безпеки та контролю, наприклад у фінансових системах або системах охорони здоров’я, де захист даних є першочерговим.	JWT добре підходять для сценаріїв, де пріоритетом є простота та сумісність, як-от єдиний вхід (SSO) і веб-інтерфейси API, але вони можуть не відповідати вимогам безпеки чутливих середовищ.

1.4. Висновки до першого розділу

1. В пункті 1.1. було проведено аналіз сучасного стану та розглянуто проблеми щодо забезпечення безпеки інформаційних мережах
2. В пункті 1.2. було досліджено підходи щодо автентифікації користувачів на базі токенів.
3. В пункті 1.3. було досліджено класифікацію токенів для віддаленої автентифікації.

РОЗДІЛ 2

ПРОБЛЕМИ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ ТА ІНТЕГРАЦІЯ ПРОГРАМНОГО МОДУЛЮ АВТЕНТИФІКАЦІЇ З ВИКОРИСТАННЯМ НЕПРОЗОРИХ ТОКЕНІВ

2.1. Теоретичні основи оптимізації процесу автентифікації

1. Роль оптимізації у забезпеченні безпеки:

Оптимізація автентифікаційного процесу не повинна піддавати сумніву безпеку системи. Врахування наступних аспектів гарантує збалансований підхід:

- **швидкість та безпека ключових алгоритмів**: вибір швидких криптографічних алгоритмів, таких як aes для шифрування та sha-256 для хешування, забезпечить ефективність без втрати безпеки.

- **мінімізація обчислень на стороні клієнта**: важливо уникати виконання складних обчислень на стороні клієнта, оскільки це може бути піддане атакам. збереження обчислень на сервері забезпечить більший контроль та безпеку.

- **автоматизована відновлюваність**: оптимізація повинна враховувати можливість автоматизованого відновлення в разі помилок або атак. визначення автоматичних механізмів реагування дозволить системі швидше відновлювати нормальну роботу.

- **шифрування ключів та матеріалів сесії**: застосування оптимізованих методів генерації та обміну ключами, а також ефективного управління матеріалами сесії, дозволить забезпечити безпеку ключових процесів.

- **використання продуктивних алгоритмів хешування**: вибір швидких алгоритмів хешування з гарним рівнем безпеки (наприклад, sha-256) дозволить забезпечити ефективність та відповідати стандартам безпеки.

- **моніторинг та логування**: система оптимізації повинна включати механізми моніторингу та логування для виявлення аномалій та потенційних загроз безпеці.

2. Методи Паралельної Обробки:

Паралельна обробка є ефективним засобом збільшення продуктивності аутентифікаційного процесу.

- **багатоядерні системи та потоки:** використання багатоядерних систем дозволяє розподіляти завдання між різними ядрами, прискорюючи обробку. застосування паралельних потоків дозволяє розділити обчислення на багато менших частин, які виконуються паралельно.

- **openmp та cuda**: використання технологій паралельного програмування, таких як openmp для cpu або cuda для gpu, може значно покращити швидкість обчислень. це особливо корисно в сучасних системах з потужними графічними процесорами.

- **розпаралелювання великих обчислень**: деякі аутентифікаційні задачі можуть бути розпаралелені, наприклад, обчислення хешу паролів чи перевірка цифрових підписів. важливо визначити такі обчислення та використовувати паралельні методи для їх виконання.

- **управління ресурсами**: ефективне управління ресурсами системи, включаючи керування паралельністю та розподіл ресурсів між потоками, гарантує стійку та ефективну роботу системи під час паралельної обробки.

- **безпека паралельної обробки**: важливо враховувати можливі ризики та вразливості, пов'язані з паралельною обробкою, такі як гонки даних та синхронізація. застосування відповідних механізмів синхронізації та безпеки дозволяє уникнути проблем безпеки.

- **масштабованість**: забезпечення масштабованості паралельних обчислень дозволяє системі ефективно працювати при збільшенні обсягу оброблюваної інформації та кількості користувачів.

3. Оптимізовані алгоритми шифрування та хешування

Шифрування та хешування відіграють важливу роль у забезпеченні безпеки аутентифікаційного процесу. Оптимізація цих алгоритмів важлива для підвищення ефективності та швидкодії.

- **вибір ефективних алгоритмів**: важливо вибирати шифрувальні та хеш-алгоритми, які поєднують високий рівень безпеки із швидкістю обчислень. наприклад, використання aes для шифрування та sha-256 для хешування може забезпечити оптимальний баланс.

- **апаратне прискорення**: враховуючи можливості апаратного прискорення шифрування (наприклад, застосування апаратних модулів aes), можна покращити швидкодію обробки.

- **швидкість хешування та підтримка ключів**: оптимізовані алгоритми хешування забезпечують швидку генерацію хешів, що особливо важливо в аутентифікаційних системах. крім того, використання ключів у хеш-функціях може підвищити безпеку та ускладнити атаки.

- **кількість раундів та ітерацій**: налаштування кількості раундів шифрування та ітерацій у хеш-функціях може впливати на якість та швидкодію. важливо знаходити оптимальні значення, які забезпечують достатню безпеку.

- **розподіл обчислень**: розподіл обчислень між серверами та клієнтами може зменшити навантаження та покращити загальну швидкодію. розглядайте можливість використання асиметричних методів, де частину обчислень виконує сервер, а частину – клієнт.

- **апаратне ускладнення хешування**: застосування апаратного ускладнення (наприклад, застосування "солі" в хеш-функціях) може додатково ускладнити атаки методом підбору та забезпечити безпеку паролів.

4. Використання кешування

- **кешування результатів запитів**: система може зберігати результати попередньо виконаних запитів, у яких параметри повторюються. це дозволяє уникати повторного виконання обчислень для тих самих запитів та значно зменшує час відповіді.

- **кешування сесій**: кешування даних сесій, таких як токени аутентифікації, може знизити навантаження на систему та покращити час аутентифікації для повторних запитів.

- **меморізація проміжних результатів**: зберігання інтермедіатних результатів обчислень для подальшого використання може бути корисним у випадках, коли ті самі обчислення використовуються у різних частинах аутентифікаційного процесу.

- **стратегії спільного кешування**: врахування стратегій спільного кешування між серверами може забезпечити ще більшу ефективність. поділ кешів між різними елементами архітектури дозволяє оптимізувати використання ресурсів.

- **асинхронне оновлення кешу**: використання асинхронного оновлення кешу дозволяє системі продовжувати обслуговування запитів, навіть якщо кеш ще не оновлено. це забезпечує швидку відповідь на нові запити.

- **керування витратами пам'яті**: важливо враховувати витрати пам'яті при кешуванні та меморизації. ефективне управління пам'яттю дозволяє уникати перевитрат та забезпечує стійку роботу системи.

5. Оптимізація комунікації та мережевих протоколів

- **використання протоколів з низькою затримкою**: вибір мережевих протоколів з низькою затримкою, таких як udp (user datagram protocol), може знизити час передачі даних, особливо для критичних за швидкістю аутентифікаційних операцій.

- **компресія даних та мінімізація запитів**: використання методів компресії даних дозволяє зменшити обсяг передаваної інформації, а також мінімізація кількості запитів може покращити час відповіді.

- **розподілення завдань між серверами**: розподілення завдань між різними серверами може сприяти оптимізації комунікацій. наприклад, окремі сервери можуть бути відповідальні за аутентифікацію, а інші - за взаємодію з базою даних чи збереженням сесій.

- **використання асинхронної комунікації**: використання асинхронних методів комунікації дозволяє серверам обробляти одночасно більше запитів, що сприяє паралельній обробці та зменшенню затримок.

- **використання cdn (content delivery network):** використання cdn для розподілення ресурсів та динамічного кешування може значно зменшити час завантаження статичних елементів та поліпшити швидкість взаємодії.

- **використання http/2 та http/3:** апгрейд до нових версій протоколів http, таких як http/2 та http/3, може покращити ефективність комунікацій шляхом удосконалення механізмів передачі даних та зменшення затримок.

6. Впровадження методів виявлення та захисту від атак

- **аналіз запитів та виявлення аномалій:** використання методів аналізу звернень та виявлення аномалій дозволяє вчасно виявляти та блокувати підозрілі або неавторизовані спроби доступу.

- **захист від брутфорс-атак:** впровадження механізмів захисту від брутфорс-атак, таких як блокування акаунтів після кількох невдалих спроб, покращує безпеку та знижує ризик несанкціонованого доступу.

- **використання captcha та двофакторної аутентифікації:** застосування captcha та двофакторної аутентифікації ускладнює автоматизовані атаки та забезпечує додатковий рівень безпеки.

- шифрування та захист відповідей: важливо застосовувати шифрування та механізми захисту відповідей для уникнення витоку конфіденційної інформації від атак з боку користувачів чи зловмисників.

- моніторинг журналів та інцидентів: система моніторингу журналів та автоматизований аналіз інцидентів дозволяють вчасно виявляти та вирішувати потенційні загрози безпеки.

- аналіз інтелектуальних заходів безпеки: використання інтелектуальних систем безпеки, що базуються на штучному інтелекті, дозволяє автоматично розпізнавати та реагувати на нові види загроз.

- реакція на загрози в реальному часі: реалізація механізмів реакції на загрози в реальному часі дозволяє швидко втручатися та блокувати атаки перед їхнім завершенням.

7. Забезпечення високої доступності та резервування

- **відмовостійкість та дублювання систем:** реалізація механізмів відмовостійкості та дублювання систем дозволяє забезпечити неперервну роботу в разі виникнення проблем чи відмов в роботі одного з вузлів.

- **балансування навантаження:** використання систем балансування навантаження дозволяє розподіляти трафік між різними серверами, що зменшує ризик перевантаження та забезпечує оптимальне використання ресурсів.

- **резервні копії та відновлення даних:** регулярне створення резервних копій та реалізація механізмів відновлення даних дозволяє відновлювати систему до попереднього стану у випадку втрати інформації або несправності.

- **географічне розміщення серверів:** розміщення серверів у різних географічних регіонах забезпечує неперервну роботу в разі регіональних або локальних проблем, таких як природні катастрофи чи технічні збої.

- **моніторинг та системи сповіщення:** реалізація систем моніторингу та сповіщення дозволяє оперативно виявляти та реагувати на проблеми, надаючи можливість адміністраторам швидко приймати заходи.

- **шифрування та захист даних:** застосування шифрування даних під час передачі та зберігання допомагає захистити конфіденційну інформацію від несанкціонованого доступу.

- **тестування та відновлення систем:** регулярне тестування систем на відмовостійкість та впровадження механізмів відновлення допомагає виявляти та виправляти можливі проблеми перед виникненням реальних ситуацій відмов.

8. Автоматизоване тестування та моніторинг безпеки

- **автоматизоване тестування аутентифікації:** розробка та використання автоматизованих тестів для перевірки працездатності та безпеки аутентифікаційного модуля дозволяє швидко виявляти та виправляти помилки та вразливості.

- **тестування навантаження та стресостійкості:** проведення тестів на навантаження та стресостійкість допомагає визначити межі працездатності системи та виявити проблеми, пов'язані зі значними обсягами запитів.

- **виявлення вразливостей та тестування злому:** використання інструментів для виявлення вразливостей та проведення тестів на злом дозволяє виявити та виправити можливі проблеми з безпекою.

- **моніторинг журналів та системи**: система моніторингу, яка автоматично аналізує журнали та системні події, дозволяє вчасно виявляти та реагувати на події, які можуть свідчити про атаки або виникнення проблем безпеки.

- **моніторинг використання ресурсів**: слідкування за використанням ресурсів, таких як cpu та пам'ять, дозволяє виявляти аномалії та вчасно реагувати на можливі перевантаження.

- **автоматизовані системи виявлення загроз**: використання систем виявлення загроз, які використовують машинне навчання та аналіз великих обсягів даних, допомагає виявляти навіть складні атаки.

- **регулярні тестування безпеки**: проведення регулярних тестів на безпеку та вдосконалення методів тестування дозволяє підтримувати високий рівень захищеності системи.

9. Системи управління сесіями

- **типи сесій:** визначення різних типів сесій, таких як постійні (зберігаються на більший термін) та тимчасові (видаляються після закінчення сесії), дозволяє адаптувати підходи до різних вимог безпеки.

- **управління часом життя сесії**: задання обмежень на час життя сесій, які регулюють тривалість активності користувачів у системі, зменшує ризик несанкціонованого доступу через витікання аутентифікаційних даних.

- **взаємодія з кукі та токенами**: інтеграція засобів взаємодії з кукі та токенами дозволяє ефективно управляти сесіями через механізми браузера та безпечних методів передачі даних.

- **моніторинг та завершення неактивних сесій**: реалізація механізму моніторингу та автоматичного завершення неактивних сесій допомагає уникнути ризику несанкціонованого використання залишених відкритими сесій.

- **додаткові засоби безпеки**: використання додаткових засобів безпеки, таких як вимагання повторної аутентифікації для критичних операцій або двофакторна аутентифікація, забезпечує додатковий рівень захисту сесій.

- **захист від атак на сесії**: розробка та впровадження механізмів захисту від атак на сесії, таких як атаки типу session hijacking або session fixation, гарантує стійкість системи до потенційних загроз.

- **аудит та журналювання дій**: ведення аудиту та журналювання дій користувачів під час сесії дозволяє виявляти та реагувати на небезпечні або несподівані події.

10. Адаптивні механізми контролю доступу

- **системи ризик-орієнтованої аутентифікації**: використання алгоритмів, що аналізують рівень ризику під час аутентифікації, дозволяє системі адаптивно підходити до ситуацій та вимагати додаткові перевірки при виявленні підозрілих дій чи пристроїв.

- **множинна аутентифікація на основі контексту**: застосування множинної аутентифікації на основі контексту дозволяє використовувати різні методи аутентифікації в залежності від умов, наприклад, використовуючи біометричні дані або одноразові паролі.

- **виявлення аномалій та поведінковий аналіз**: застосування систем виявлення аномалій та аналізу поведінки допомагає визначити незвичайні чи підозрілі дії користувачів та вживати відповідні заходи безпеки.

- **динамічні правила контролю доступу**: розробка динамічних правил контролю доступу дозволяє системі адаптивно встановлювати обмеження на основі різних умов, таких як час доби, геолокація чи тип пристрою.

- **аналіз зміни поведінки користувача**: використання алгоритмів, які аналізують зміни у поведінці користувача, дозволяє виявляти підозрілі сценарії та вживати заходів для запобігання можливим загрозам.

- **управління привілеями на основі ролей**: адаптивне управління привілеями на основі ролей дозволяє автоматично змінювати рівень доступу користувачів в залежності від їхніх функцій та контексту використання.

- **системи сповіщення та моніторингу:** реалізація систем сповіщення та моніторингу дозволяє оперативно реагувати на події, що вимагають уваги адміністратора чи заходів безпеки.

11. Оптимізація запитів до бази даних

- **індексація таблиць**: створення індексів на полях, за якими відбувається пошук та фільтрація, значно прискорює виконання запитів до бази даних.

- **коректне використання join**: використання оптимальних типів join та розумне використання їхніх умов дозволяє уникати зайвих обчислень та запобігає перевантаженню запитів.

- **оптимізація sql запитів**: ретельна оптимізація тексту sql-запитів, використання відомих оптимізаторів та врахування специфічностей конкретної бази даних сприяють швидкому виконанню запитів.

- **кешування результатів запитів**: використання кешування результатів запитів дозволяє уникнути зайвого повторного виконання запитів, особливо для тих, які рідко змінюються.

- **оптимізація схеми бази даних**: розробка оптимальної схеми бази даних, з урахуванням вимог до аутентифікації, може суттєво покращити продуктивність.

- **використання індексів повторного використання**: використання індексів повторного використання для збереження попередньо вирахованих результатів обчислень може зменшити навантаження на базу даних.

- **аналіз та виправлення помилок виконання запитів**: регулярний аналіз та виправлення помилок виконання запитів дозволяє забезпечити стабільну та ефективну роботу бази даних.

- **використання партицій та інших оптимізацій**: використання партицій, кешування результатів попередніх запитів, та інші оптимізаційні прийоми можуть покращити швидкість виконання запитів.

12. Використання компресії даних

- **методи компресії:** вибір ефективних методів компресії, таких як gzip, zlib, чи brotli, дозволяє зменшити обсяг передаваних даних без втрати їхньої інформаційної цінності.

- **компресія перед передачею даних:** застосування компресії перед передачею даних по мережі зменшує обсяг трафіку та підвищує швидкість обміну інформацією між користувачем та сервером.

- **компресія збережених даних:** застосування компресії для збереження даних у базі даних дозволяє зменшити вимоги до місця та прискорити процес читання/запису.

- **компресія веб-ресурсів:** компресія веб-ресурсів, таких як зображення, стилі, та скрипти, допомагає покращити швидкість завантаження сторінок та забезпечити більш ефективне використання мережевого трафіку.

- **автоматичне визначення найефективнішого методу**: реалізація механізму автоматичного визначення найефективнішого методу компресії для конкретного типу даних дозволяє динамічно адаптуватися до різних сценаріїв.

- **моніторинг та оптимізація використання ресурсів:** використання систем моніторингу для визначення впливу компресії на використання ресурсів дозволяє здійснювати оптимізації та підтримувати баланс між швидкістю та використанням пам'яті.

- **захист від втрат даних**: важливо враховувати можливість втрати даних під час компресії, особливо для даних, які можуть бути критичними для безпеки аутентифікаційного процесу.

- **компресія даних на рівні програмного забезпечення**: реалізація компресії на рівні програмного забезпечення дозволяє ефективно використовувати ресурси системи та динамічно вибирати метод компресії.

13. Використання локальних копій даних

- **кешування запитів та результатів**: застосування механізмів кешування дозволяє зберігати локальні копії результатів запитів, що регулярно використовуються, зменшуючи час їхнього виконання.

- **локальне збереження сесій та контексту**: використання локальних копій для збереження інформації про сесії користувачів та їхній контекст дозволяє уникати постійних звернень до центрального зберігання даних.

- **резервування та копіювання даних:** розгляд можливостей резервування та локального копіювання даних, які часто використовуються, сприяє прискоренню доступу та зменшенню завантаження центрального сховища.

- **офлайн режим та локальна робота**: врахування можливості офлайн режиму, де локальні копії даних використовуються, дозволяє забезпечити неперервну роботу аутентифікаційного модуля навіть при відсутності з'єднання з центральним сервером.

- **використання локальних сховищ**: застосування локальних сховищ даних на клієнтському рівні, таких як local storage або indexeddb, дозволяє зберігати частину необхідної інформації локально для подальшого використання.

- **механізми синхронізації даних**: визначення механізмів синхронізації даних між локальними копіями та центральним зберіганням дозволяє уникнути конфліктів та зберігати актуальну інформацію.

- **керування версіями локальних даних**: реалізація системи керування версіями локальних даних дозволяє ефективно вирішувати конфлікти та забезпечує цілісність інформації.

- **безпека та шифрування локальних даних**: врахування питань безпеки та використання шифрування для збереження локальних копій даних є ключовим для захисту конфіденційної інформації.

14. Забезпечення прозорості та спрощення інтеграції

- **документація та API доступ**: наявність детальної документації та зручного арі спрощує інтеграцію аутентифікаційного модуля у різні системи та додатки.

- **механізми розширення та плагіни**: розробка механізмів розширення та підтримка плагінів дозволяє легко додавати новий функціонал та адаптувати систему під конкретні вимоги.

- **стандартизація та сумісність**: стандартизація протоколів та забезпечення сумісності з існуючими стандартами допомагає уникати проблем при інтеграції та забезпечує високу прозорість роботи.

- **забезпечення простоти налаштування**: забезпечення простих та зрозумілих інтерфейсів для налаштування аутентифікаційного модуля сприяє його швидкій і безболісній інтеграції в різні середовища.

- **можливість власних розширень**: надання можливості розробки та використання власних розширень дозволяє адаптувати аутентифікаційний модуль до конкретних потреб користувачів.

- **взаємодія з ідентифікаційними системами**: забезпечення можливості взаємодії з різними ідентифікаційними системами та протоколами (ldap, saml, oauth) спрощує інтеграцію та забезпечує універсальність використання.

- **аудит та логування подій**: введення системи аудиту та логування подій дозволяє відстежувати та аналізувати дії користувачів, що сприяє прозорості та підвищує безпеку системи.

- **інтеграція з централізованими системами управління**: можливість інтеграції аутентифікаційного модуля з централізованими системами управління (identity management) дозволяє ефективно керувати користувачами та їхніми правами.

- **підтримка сучасних протоколів та стандартів**: забезпечення підтримки сучасних протоколів (tls, jwt) та стандартів безпеки дозволяє забезпечити високий рівень захищеності системи.

- **наявність SDK та зручні інструменти розробки**: розробка та надання sdk разом із зручними інструментами для розробників сприяє швидкій інтеграції та розвитку функціоналу.

- **системи моніторингу та аналізу продуктивності**: інтеграція з системами моніторингу та аналізу продуктивності допомагає виявляти проблеми та оптимізувати роботу аутентифікаційного модуля.

- **підтримка різних операційних систем**: забезпечення сумісності та підтримки різних операційних систем (windows, linux, macos) розширює можливості використання аутентифікаційного модуля в різних середовищах.

- **оптимізація часу відгуку та швидкості роботи**: впровадження оптимізацій для покращення часу відгуку та швидкості роботи сприяє ефективності та зручності використання.

- **створення доступних інтерфейсів для адміністрування**: розробка зручних та доступних інтерфейсів для адміністрування допомагає швидко та ефективно налаштовувати параметри роботи аутентифікаційного модуля.

2.2. Проблеми автентифікації користувачів

Вразливості автентифікації – це проблеми, які впливають на процеси автентифікації та роблять веб-сайти та програми сприйнятливими до атак на безпеку, під час яких зловмисник може маскуватися під законного користувача.

Автентифікація є важливою частиною будь-якого веб-сайту чи програми, оскільки це просто процес розпізнавання ідентифікаторів користувачів.

Кілька поширених категорій проблем автентифікації можуть виникати через уразливості облікових даних для входу, захисту сайту чи навіть основного коду.

Є кілька способів, за допомогою яких можуть виникати вразливості автентифікації, найпоширенішими з яких є ігнорування зон ризику, помилки в коді чи логіці та неправильний вибір користувача, який може поєднуватися з попередніми двома. Якщо програма має поганий захист від грубої сили(brute force), зловмисники можуть скористатися цим, щоб отримати доступ навіть до

добре захищених облікових записів або дешевий масовий доступ до погано захищених. Так само, якщо є логічні помилки або помилки кодування, зловмисники можуть обійти частину або весь процес автентифікації.

Уразливості автентифікації мають серйозні наслідки — чи то через слабкі паролі, чи через поганий дизайн і реалізацію автентифікації.

Зловмисники можуть використовувати ці вразливості, щоб отримати доступ до систем і облікових записів користувачів, щоб:

- Викрасти конфіденційну інформацію
- Видавати себе за законного користувача
- Отримати контроль над програмою
- Повністю знищити систему

Якщо зловмисники можуть незаконно пройти процес автентифікації та отримати доступ до облікового запису користувача, вони можуть викрасти важливі дані, такі як імена, дані кредитної картки, медичні записи та податкові ідентифікаційні номери. Вони також можуть виконувати дії від імені користувача, наприклад ініціювати фінансові операції, видаляти дані або передавати право власності на ресурси.

Що ще гірше, зловмисники можуть повністю контролювати вашу систему або повністю вимкнути її, якщо знайдуть доступ до облікових записів верхнього рівня, таких як профілі адміністратора. Проявивши ці вразливості автентифікації, вони можуть суттєво вплинути на життєздатність і довіру до вашої компанії.

Уразливі місця автентифікації, якщо їх не контролювати належним чином, можуть завдати шкоди не лише безпеці компанії, але й її репутації.

Найпоширеніші вразливості автентифікації, на які варто звернути увагу:

1. Помилка захисту від грубої сили(brute force)

Атака методом грубої сили, наприклад атака за словником, — це спроба отримати незаконний доступ до системи або облікового запису користувача шляхом введення великої кількості випадково згенерованих або попередньо

згенерованих комбінацій імен користувачів і паролів, доки не буде знайдено те, що працює.

Якщо є помилкова система захисту від грубої сили, наприклад помилка в логіці автентифікації, брандмауері або протоколі захищеної оболонки (SSH), зловмисники можуть викрасти облікові дані та процеси для входу, ставлячи під загрозу безпеку облікових даних користувача.

2. Слабкі облікові дані для входу

Коли користувачі реєструються для облікового запису на сайті або в програмі, яка використовує вхід на основі пароля, їм пропонується створити ім'я користувача та пароль.

Однак якщо пароль передбачуваний, це може призвести до вразливості в процесі автентифікації. Передбачувані імена користувачів можуть полегшити зловмисникам націлювання на конкретних користувачів.

Замість того, щоб використовувати повну атаку грубої сили, зловмисники шукатимуть облікові записи з паролями, які легко вгадати, які використовуються занадто часто. Вони спробують ввести такі звичайні облікові дані, як «admin», «admin1» і «password1». Без обмежень на слабкі паролі навіть сайти, захищені від атак грубої сили, можуть виявитися скомпрометованими.

3. Перерахування імен користувачів

Перерахування імен користувачів не є точною вразливістю автентифікації. Але це може полегшити життя зловмисника, знизивши вартість інших атак, таких як атаки грубої сили або слабкі перевірки облікових даних.

Цей процес нумерації імені користувача підтверджує, чи дійсне ім'я користувача. Наприклад:

```
Error: The password you entered for the username.
Sally is incorrect. Lost your password?
```

У цьому випадку ім'я користувача правильне, а пароль – ні.

```
Error: The username user3333211 is not registered on this site.
If you are unsure of your username, try your email address
instead.
```

Тут ім’я користувача недійсне.

Проблема з нумерацією імен користувачів полягає в тому, що зловмисники можуть визначити, які імена користувачів дійсні. Потім вони можуть спробувати зламати дійсні облікові записи користувачів за допомогою методів грубої сили, не витрачаючи час і гроші на тестування безлічі недійсних імен облікових записів.

4. Базова автентифікація HTTP

Цей класичний протокол веб-автентифікації простий у застосуванні, однак він не позбавлений ризиків.

Базова автентифікація HTTP проста: під час кожного запиту надсилаються ім’я користувача та пароль. Однак якщо відповідні протоколи безпеки, такі як шифрування сеансу TLS, не використовуються для зв’язку, інформація про ім’я користувача та пароль може бути надіслана відкрито, що полегшить зловмисникам викрадення облікових даних.

Оскільки включені облікові дані містять дуже мало контексту, їх можна легко використати в атаках, таких як підробка міжсайтових запитів (CSRF). Крім того, оскільки вони додаються до кожного окремого запиту, сучасні браузери зазвичай кешують цю інформацію на невизначений термін, з мінімальною можливістю «вийти» та запобігти повторному використанню облікових даних локальним зловмисником у майбутньому.

5. Погане керування сеансами

Уразливість в управлінні ідентифікаторами сеансів призводить до викрадення дійсних автентифікованих сеансів. Це одна з поширених веб-уразливостей для обходу паролів.

Існує кілька вразливостей, пов’язаних із неправильним керуванням сеансом, наприклад відсутність тайм-аутів сеансу, виявлення ідентифікаторів сеансу в URL-адресах, файли cookie сеансу без установленого прапорця «Лише Http» і погане визнання сеансу недійсним.

Якщо зловмисники можуть захопити контроль над існуючим сеансом, вони легко проникають у систему, припускаючи особу вже автентифікованого користувача, повністю минаючи процес автентифікації.

6. Залишатися в системі

Прапорець «Запам’ятати мене» або «Залишати мене в системі» під формою входу дозволяє надзвичайно легко залишатися в системі після закриття сеансу. Він генерує файл cookie, який дозволяє вам пропустити процес входу.

Однак це може призвести до вразливості автентифікації на основі файлів cookie, якщо зловмисник може передбачити файл cookie або визначити шаблон його створення. Вони можуть використовувати такі зловмисні методи, як атаки грубої сили, щоб передбачити файли cookie, і міжсайтовий сценарій (XSS) для злому облікових записів користувачів, дозволяючи зловмисному серверу використовувати законний файл cookie.

Якщо файл cookie погано розроблений або захищений, зловмисники можуть отримати паролі або інші конфіденційні (і захищені законом) дані, такі як адреси користувачів або дані облікового запису, із збережених файлів cookie.

7. SQL ін'єкція

SQL-ін’єкція — це вектор атаки, який неочікуваним чином використовує зловмисний код SQL для маніпулювання та доступу до бази даних.

Ін’єкції SQL можуть уможливити атаки на механізми автентифікації шляхом викрадення відповідних даних (наприклад, погано захищених хешів паролів) із незахищеної бази даних. Вони також можуть обійти механізми автентифікації, якщо впроваджений код SQL виконується внутрішнім (і вже авторизованим) інструментом, який не зміг достатньо перевірити зовнішній вхід.

8. Небезпечна зміна та відновлення пароля

Іноді користувачі забувають або просто хочуть змінити свої паролі та натискають посилання «Забули пароль» або «Забули пароль».

Процес скидання пароля створює вразливість автентифікації, якщо програма використовує слабкий механізм відновлення пароля, як-от прості

контрольні запитання, відсутність CAPTCHA або повідомлення електронної пошти для скидання пароля з надто довгими часами очікування або без них.

Якщо функція відновлення пароля має недоліки, зловмисники потенційно можуть використати методи грубої сили або отримати доступ до інших зламаних облікових записів, щоб отримати доступ до облікових записів користувачів і облікових даних, які добре захищені за звичайних обставин.

9. Помилка двофакторної автентифікації

Хоча двофакторна автентифікація (2FA) ефективна для безпечної автентифікації, вона може спричинити критичні проблеми безпеки, якщо її не впровадити належним чином.

Зловмисники можуть визначити чотири- та шестизначні коди перевірки 2FA за допомогою атак обміну SIM-картою, якщо вони надіслані через SMS. Деяка двофакторна автентифікація також не є справді двофакторною; якщо користувач намагається отримати доступ до конфіденційної інформації на викраденому телефоні за допомогою кешованих облікових даних, «другий фактор», який надсилає повідомлення на той самий телефон, не додає додаткової безпеки.

Уразливість двофакторної автентифікації також може виникнути, якщо немає захисту від грубої сили для блокування облікового запису після певної кількості спроб входу.

10. Вразлива логіка автентифікації

Логічні вразливості поширені в програмних додатках. Це відбувається в результаті поганого кодування або дизайну, що впливає на автентифікацію та авторизацію доступу та функціональність програми.

Помилка логіки програми може бути спричинена зловживанням функціональністю, слабкими заходами безпеки або пропущеним кроком у процедурі перевірки.

Наприклад, програма може запропонувати користувачеві відповісти на таємне запитання, яке логіка вважає таким, що «знає лише правильна особа». Але такі запитання, як день народження користувача чи дівоче прізвище матері, часто

знайти легко. Ця вразливість дозволяє кібер-зловмисникам легко обійти автентифікацію та отримати незаконний доступ до таких облікових записів.

11. Людська недбалість

Згідно зі звітом Shred-it за 2020 рік, до 31% керівників старших класів повідомили, що недбалість співробітників є другою основною причиною витоку їхніх даних.

Людська помилка може призвести до серйозних уразливостей автентифікації, якими набагато легше скористатися, ніж атаками грубої сили, ін'єкціями SQL та обходами автентифікації. Ця недбалість включає такі дії, як:

- Залишати комп'ютер увімкненим і незаблокованим у громадському місці
- Втрата пристроїв через крадіжку
- Витік конфіденційної інформації незнайомцям
- Написання поганого коду
- Як запобігти вразливості автентифікації?

Хоча вразливі місця автентифікації легко визначити, вони значно впливають на кібербезпеку.

Способи вирішення проблем автентифікації:

1. Впровадження надійної системи захисту від грубої сили: атаки грубої сили можна запобігти шляхом блокування облікових записів, обмеження швидкості, IP-моніторингу, брандмауерів програм і CAPTCHA.

2. Застосування політики безпечних паролів. Створити засіб перевірки паролів, який повідомляє користувачам, наскільки надійні їхні паролі в режимі реального часу. Також можна запровадити автентифікацію без пароля за допомогою таких стандартів, як FIDO2, щоб зменшити ризик і стрес, пов'язаний із керуванням паролями.

3. Застосування суворої транспортної безпеки HTTP (HSTS): це змушує веб-сеанси використовувати шифрування TLS, запобігаючи доступ до конфіденційної інформації під час передачі.

4. Вимикати перерахування імен користувачів: генеруючи ту саму помилку для помилки входу незалежно від того, чи було ім'я користувача

дійсним чи недійсним, ви змушуєте зловмисника підбирати не лише набір можливих паролів, але й набір ймовірних імен користувачів, а не дотримуватися ті, які вони знають, є дійсними.

5. Змінити заголовки файлів cookie: зміна заголовків файлів cookie захищає їх від зловмисних атак. Використання тегів HttpOnly і SameSite під час налаштування заголовків файлів cookie запобігає атакам XSS і CSRF відповідно.

6. Уважно перевіряти свій код під час перевірки: це важливо для виявлення будь-яких вразливостей у вашому коді.

7. Використання параметризованих операторів: запобігання атакам SQL-ін’єкції за допомогою перевірки вхідних даних і параметризованих запитів. За допомогою них безпечніше уникати безпосереднього розміщення введених користувачем даних безпосередньо в оператори SQL.

8. Застосування належної багатофакторної автентифікацію: використання багатофакторної автентифікації безпечніше, ніж механізми на основі паролів. Однак для ефективної реалізації цієї форми автентифікації вам потрібен надійний код і безпечна генерація кодів підтвердження.

2.3. Теоретичні основи інтеграції програмного модулю автентифікації користувачів з використанням непрозорих токенів.

2.2.1. Основні принципи непрозорості токенів

- Відкритість процесу

Усі етапи автентифікації повинні бути детально описані та доступні для перевірки. Це включає в себе способи збору даних, методи перевірки ідентифікації, алгоритми шифрування та інші складові процесу.

- Криптографічна безпека

Застосування криптографічних методів для захисту конфіденційної інформації, таких як паролі чи інші ідентифікатори. Важливо використовувати безпечні алгоритми шифрування та перевірки цілісності даних.

- Логування та аудит

Запис подій та дій під час процесу автентифікації. Логи дозволяють відстежувати, хто, коли і як використовував систему автентифікації. Це корисно для виявлення можливих загроз та реагування на інциденти.

- Децентралізація і дистрибуція даних

Розподілення інформації та функцій автентифікації, щоб уникнути ризику однієї точки збою чи злому. Також це може допомагати у збереженні конфіденційності та уникненні централізованого контролю.

- Відкритий стандарт

Використання широко прийнятих відкритих стандартів для автентифікації, таких як OAuth, OpenID Connect, або SAML. Це сприяє сумісності та дозволяє різним системам взаємодіяти між собою.

Ці принципи спрямовані на створення безпечних, прозорих та надійних систем автентифікації. Вони дозволяють уникати можливих атак, забезпечують відстеження дій користувачів та допомагають зберегти конфіденційність особистої інформації.

2.2.2. Процес автентифікації з використанням непрозорих токенів

- Видача токена:

Користувач намагається отримати доступ до системи чи ресурсу, що вимагає автентифікації. Замість традиційного введення пароля, система може видалити непрозорий токен.

- Аутентифікація:

Користувач представляє свої ідентифікаційні дані системі. Це може бути введення пароля, сканування відбитку пальця, використання двофакторної аутентифікації, чи інші методи.

- Створення та передача токена

Після успішної аутентифікації система генерує непрозорий токен, який призначений для подальшої ідентифікації користувача. Цей токен містить інформацію про права доступу та інші деталі.

- Збереження токена:

Клієнтська сторона (наприклад, додаток чи веб-браузер) зберігає отриманий токен. Зазвичай це відбувається за допомогою механізмів, таких як cookies, localStorage, або інші методи зберігання на боці клієнта.

- Надсилання токена при кожному запиті:

При подальших запитах до системи чи ресурсу користувач включає токен в заголовок запиту чи в іншу частину запиту. Це дозволяє серверу ідентифікувати користувача та перевіряти його права доступу.

- Перевірка та оновлення токена:

Сервер перевіряє валідність токена при кожному запиті. Якщо токен валідний і має право доступу до ресурсу, сервер обробляє запит. Якщо токен застарів або потрібно оновлення, може використовуватися механізм оновлення токенів без повторної аутентифікації користувача.

- Завершення сесії та видалення токена:

Після завершення сесії або виходу з системи токен може бути видалений або заблокований для забезпечення безпеки та уникнення несанкціонованого доступу.

Цей процес забезпечує безпеку та непрозорість, оскільки інформація про користувача зберігається у вигляді токенів, а не прямо в ідентифікаційних данних. Також, використання криптографічних методів забезпечує захист токенів від підробки та несанкціонованого доступу.

2.2.3. Захист Токенів та Безпека Обміну Інформацією

- Використання протоколів безпеки:

Використання протоколів, таких як HTTPS (HTTP Secure), для забезпечення шифрування даних під час їхньої передачі між клієнтом і сервером.

Це запобігає можливому перехопленню або модифікації інформації під час транзакції.

- Шифрування Токенів:

Токени, які передаються між клієнтом і сервером, повинні бути шифрованими. Використання криптографічних алгоритмів, таких як AES (Advanced Encryption Standard), може допомогти захистити інформацію в токенах від незаконного доступу.

- Використання токенів з коротким терміном дії:

Токени мають обмежений термін дії. Це допомагає зменшити ризик використання вже викраденого токена. Періодична зміна токенів також підвищує рівень безпеки.

- Захист від атаки перехоплення сесій (session hijacking):

Використання захисних механізмів, таких як токени з одноразовим використанням (One-Time Tokens) чи механізми обміну ключами, може запобігти атакам перехоплення сесій.

- Цифрові підписи та перевірка аутентичності:

Включення цифрових підписів до токенів дозволяє перевірити їхню аутентичність. Це гарантує, що токени не були змінені або створені незаконним чином.

- Механізми Багатофакторної Аутентифікації::

Використання багатофакторної аутентифікації (Multi-Factor Authentication) дозволяє додатково забезпечити ідентифікацію користувача, навіть якщо токен отримано незаконно.

- Захист Від Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF):

Захист від атак, таких як XSS та CSRF, є критичним для токенів. Забезпечення валідації та обробки даних на клієнтському та серверному рівнях може допомогти уникнути цих загроз.

- Аудит та Моніторинг:

Встановлення систем аудиту та моніторингу для виявлення незвичайних або підозрілих активностей у системі. Це дозволяє реагувати на потенційні загрози та атаки.

Загальний підхід повинен включати комплекс заходів, які спрямовані на мінімізацію ризиків та забезпечення безпеки обміну інформацією та управління токенами.

2.2.4. Інтеграція з ідентифікаційною системою

- Стандартизація та відкриті протоколи:

Використовуйте стандартні та відкриті протоколи для забезпечення взаємодії між вашою системою та ідентифікаційною системою. Наприклад, SAML (Security Assertion Markup Language) чи OpenID Connect можуть бути використані для обміну ідентифікаційною інформацією.

- Одноразова Аутентифікація (Single Sign-On, SSO):

Реалізуйте SSO для того, щоб користувач міг увійти в систему лише одного разу та мати доступ до різних ресурсів без повторної аутентифікації. Це полегшить використання системи та зменшить ризик забуття паролів.

- Локальне управління даними:

Виводьте локальні ролі та інші ідентифікаційні атрибути в ідентифікаційну систему, де це можливо. Це дозволяє централізовано управляти даними про користувачів.

- Багатофакторна аутентифікація (mfa):

Інтегруйте можливості багатофакторної аутентифікації в ідентифікаційну систему. Це додасть додатковий рівень безпеки, особливо для осіб з доступом до важливих ресурсів.

- Синхронізація даних:

Забезпечте регулярну синхронізацію даних між ідентифікаційною системою та локальними системами для підтримки актуальності інформації.

- Управління правами доступу:

Використовуйте ідентифікаційну систему для управління правами доступу користувачів. Це дозволяє ефективно контролювати, хто має доступ до яких ресурсів у вашій системі.

- Логування та аудит:

Реалізуйте механізми логування та аудиту для відстеження дій користувачів у вашій системі. Це важливо для виявлення незвичайної або підозрілої активності.

- Інтеграція з каталогами та керуванням персоналом:

Якщо у вас є система керування персоналом або каталог корпоративних даних, інтегруйте їх з ідентифікаційною системою для автоматизації процесу управління користувачами.

Інтеграція з ідентифікаційною системою допомагає оптимізувати процеси автентифікації та авторизації, забезпечуючи при цьому високий рівень безпеки та ефективність в управлінні ідентифікацією користувачів.

2.2.5. Роль токенів в системі та їх використання

- Автентифікація та авторизація:

Токени використовуються для ідентифікації користувача під час входу в систему (автентифікації) і визначення його прав доступу (авторизації). Після успішної аутентифікації користувач отримує токен, який потім використовується для отримання доступу до різних ресурсів в системі.

- Безпека сесій:

Токени можуть бути використані для забезпечення безпеки сесій. Наприклад, використання токенів з одноразовим використанням допомагає уникнути атак типу перехоплення сесії.

- Арі та мікросервіси:

Використання токенів у взаємодії між клієнтами та серверами в API або серед мікросервісів. Токени можуть служити як механізм автентифікації для забезпечення безпеки обміну інформацією.

- Одноразові токени для відновлення паролю:

Використання одноразових токенів для відновлення паролю. Коли користувач забуває пароль, система може вислати йому тимчасовий токен для встановлення нового пароля.

- Безпека клієнтських додатків:

Токени можуть використовуватися для забезпечення безпеки клієнтських додатків, які взаємодіють з веб-серверами чи іншими ресурсами. Наприклад, OAuth та OpenID Connect використовують токени для авторизації та отримання доступу.

- Jwt (json web tokens):

JWT — це формат токена, який використовується для передачі інформації між сторонами у вигляді JSON об'єкта. Вони часто використовуються в веб-розробці для передачі структурованої інформації, такої як права доступу чи інші клейові дані.

- Багатофакторна аутентифікація:

Токени можуть бути частиною багатофакторної аутентифікації, де додаткові токени (наприклад, SMS-коди або токени, отримані через мобільні додатки) використовуються для підтвердження особи.

- Захист від csrf-атак (cross-site request forgery):

Використання токенів може захищати від атак, пов'язаних з CSRF, де атакуюча сторінка намагає виконати незаконний запит від імені автентифікованого користувача.

- Ідентифікація клієнта в мобільних додатках:

В мобільних додатках токени часто використовуються для ідентифікації клієнта і обміну даними між клієнтом та сервером.

Токени допомагають покращити безпеку систем, забезпечуючи ефективний механізм ідентифікації, автентифікації та авторизації, а також підтримуючи інші аспекти інформаційної безпеки.

2.2.6. Безпека та Відновлення

a. Безпека

- Шифрування даних:

Використовуйте сучасні криптографічні алгоритми для шифрування чутливої інформації під час передачі та зберігання.

- Багатофакторна Аутентифікація:

Впроваджуйте системи багатофакторної аутентифікації для забезпечення додаткових рівнів ідентифікації та захисту від несанкціонованого доступу.

- Регулярні Аудити та Моніторинг:

Проводьте регулярні аудити безпеки для виявлення потенційних слабких місць та моніторінг системи на предмет аномальної активності.

- Захист від атак:

Використовуйте заходи безпеки, такі як захист від SQL-ін'єкцій, кросс-сайтового скриптінгу (XSS), кросс-сайтового request forgery (CSRF) та інших відомих атак.

- Ведення бекапів та відновлення:

Регулярно створюйте резервні копії системи та даних для можливості відновлення інформації в разі втрати або пошкодження даних.

- Управління правами доступу:

Встановлюйте строгі політики управління правами доступу, щоб забезпечити принцип найменших привілеїв та обмежити доступ до інформації.

- Патчі та оновлення:

Своєчасно встановлюйте патчі та оновлення для всіх компонентів системи, щоб усунути виявлені уразливості та зменшити ризик атак.

b. Відновлення:

- Бізнес-план відновлення після аварії:

Розробляйте та впроваджуйте плани відновлення після аварії, які включають процеси відновлення та відновлення даних.

- Регулярні тести відновлення:

Проводьте тести відновлення для перевірки ефективності та реалізованості ваших планів відновлення.

- Забезпечення резервних копій:

Забезпечуйте безперервність операцій шляхом зберігання регулярних резервних копій інформації та системних конфігурацій.

- Автоматизація відновлення:

Автоматизуйте процеси відновлення, де це можливо, щоб зменшити час відновлення та помилкові ситуації.

- Планування для відновлення життєважливих функцій (rto, rpo):

Визначте час відновлення (Recovery Time Objective - RTO) та точку відновлення (Recovery Point Objective - RPO) для критичних функцій інформаційної системи.

- Співпраця з постачальниками:

Перевіряйте плани відновлення ваших ключових постачальників послуг та сторонніх партнерів.

- Тренування персоналу:

Проводьте тренування та навчання персоналу з питань відновлення, щоб забезпечити ефективну реакцію на аварійні ситуації.

Ці практики сприяють створенню високопродуктивної та безпечної інформаційної системи, яка може ефективно впоратися з потенційними загрозами та відновитися після інцидентів.

2.4. Висновки до другого розділу

1. В пункті 2.1 було описано теоретичні основи оптимізації процесу автентифікації.

2. В пункті 2.2. було описано сучасні основні проблеми автентифікації користувачів та способи їх вирішення.

3. В пункті 2.3. було описано теоретичні основи інтеграції програмного модулю автентифікації користувачів з використанням непрозорих токенів.

РОЗДІЛ 3

ОПИС РОЗРОБКИ РІШЕННЯ

3.1. Технологічний стек і засоби реалізації

Вибір мови програмування — критичне рішення при розробці будь-якого програмного модуля, особливо присвяченого аутентифікації користувачів та безпеці. При втіленні запропонованої системи класифікації токенів для віддаленої аутентифікації Java обрана основною мовою програмування з численних причин.

– **Платформова незалежність**

Філософія "напишіть один раз, запустіть будь-де" в Java дозволяє виконувати скомпільований код на будь-якому пристрої з віртуальною машиною Java (JVM). Ця платформова незалежність є важливою в контексті модуля аутентифікації користувачів, який може бути використаний в різних середовищах та на різних пристроях, забезпечуючи гнучкість та масштабованість.

– **Стійка та зріла Екосистема**

Java визначається як мова програмування з великою екосистемою бібліотек, фреймворків і інструментів. Використання цієї розвиненої екосистеми спрощує процес розробки, надаючи готові компоненти та інструменти, які покращують ефективність і скорочують час втілення.

– **Сильні заходи безпеки**

Безпека має велике значення у сфері аутентифікації користувачів, і Java відома своїми вбудованими заходами безпеки. Мова включає різноманітні механізми безпеки, включаючи міцний менеджер безпеки, криптографічні

бібліотеки та підтримку безпечних практик кодування, що ідеально відповідає вимогам безпеки запропонованого модуля.

– **Підтримка спільноти**

Java користується активною та живою спільнотою розробників. Наявність обширних ресурсів, форумів та підтримки від спільноти гарантує швидке вирішення викликанних проблем під час втілення, сприяє інноваціям та колективному вирішенню проблем.

– **Об'єктно-орієнтований парадигма**

Об'єктно-орієнтована природа Java сприяє модульному та масштабованому проектуванню коду. Враховуючи складність системи класифікації токенів, об'єктно-орієнтований підхід сприяє створенню добре організованого, підтримуваного та розширюваного коду, підтримуючи довгострокову стійкість та адаптивність.

– **Надійність на рівні підприємства**

Java є мовою, якій довіряють підприємства для створення надійних та стійких систем. Її доведений успіх у великообсягових додатках, разом із функціями, такими як збірка сміття та сильне управління пам'яттю, внесли впевненість у надійності реалізованого модуля аутентифікації.

– **Сумісність з існуючими системами**

Сумісність Java з різноманітними базами даних, веб-серверами та системами підприємств забезпечує безшовну інтеграцію з існуючою інфраструктурою. Ця взаємодія є важливою для успішного розвитку системи класифікації токенів, дозволяючи їй легко вписуватися в різноманітні інформаційні мережі.

Підсумовуючи, комбінація незалежності від платформи, обширного екосистеми, потужних засобів безпеки, підтримки спільноти, об'єктно-орієнтованої парадигми, надійності на рівні підприємства та сумісності робить Java стратегічним та надійним вибором для розробки безпечної, масштабованої та сумісної системи класифікації токенів для віддаленої аутентифікації.

Інструмент збірки: Maven

Вибір між Maven і Gradle часто залежить від різних факторів і вимог в кваліфікаційні роботі. Обидва інструменти, Maven і Gradle, є популярними інструментами автоматизації збірки та управління залежностями в екосистемі Java, але вони мають різні підходи та функціонал.

- **Стандартні конвенції:**

Maven базується на концепції "конвенція над конфігурацією", що дозволяє визначити стандартні шляхи та структури додатків. Це спрощує процес конфігурації та збірки.

- **Зручне управління залежностями:**

Maven автоматично керує залежностями додатку, завантажуючи їх з централізованого репозитарію (Maven Central). Це спрощує вирішення залежностей і підтримує порядок.

- **Широкий вибір плагінів:**

Maven має велику кількість вбудованих плагінів для різноманітних завдань, таких як компіляція, тестування, пакування та інші. Це полегшує розширення функціональності додатку.

- **Легка інтеграція з іншими інструментами:**

Maven легко інтегрується з популярними інтегрованими середовищами розробки (IDE), такими як Eclipse, IntelliJ IDEA та NetBeans. Є також підтримка для інших інструментів.

- **Централізований репозитарій:**

Maven Central - це централізований репозитарій для зберігання і розповсюдження бібліотек і плагінів. Це полегшує обмін та використання сторонніх компонентів.

- **Стабільність та надійність:**

Maven використовується великою кількістю програмних модулів у всьому світі та вважається стабільним та надійним інструментом для автоматизації збірки.

- Широкий доступ до документації та спільноти:

Maven має обширну документацію та активну спільноту, що полегшує вирішення проблем та знаходження відповідей на питання.

- Історія та досвід використання:

Maven є одним із найдавніших інструментів у світі Java і здобув значний досвід використання та підтримки великої кількості програмних модулів.

- Освідченість та стандартизація:

Завдяки своїм конвенціям та стандартам, Maven робить робочий процес більш освідченим та сприяє зрозумілості програмного модулю для нових учасників команди.

Таблиця 3.1

Порівняльна таблиця інструментів автоматизації збірок проєктів

Характеристика	**Maven**	**Gradle**
Спрямованість	Стабільність та Впевненість у Стандартах	Гнучкість та Повна Контроль над Конфігурацією
Мова Конфігурації	Простота та Зрозумілість (XML)	Гнучкість та Виразність (Groovy/Kotlin DSL)
Цикл Життя Збірки	Стандартні та Надійні Фази	Гнучкість та Індивідуалізація Завдань
Залежності	Простота Оголошення (pom.xml)	Гнучкість Оголошення (build.gradle)
Репозитарій	Maven Central	Можливість Використання Різних Репозитаріїв
Підтримка IDE	Інтеграція з Великою Кількістю IDE	Інтеграція З Всіма Популярними IDE
Плагіни	Широка База Вбудованих Плагінів	Гнучкість та Легкість Створення Власних Плагінів

Продовження таблиці 3.1

Доступність плагінів	Багато Років Досвіду, Впровадженість	Можливість Використання Maven-Плагінів
Гнучкість в Конфігурації	Орієнтація на Конвенції та Стандарти	Більша Гнучкість та Настроювання
Комунітет та Підтримка	Велика Кількість Користувачів та Активність	Активний Комунітет, Швидке Вирішення Проблем, Регулярні Оновлення
Відсутність Кешування	Використання .m2 репозитарію для Кешування	Власний Кешований Механізм для Швидшої Збірки
Вибір Мови Конфігурації	Простота XML	Гнучкість Groovy/Kotlin DSL
Підтримка Kotlin DSL	Немає	Повна Підтримка Kotlin DSL
Зрозумілість Синтаксису	Простий та Зрозумілий Синтаксис (XML)	Виразний та Читабельний Синтаксис (Groovy/Kotlin DSL)
Простота Розширення	Завдатки Легко Розширюються	Легко Використовувати Власні Завдатки та Розширення
Швидкодія з Операційною Системою	Швидше виконання в операційних системах на основі Unix	Оптимізоване для Швидкості на Всіх Операційних Системах

Допоміжний фреймворк: Spring Framefork

Використання Spring Framework у розробці програмного забезпечення надає ряд переваг, роблячи його популярним вибором для побудови корпоративних додатків.

– **Модульність та легкість:**

Модульний Дизайн. Spring сприяє модульному та організованому дизайну додатків. Він побудований на модульній архітектурі, що дозволяє використовувати лише ті компоненти, які їм потрібні, поліпшуючи збереженість та масштабованість.

Легкий Контейнер: Контейнер Spring є легким і неінтрузивним, що дозволяє легко інтегруватися з існуючими системами.

– **Інверсія контролю (IoC):**

Слабка зв'язаність. IoC контейнер Spring керує об'єктами додатка та їх взаємовідносинами, сприяючи слабкій зв'язаності. Це полегшує заміну або оновлення компонентів без впливу на весь систему.

– **Впровадження Залежностей (DI):**

Підтримка повторного використання. Впровадження залежностей дозволяє вводити залежності в компоненти, сприяючи повторному використанню та тестуванню коду.

Зменшення зайвого коду. DI зменшує необхідність у зайвому коді, що призводить до більш чистому коду.

– **Доступ до Даних:**

Декларативне управління транзакціями. Spring спрощує управління транзакціями за допомогою анотацій або конфігурацій XML, зменшуючи складність управління транзакціями баз даних.

Інтеграція з ORM-Фреймворками. Spring легко інтегрується з популярними об'єктно-реляційними фреймворками, такими як Hibernate, забезпечуючи ефективний доступ до даних.

– **Фреймворк моделювання-вигляд-контролер (MVC):**

Розробка Веб-Додатків. Spring MVC пропонує потужний та гнучкий фреймворк для розробки веб-додатків. Він дотримується патерну MVC, що полегшує розробку та збереження веб-додатків.

– **Безпека:**

Декларативна безпека. Spring Security надає всебічний та настроюваний фреймворк аутентифікації та контролю доступу. Він дозволяє визначати вимоги до безпеки за допомогою анотацій або конфігурацій XML.

– **Спрощення розробки Java EE:**

Усунення зайвого коду. Spring усуває багато зайвого коду, пов'язаного з розробкою Java EE, дозволяючи фокусуватися на бізнес-логіці, а не на інфраструктурних питаннях.

– **Інтеграція з Іншими Технологіями:**

Можливості інтеграції. Spring легко інтегрується з різними технологіями та фреймворками, включаючи системи обміну повідомленнями, джерела даних та сторонні бібліотеки.

Архітектура мікросервісів. Spring Boot сприяє розробці мікросервісів, надаючи підхід конвенція-перед-конфігурацією та вбудованими серверами додатків.

– **Активна спільнота та екосистема:**

Підтримка спільноти. У Spring велика та активна спільнота, яка надає підтримку, ресурси та велику кількість сторонніх розширень і плагінів.

Постійна еволюція. Екосистема Spring постійно розвивається, регулярно впроваджуючи нові оновлення та функції для задоволення змінних потреб розробників.

– **Підтримка тестування:**

Сприяє юніт-тестуванню. Принципи дизайну Spring, такі як IoC та DI, полегшують написання юніт-тестів для компонентів, що призводить до надійного коду.

Узагальнюючи, Spring Framework пропонує комплексне рішення для побудови надійних, масштабованих та збережених корпоративних застосунків. Його модульний дизайн, підтримка ключових корпоративних функцій та активна спільнота роблять його популярним вибором для багатьох розробників та організацій.

3.2. Детальний опис реалізації програмного модуля автентифікації користувачів з використанням непрозорих токенів

Загальна інформація про проєкт. Додаток А.

Ідентифікація: Програмний модуль ідентифікується як auth-module-opaque-tokens в групі org.yakymchuk-eugene.

Версія: Версія програмного модулю вказана як 0.0.1.

Модель: Використовується POM-модель версії "4.0.0".

Властивості програмного модулю: Властивості включають версії Java, компілятора Maven та інші ключові значення.

Вкладені залежності: spring-boot-starter-parent версії "3.1.1".

Компоненти: Залежності та управління залежностями:

Задано керування залежностями для програмного модуля, включаючи версії бібліотек, таких як Spring Security OAuth, Spring Boot OAuth2 Resource Server, Springdoc OpenAPI, та Google Guava.

Конфігурація збірки: Визначено конфігурацію для збірки програмного модулю, включаючи управління плагінами, такий як frontend-maven-plugin.

Цей POM-файл відображає структуру, залежності та налаштування програмного модулю, які використовуються в розробці серверу авторизації на основі Spring та інших технологій.

Ідентифікація: Програмний модуль ідентифікується як auth-module в групі org.yakymchuk-eugene з версією "0.0.1".

Назва і опис: Програмний модуль має назву "auth-module" і опис "Програмний модуль аутентифікації користувачів на базі непрозорих токенів".

Властивості: Визначені різні властивості, такі як версії компілятора Maven, дата збірки, параметри збірки клієнта тощо.

Профілі: Визначено кілька профілів для збірки та копіювання фронтенд додатку в різних середовищах (dev, test, prod).

Залежності: Вказані залежності програмного модуля включають бібліотеки для веб-розробки (Spring Boot, Thymeleaf), безпеки (Spring Security, OAuth), роботи з даними (Spring Data JPA, Liquibase), роботи з Redis, та інші.

Плагіни збірки: Використовуються плагіни для збірки програмного модулю, включаючи Spring Boot Maven Plugin для збірки і конфігурації, Liquibase Maven Plugin для керування базою даних, і Frontend Maven Plugin для збірки та копіювання фронтенду.

Ресурси: Визначено ресурси програмного модулю, такі як файли бази даних та інші конфігураційні файли.

Цей POM-файл розкриває комплексну конфігурацію програмного модулю, яка враховує різні аспекти розробки, включаючи роботу з фронтендом, базою даних, безпекою та інші.

Створив основний клас додатку. А саме Application.class. Він включає:

- @SpringBootApplication: Аннотація, яка об'єднує кілька аннотацій в одній. У цьому випадку, вона включає @Configuration, @EnableAutoConfiguration, і @ComponentScan. Це забезпечує конфігурацію Spring Boot, автоматичне налаштування та сканування компонентів в пакеті та його підпакетах.

- Метод main, який є входовою точкою для виконання програми. Викликає SpringApplication.run(Application.class, args), щоб запустити додаток Spring Boot.

- SpringApplication.run(Application.class, args);: Запускає додаток Spring Boot, ініціалізує контейнер і виконує конфігурацію, включену за допомогою аннотації @SpringBootApplication.

Основний клас, що визначає конфігурацію безпеки. SecurityConfig.class.

- @EnableWebSecurity та @EnableMethodSecurity: Аннотації, які позначають цей клас як конфігурацію безпеки та дозволяють метод-рівень безпеки в Spring.

- @RequiredArgsConstructor: Аннотація Lombok, яка автоматично генерує конструктор з обов'язковими аргументами для класу.

- SecurityFilterChain: Конфігурація ланцюга фільтрів безпеки для застосовуваного http об'єкта.

- Аннотації та поля для налаштувань безпеки, такі як LOGIN_PAGE, PERMIT_ALL_PATTERNS.

- Поля, що інжектяться через конструктор: Використовуються для введення залежностей, таких як CustomOAuth2UserService, CustomUserDetailsService, PasswordEncoder, AuthorizationServerProperties.

- Метод defaultSecurityFilterChain: Конфігурація засобів безпеки. Включає налаштування для OAuth2, виключення CSRF, налаштування для аутентифікації та авторизації.

- Метод initializeHandlers: Ініціалізація обробників подій для успішної та неуспішної аутентифікації.

- Кастомні обробники для успішної та неуспішної аутентифікації.

AuthorizationServerConfig

Цей клас визначає конфігурацію авторизаційного сервера та його взаємодію з іншими компонентами додатку Spring Boot для обробки та інтроспекції токенів авторизації.

- @Configuration і @RequiredArgsConstructor: Клас-конфігурація, який використовує Lombok для автоматичного генерування конструктора з обов'язковими аргументами.

- @Bean: Метод, який створює бін для об'єкта SecurityFilterChain, який конфігурує фільтри безпеки для авторизаційного сервера.

- @Order(Ordered.HIGHEST_PRECEDENCE): Вказує порядок, в якому цей фільтр застосовується.

- OAuth2AuthorizationServerConfigurer: Конфігуратор для налаштування параметрів авторизаційного сервера.

- RequestMatcher і HttpSecurity: Конфігурація захисту ресурсів, включаючи інтроспекційний ендпоінт.

- AuthorizationServerSettings: Налаштування авторизаційного сервера, включаючи ендпоінт інтроспекції токенів.

- introspectionResponse: Метод для обробки відповіді інтроспекції токенів. Визначає, які дані повертаються при інтроспекції токенів.

Далі потрібно налаштувати CORS політику, для коректної міждоменної комунікації:

- @Bean: Вказує, що цей метод повертає об'єкт, який буде зареєстрований як бін у контейнері Spring.

- FilterRegistrationBean<CorsFilter>: Спеціалізований клас Spring для реєстрації фільтра у контексті веб-додатка.

- UrlBasedCorsConfigurationSource: Конфігураційна компонента для CORS, яка базується на URL.

- CorsConfiguration: Клас, який містить налаштування CORS для конкретного паттерна.

- CorsProperties: Параметри CORS, які, очевидно, визначаються в іншому місці у вашому коді.

- FilterRegistrationBean.setOrder(Ordered.HIGHEST_PRECEDENCE): Вказує порядок виклику фільтра, де Ordered.HIGHEST_PRECEDENCE означає найвищий пріоритет

Ролі, які використовують для ідентифікації ресурсів:

- **Власник ресурсу(Resource owner):** власник ресурсу – це користувач, який авторизує програму для доступу до свого облікового запису. Доступ програми до облікового запису користувача обмежено обсягом наданого дозволу (наприклад, доступ для читання або запису)

- **Клієнт(Client):** клієнт – це програма, яка хоче отримати доступ до облікового запису користувача. Перш ніж він зможе це зробити, він має бути авторизований користувачем, і авторизація має бути підтверджена API.

- **Сервер ресурсів(Resource server):** сервер ресурсів містить захищені облікові записи користувачів.

- **Сервер авторизації(Auth server):** Сервер авторизації перевіряє особу користувача, а потім видає маркери доступу до програми.

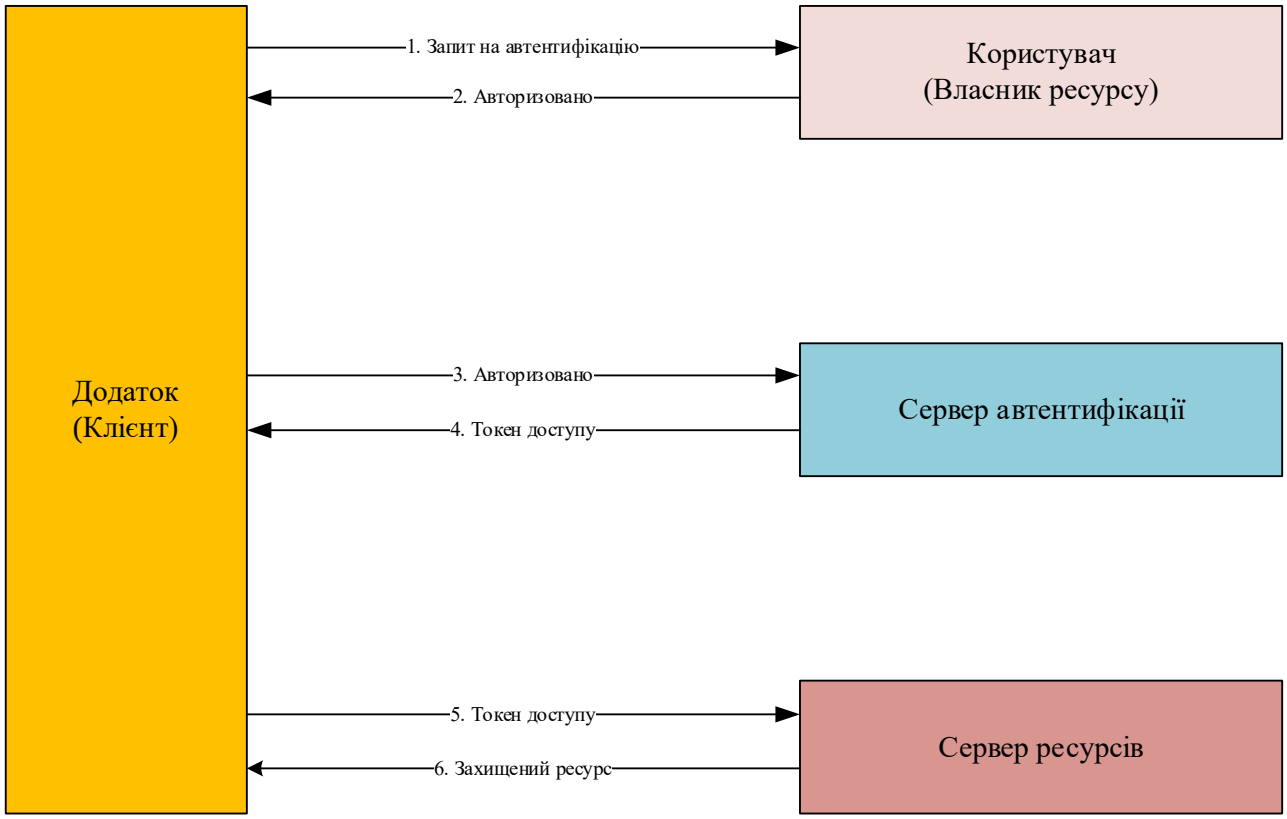


Рис. 3.1. Абстрактна діаграма роботи модуля автентифікації

- Додаток запитує у користувача авторизацію доступу до ресурсів сервісу

- Якщо користувач авторизував запит, програма отримує дозвіл на авторизацію

- Додаток запитує токен доступу від сервера авторизації (API), представляючи автентифікацію своєї власної ідентифікації та надання авторизації

- Якщо ідентифікатор програми автентифікований і надання авторизації є дійсним, сервер авторизації (API) видає маркер доступу до програми. Авторизація завершена.

- Програма запитує ресурс із сервера ресурсів (API) і представляє токен доступу для автентифікації

- Якщо токен доступу дійсний, сервер ресурсів (API) надає ресурс програмі

Реєстрація додатку

Перш ніж використовувати модуль, ви повинні зареєструвати свій додаток.

Необхідна інформація:

- Назва програми

- Веб-сайт програми

- URI перенаправлення або URL зворотного виклику

URI перенаправлення – це місце, куди служба перенаправлятиме користувача після того, як він авторизує (або відхилить) ваш додаток, і, отже, ту частину вашого додатку, яка оброблятиме коди авторизації або маркери доступу.

Ідентифікатор клієнта та секретний ключ клієнта

Після реєстрації служба видасть облікові дані клієнта у вигляді ідентифікатора клієнта та секрету клієнта. Ідентифікатор клієнта – це загальнодоступний рядок, який використовується API служби для ідентифікації програми, а також використовується для створення URL-адрес авторизації, які надаються користувачам. Секрет клієнта використовується для перевірки автентичності програми в API служби, коли програма запитує доступ до облікового запису користувача, і має бути закритою між програмою та API.

Код авторизації

Тип надання коду авторизації є найбільш часто використовуваним, оскільки він оптимізований для програм на стороні сервера, де вихідний код не є публічним, і можна зберегти конфіденційність секрету клієнта. Це потік на основі перенаправлення, що означає, що програма має бути здатна взаємодіяти з агентом користувача (тобто веб-браузером користувача) і отримувати коди авторизації API, які направляються через агента користувача.

Результат успішного проходження автентифікації:


```
{
    "active": true,
    "sub": "admin@example.com",
    "aud": [
        "test-client"
    ],
    "nbf": "2023-05-21T16:05:19.823028414Z",
    "scopes": [
        "write.scope",
        "read.scope"
    ],
    "iss": "http://localhost:7777",
    "exp": "2023-05-21T16:35:19.823028414Z",
    "iat": "2023-05-21T16:05:19.823028414Z",
    "jti": "b329da0b-d2ad-4df3-b491-2f8f0715d028",
    "clientId": "test-client",
    "tokenType": "Bearer",
    "principal": {
        "password": null,
        "username": "admin@example.com",
        "authorities": [],
        "accountNonExpired": true,
        "accountNonLocked": true,
        "credentialsNonExpired": true,
        "enabled": true,
        "id": "05a8624e-22be-4ec6-9f3a-4b0a20e5409e",
        "firstName": "Admin",
        "secondName": "Admin",
        "middleName": null,
        "birthday": "1998-07-14",
        "avatarUrl": null,
        "oauthAttributes": null,
        "name": "admin@example.com",
        "attributes": null,
        "email": "admin@example.com"
    },
    "authorities": []
}
```

- Active -активний токен чи ні
- Sub – унікальний ідентифікатор
- Aud – кому призначений токен
- Nbf – час, до якого повинний бути використаний токен
- Scopes – перелік дозволених прав
- Iss – видавець токену
- Jti – унікальний ідентифікатор токену
- clientId – ідентифікатор клієнта
- tokenType – тип токена
- principal - користувач
    - password - пароль
    - username – ім’я користувача
    - accountNonExpired – чи не пройшов термін дії аккаунту користувача
    - accountNonLocked – чи не заблокований аккаунт користувача
    - credentialsNonExpired – чи активні авторизаційні дані
    - enabled – чи актоваваний аккаунт
    - id – унікальний ідентифікатор
    - firstName – ім’я
    - secondName - прізвище
    - middleName – по-батькові
    - birthday – день народження
    - avatarUrl – посилання на картинку
    - attributes - атрибути
    - email – електронна адреса
- authorities – перелік ролей або прав

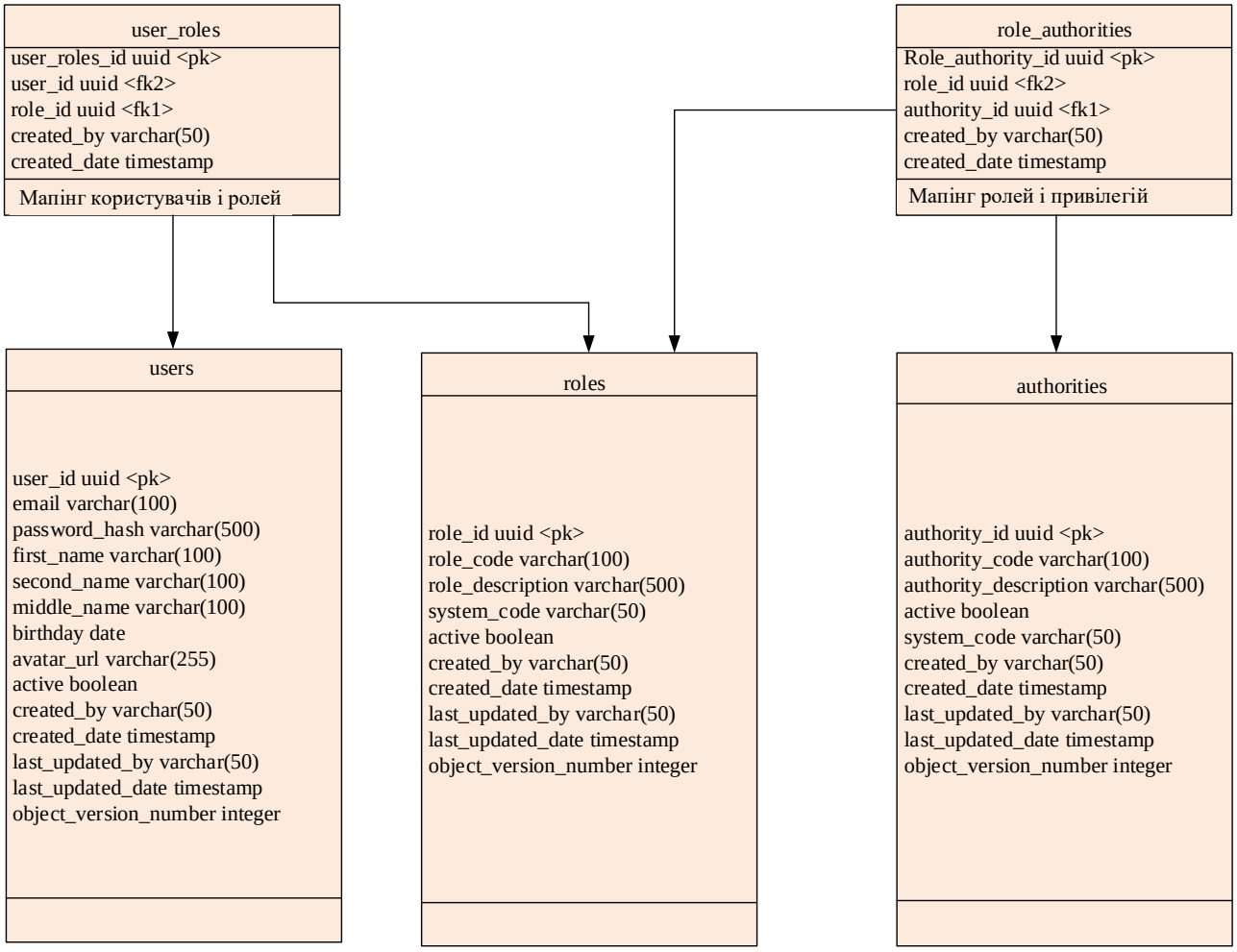


Рис. 3.2. Фізична схема бази даних рольової моделі

Дана рольова модель дозволяє самостійно створювати ролі, даючи тій чи іншій ролі доступ до конкретних ресурсів(як правило ендпоінтів). В цьому випадку звичайним набором ролей не можливо обійтися. Тому потрібно створити список всіх ресурсів. Далі на основі цього списку можна створити список привілей, так звана міні-роль, яка надає доступ до конкретного ресурсу. На кожний ендпоінт – свій привілей. Далі з цього списку привілей можна створити групи привілей – вони і будуть ролями, які можна призначити користувачу.

Привілеї доступні для користувача:

- GET_OWN_DATA – Привілей, який дозволяє отримати дані поточного користувача.

- CHANGE_OWN_DATA – Привілей, який дозволяє змінювати дані поточного користувача.

- CHANGE_OWN_PASSWORD – Привілей, який дозволяє змінювати пароль поточного користувача.

- DELETE_OWN_ACCOUNT – Привілей, який дозволяє видаляти аккаунт поточного користувача.

Приклад відповіді серверу автентифкації з ролями та привілеями:


```
{
 "active": true,
 "sub": "zhenyakymchuk@gmail.com",
 "aud": [
  "test-client"
 ],
 "nbf": "2023-11-26T15:23:22.697355152Z",
 "scopes": [
  "write.scope",
  "read.scope"
 ],
 "iss": "http://localhost:7777",
 "exp": "2023-11-26T15:28:22.697355152Z",
 "iat": "2023-11-26T15:23:22.697355152Z",
 "clientId": "test-client",
 "tokenType": "Bearer",
 "principal": {
  "id": "c74f6ec6-907e-11ee-b9d1-0242ac120002",
  "firstName": "Eugene",
  "secondName": "Yakymchuk",
  "middleName": "Anatoliyovych",
  "birthday": null,
  "avatarUrl": "/eugene.png",
  "username": "zhenyakymchuk@gmail.com",
  "email": "zhenyakymchuk@gmail.com",
```

```
    "authorities": [
      {
        "authority": "CHANGE_OWN_DATA"
      },
      {
        "authority": "CHANGE_OWN_PASSWORD"
      },
      {
        "authority": "CHANGE_OWN_ACCOUNT"
      },
      {
        "authority": "GET_OWN_DATA"
      }
    ]
  }
}
```


Контроль доступу до ендпоінтів виконується за допомогою анотації @PreAuthorize

Приклад:

```
@RestController
public class TestController {

    @GetMapping("/test")
    @PreAuthorize("hasAnyAuthority('GET_OWN_DATA')")
    public String test() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        return authentication.getName();
    }
}
```

Тепер, коли успішно пройдено автентифікацію, користувач отримає доступ до ресурсу.

Якщо ж замінити привілей, на будь-який інший, якого немає в користувача, то, наприклад

@PreAuthorize(“hasAnyAuthority(‘NOT_EXISTING_AUTHORITY’)”, то після автентифікації і спроби отримати доступ до цього ресурсу, користувач отримає помилку 403 – доступ заборонений.

3.3. Тестування

Тестування безпеки та вразливостей систем автентифікації, включаючи перехоплення непрозорих токенів, є важливою частиною розробки безпеки в додатках.

Типи тестування:

1. Модульне тестування:

Опис: Модульне тестування спрямоване на перевірку окремих компонентів коду. Кожна функція, клас чи модуль перевіряється окремо, щоб впевнитися в їх коректності.

Методи: Використання тестових наборів для запуску тестів на кожен модуль, враховуючи різноманітні входи та умови.

2. Інтеграційне тестування:

Опис: Цей вид тестування оцінює взаємодію між різними модулями програми, переконуючись, що вони працюють у спільноті без конфліктів.

Методи: Тестування взаємодії модулів на різних рівнях, таких як API-виклики або обробка подій.

3. Системне тестування:

Опис: Системне тестування виконується на рівні програми в цілому, перевіряючи, як всі її частини працюють разом в реальних умовах експлуатації.

Методи: Тестування функціональності, продуктивності, стійкості та сумісності.

4. Функціональне тестування:

Опис: Тестування функціональності перевіряє, чи програма відповідає визначеним вимогам та чи здатна ефективно виконувати заплановані завдання.

Методи: Використання тестових сценаріїв для відтворення реальних умов використання.

5. Навантажувальне тестування:

Опис: Тестування на максимальне навантаження допомагає визначити межі продуктивності та оцінити, як програма веде себе під великою кількістю одночасних користувачів або операцій.

Методи: Створення тестових сценаріїв, що моделюють велику кількість запитів та взаємодій.

6. Тестування на витривалість:

Опис: Тестування на витривалість перевіряє, як програма веде себе протягом тривалого часу, чи не виникають пам'ятеві витоки та ресурсоємність.

Методи: Запуск програми на довгий період часу та моніторинг її роботи.

7. Тестування на безпеку:

Опис: Тестування на безпеку допомагає виявити та усунути потенційні вразливості, щоб запобігти неправомірному доступу та атакам.

Методи: Аудит коду, тестування на вразливості, виявлення та усунення можливих шляхів атаки.

8. Автоматизоване тестування:

Опис: Використання інструментів для автоматизації виконання тестів, що дозволяє швидше та ефективніше перевіряти функціонал.

Методи: Створення автоматизованих тестових скриптів, використання інструментів автоматизації.

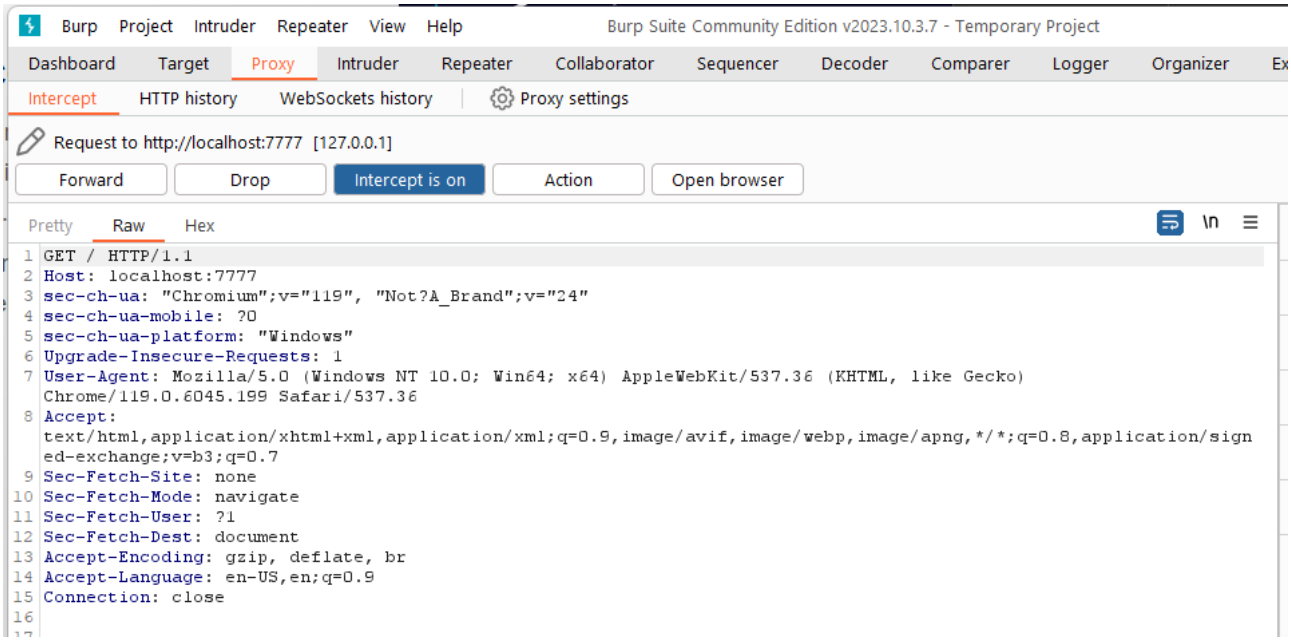
9. Тестування на різних платформах та пристроях:

Опис: Визначення того, як програма працює на різних операційних системах, пристроях та браузерах.

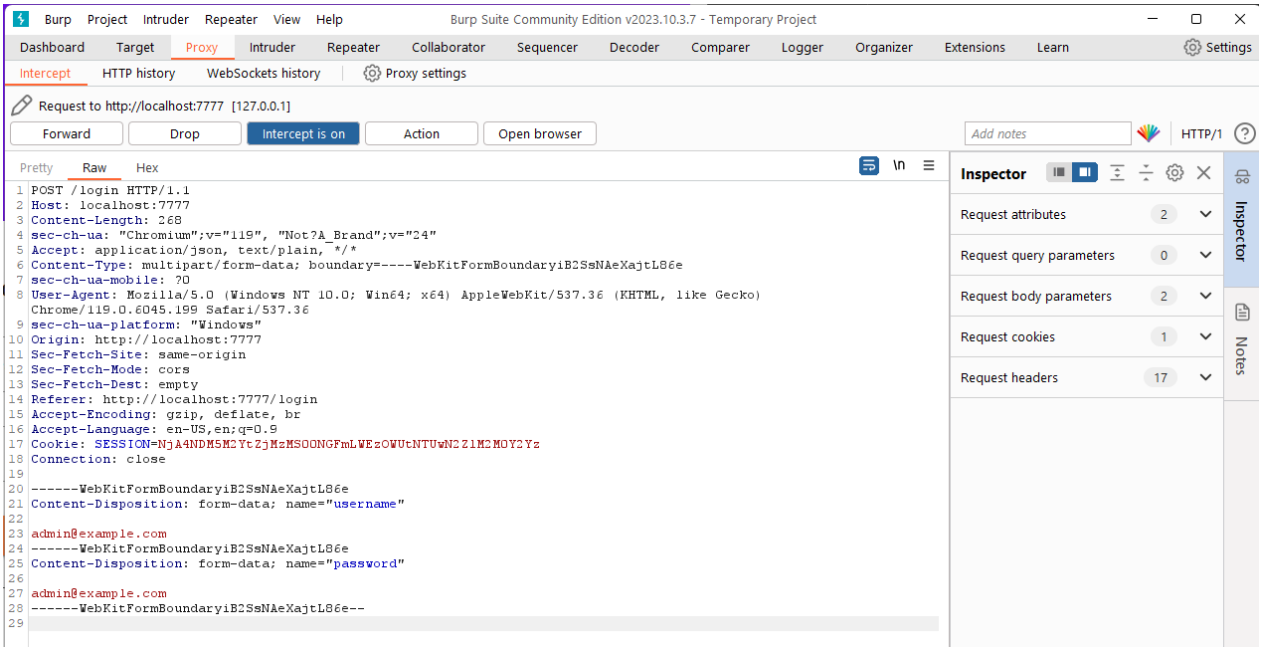
Методи: Тестування на різних конфігураціях апаратного та програмного забезпечення.

Тестування з використанням Burp Suite:

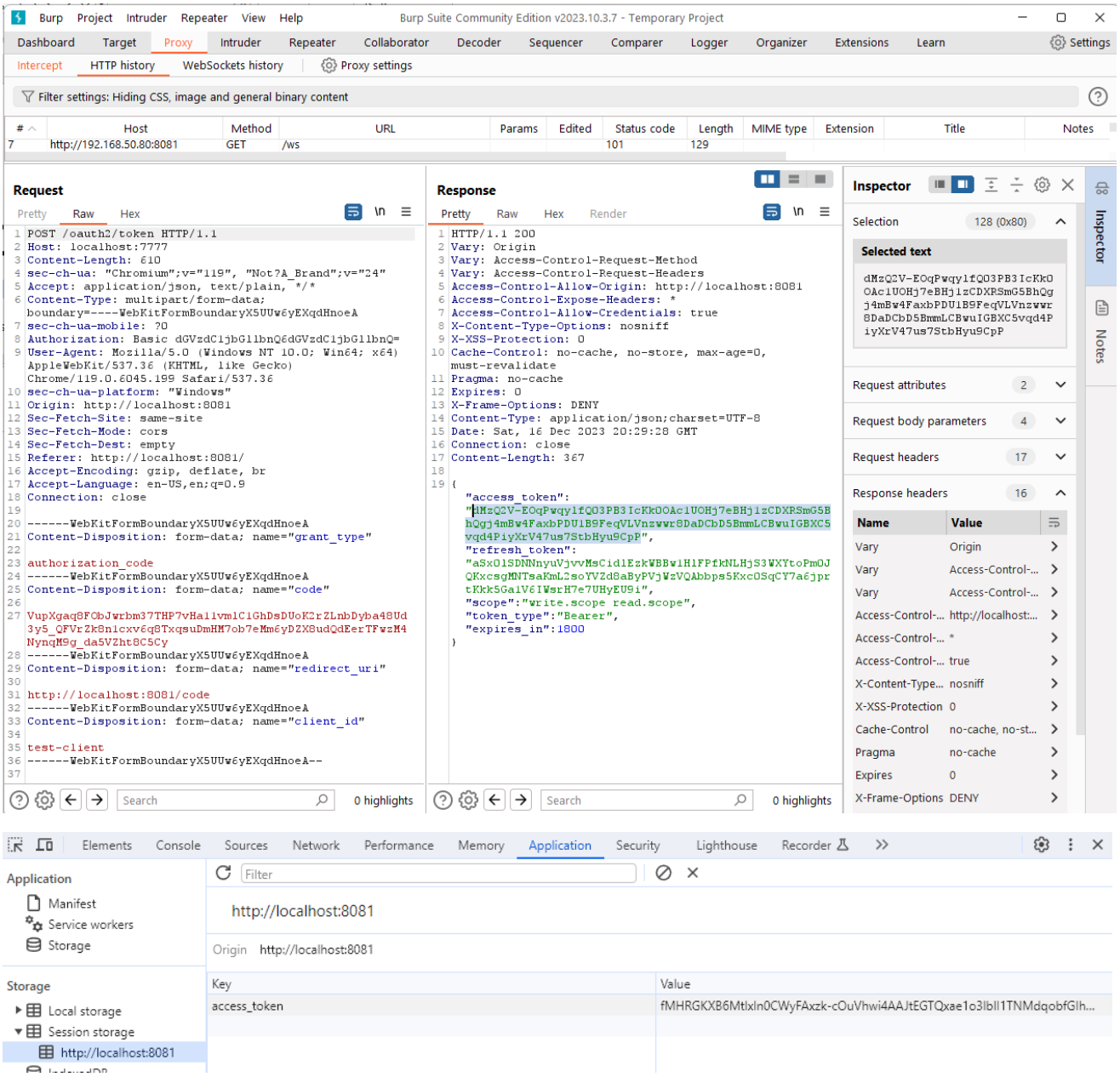
Запит на сторінку авторизації



Введення логіну і паролю



Непрозорий токен записаний у сховище сесій. Як видно, токен унікальний та не містить жодної корисної інформації. Також його не можливо декодувати, оскільки він складається з випадково підібраних символів та чисел.



3.4. Оцінка ефективності та порівняльна таблиця запропонованого рішення

1. Безпека

а. Висока якість шифрування та захист даних з використанням непрозорих токенів.

b. Успішна захист від атак, таких як перехоплення, фішинг, атаки на зберігання.

c. Ефективна валідація та перевірка цілісності токенів.

2. Відсутність повторного використання

a. Використання непрозорих та одноразових токенів для ускладнення повторного використання.

3. Швидкість та продуктивність

a. Швидка генерація та перевірка токенів.

b. Мінімальний вплив на продуктивність системи під час автентифікації.

4. Легкість інтеграції

a. Спрощення процесу інтеграції модуля з існуючою інфраструктурою.

b. Підтримка стандартів, таких як OAuth 2.0, для забезпечення сумісності.

5. Стійкість до атак

a. Ефективність захисту від різних атак, включаючи переповнення буфера, атаки на зберігання, SQL-ін'єкції.

b. Використання заходів безпеки для запобігання використанню вразливостей.

6. Керованість та моніторинг

a. Засоби керування та моніторингу для відстеження подій та аналізу безпеки.

b. Здатність виявлення та реагування на підозрілу або аномальну активність.

7. Використання ресурсів

a. Ефективне використання ресурсів сервера та клієнта під час обробки автентифікаційних запитів.

8. Загальна легкість використання та розуміння

а. Простота використання системи для кінцевого користувача та адміністратора.

Таблиця 3.2

Зведені дані порівняння запропонованого рішення з існуючими

Характеристика	**Запропоноване рішення**	**Існуючі Рішення**
Метод автентифікації	Використання непрозорих токенів	Різноманітні методи: паролі, біометрія, тощо, які легко підпадають під атаки
Типи токенів	Непрозорі токени для збільшення стійкості та конфіденційності	Залежно від рішення: сесійні, JWT
Стійкість до атак	Використання непрозорих токенів дозволяє уникнути атак перехоплення	Залежить від реалізації, можливість атак перехоплення
Конфіденційність інформації в токенах	Забезпечення конфіденційності інформації завдяки непрозорим токенам	Залежить від типу токенів і реалізації
Швидкодія та ефективність	Висока швидкодія, оптимізована робота з непрозорими токенами	Залежить від реалізації та методів автентифікації
Рівень захисту від атак	Високий рівень захисту від атак, особливо перехоплення токенів	Залежить від реалізації та використаних методів

Продовження таблиці 3.2

Легкість використання та інтеграції	Легко інтегрується з різноманітними та ідентично працює з іншими компонентами	Залежить від інтерфейсу та підтримки інтеграції
Поліпшення порівняно з існуючими рішеннями	Використання непрозорих токенів значно поліпшує рівень кібербезпеки	Різноманітні підходи, але не всі мають непрозорі токени
Можливості розширення та підтримка	Забезпечує легкість розширення та можливість додавання нових функцій	Залежить від реалізації та роботи з плагінами
Керованість залежностями	Можливість зручного управління залежностями програмного модулю та ресурсами	Залежить від використання менеджерів залежностей
Відкритий код та спільнота розробників	Можливість залучення до розробки та підтримки спільнотою	Залежить від розробника та популярності рішення
Загальна зручність та переваги	Забезпечує високий рівень захисту та зручність використання	Різноманітні підходи з різними вагомостями

3.5. Висновки до третього розділу

1. В пункті 3.1. було описано технологічний стек і засоби реалізації.
2. В пункті 3.2. було реалізовано та детально описано реалізацію програмного модуля автентифікації користувачів з використанням непрозорих токенів.
3. В пункті 3.3. проведено тестування з використанням Burp Suit, а саме моніторинг трафіку та перевірку перехоплення і безпечність використання непрозорого токену.
4. В пункті 3.4. Проведено оцінку ефективності та складено порівняльну таблицю з власним та існуючими рішеннями.

РОЗДІЛ 4
ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Екологічна інформація

1. Значення екологічної інформації

a. Свідоме прийняття рішень

Екологічна інформація є основним елементом, який допомагає суспільству приймати інформовані та обгрунтовані рішення щодо охорони навколишнього середовища. Забезпечення громадян доступом до точних та надійних екологічних даних дозволяє їм розуміти вплив своїх дій на природу.

Споживачі, бізнес-структури та владні органи, користуючись екологічною інформацією, можуть приймати рішення, спрямовані на зменшення впливу на довкілля. Наприклад, інформація про вуглецевий слід певного товару може впливати на вибір споживачів, спонукаючи їх віддавати перевагу продуктам з меншим викидом CO2.

Також, екологічна інформація важлива для прийняття рішень владою. Доступ до даних про забруднення, використання природних ресурсів та інші аспекти стану навколишнього середовища допомагає розробляти та впроваджувати політику, спрямовану на стале використання ресурсів та збереження природи.

Загальне підвищення рівня свідомості про екологічні проблеми сприяє відповідальному ставленню до навколишнього середовища, що, в свою чергу, формує стійкість та бажання залучати себе до заходів з охорони природи.

b. Моніторинг стану довкілля

Ефективний моніторинг стану довкілля є критичним для збереження біорізноманіття та забезпечення стійкості екосистем. Екологічна інформація забезпечує можливість систематичного збору та аналізу даних про якість повітря, води, ґрунту та інших компонентів природи.

Забруднення повітря: Екологічна інформація про концентрації шкідливих речовин у повітрі дозволяє визначати джерела забруднення та впроваджувати заходи для зменшення викидів.

Забруднення води: Моніторинг якості води надає можливість виявляти забруднення річок та озер, що є важливим для забезпечення безпеки питної води та підтримки акватичних екосистем.

Якість ґрунту: Аналіз екологічної інформації про ґрунт допомагає визначати рівень забруднення та ерозії, що є ключовим для збереження родючості та вирощування сільськогосподарських культур.

Моніторинг дозволяє вчасно виявляти зміни в природному середовищі та приймати заходи для його відновлення. Також, він надає науковцям та владі необхідні дані для розробки стратегій збереження природних ресурсів та зменшення впливу антропогенних факторів на природу.

с. Залучення громади

Екологічна інформація відіграє ключову роль у залученні громадськості до процесів прийняття рішень щодо охорони довкілля. Надання широкого доступу до даних про стан природи сприяє формуванню активної та екологічно обізнаної громади.

Свідоме споживання: Інформовані споживачі можуть вибирати продукцію та послуги, які мають менший екологічний вплив. Мета - стимулювати виробників до впровадження більш екологічно чистих технологій та практик.

Участь у громадських ініціативах: Громадські організації та ініціативи активно використовують екологічну інформацію для проведення своїх заходів та кампаній. Громадськість може брати участь у програмах збереження природи, реалізації проектів відновлення екосистем, тощо.

Тиск на владу та бізнес: Інформована громадськість може впливати на прийняття рішень владою та бізнес-структурами через акції, петиції та інші форми громадського тиску. Це стимулює владу до розробки та впровадження більш ефективних екологічних політик.

Залучення громадськості в процес збереження природи є важливим елементом сталого розвитку, оскільки це сприяє формуванню відповідального ставлення до навколишнього середовища на рівні суспільства. Екологічна інформація є мостом між науковими даними та широким загалом, стимулюючи активну участь громадян у прагненні до збереження природи.

2. Інструменти збору та поширення інформації

a. Системи моніторингу

Системи моніторингу є ключовим інструментом для збору екологічної інформації в реальному часі. Вони використовуються для автоматизованого спостереження за різними параметрами довкілля та надання надійних даних для подальшого аналізу. Деякі інструменти систем моніторингу включають:

Метеорологічні станції: Збирають дані про температуру, вологість, напрям вітру та інші метеорологічні параметри, які важливі для розуміння кліматичних змін.

Автоматизовані датчики забруднення повітря: Вимірюють концентрації шкідливих речовин у повітрі, таких як SO2, NO2, PM2.5, що дозволяє визначати якість повітря та виявляти джерела забруднення.

Датчики водних параметрів: Вимірюють рівень розчинених речовин, температуру та інші характеристики води, допомагаючи визначити стан водних екосистем.

Системи GPS та супутникового зображення: Використовуються для географічного мапування та відстеження змін в природних об'єктах.

Ці системи не лише забезпечують потужний засіб збору екологічних даних, але також дозволяють вченим та владі вчасно реагувати на зміни та приймати обгрунтовані рішення для збереження навколишнього середовища.

b. Географічна інформаційна система

Географічна інформаційна система (ГІС) є важливим інструментом для збору, аналізу та візуалізації екологічної інформації на географічній мапі.

ГІС дозволяє інтегрувати та обробляти різноманітні дані, спрощуючи розуміння взаємозв'язків між різними аспектами довкілля. Основні аспекти використання ГІС в екологічному моніторингу включають:

Картографування екосистем: ГІС дозволяє створювати детальні карти екосистем, визначаючи їхню розташованість та географічні особливості.

Моделювання змін в природі: ГІС використовується для прогнозування та аналізу впливу різних факторів на екосистеми, таких як зміни використання землі, кліматичні зміни, тощо.

Аналіз розподілу забруднювачів: ГІС допомагає визначити розподіл забруднювачів у просторі, враховуючи географічні та топографічні особливості.

Моніторинг та аналіз зон ризику: ГІС дозволяє визначити та аналізувати зони ризику забруднення, що є важливим для розробки стратегій екологічної безпеки.

Використання ГІС в екологічному моніторингу робить дані більш зрозумілими та доступними для прийняття рішень, а також сприяє розробці та впровадженню більш ефективних програм збереження природи.

с. Відкриті дані

Відкриті дані стали потужним інструментом для збору та поширення екологічної інформації. Цей підхід передбачає надання громадськості, дослідникам та іншим зацікавленим сторонам вільного доступу до даних про стан навколишнього середовища. Основні аспекти використання відкритих даних включають:

Прозорість та доступність: Публікація екологічних даних відкрито дозволяє громадськості отримувати інформацію про стан природи та вплив людської діяльності на довкілля.

Громадська участь: Відкриті дані сприяють залученню громади до збору та аналізу інформації. Це може включати участь громадян у програмах моніторингу, створенні додатків для мапування екологічних проблем, тощо.

Нові можливості досліджень: Дослідники можуть використовувати відкриті дані для проведення різноманітних досліджень та аналізу екологічних тенденцій.

Створення інноваційних рішень: Розповсюдження екологічних даних через відкриті ресурси може стимулювати розробку нових технологій та інноваційних рішень для вирішення проблем збереження природи.

Використання відкритих даних в екологічному моніторингу сприяє розширенню кола зацікавлених сторін, забезпечуючи більший обсяг інформації та сприяючи більш ефективному взаємодії для збереження навколишнього середовища.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці програмного модулю для автентифікації користувачів з використанням непрозорих токенів. У ході дослідження та реалізації модулю було виявлено та вирішено ключові проблеми, пов'язані з безпекою інформаційних мереж.

У першому розділі роботи проведено аналіз поточного стану та ідентифіковано проблеми в області забезпечення безпеки інформаційних мереж. Визначено, що існуючі підходи до автентифікації часто стикаються з викликами, такими як атаки на ідентифікаційні дані та витік інформації.

Другий розділ присвячений дослідженню підходів до автентифікації на основі токенів. Проведено аналіз різних типів токенів, зокрема тотокенів, HMAC-позначених одноразових паролів, а також вивчено стандарти, такі як OAuth 2.0 та OpenID Connect. Встановлено, що використання токенів дозволяє покращити безпеку процесу автентифікації та захистити ідентифікаційні дані від атак.

У третьому розділі роботи описано розроблений програмний модуль для автентифікації користувачів з використанням непрозорих токенів. Зокрема, детально розглянуто використання непрозорих токенів для збільшення стійкості до атак та забезпечення конфіденційності інформації. Впроваджені поліпшення показали позитивний вплив на показники кібербезпеки та забезпечили високий рівень захисту автентифікаційного процесу.

У цілому, розроблений програмний модуль для автентифікації користувачів на основі непрозорих токенів виявився ефективним та покращив існуючі підходи до забезпечення безпеки інформаційних мереж. Результати дослідження та розробки можуть бути використані для підвищення рівня кібербезпеки в різних сферах застосування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools / I.Cosmina, R. Harrop, C. Schaefer, C. Ho. – New York: Apress, 2017. – 878 с. – (5).

2. Greg L. Turnquist. Learning Spring Boot 3.0: Simplify the development of production-grade applications using Java and Spring / Greg L. Turnquist. – Birmingham: Packt Publishing, 2022. – 270 с. – (3).

3. Benjamin J Evans. Optimizing Java: Practical Techniques for Improving JVM Application Performance / Benjamin J Evans. – Manchester: O'Reilly, 2018. – 420 с. – (1).

4. Using JWT with Spring Security OAuth. [Online]. – Available: https://www.baeldung.com/spring-security-oauth-jwt (in English)

5. OAuth2 for a Spring REST API – Handle the Refresh Token in Angular. [Online]. – Available: https://www.baeldung.com/spring-security-oauth2-refresh-token-angular (in English)

6. Spring Security 5 – OAuth2 Login. [Online]. – Available: https://www.baeldung.com/spring-security-5-oauth2-login (in English)

7. Spring Security and OpenID Connect. [Online]. – Available: https://www.baeldung.com/spring-security-openid-connect (in English)

8. Spring + OAuth 2.0 + OpenID Connect. [Online]. – Available: https://darutk.medium.com/spring-oauth-2-0-openid-connect-3341a6ed1f92 (in English)

9. What is network security?. [Online]. – Available: https://nordvpn.com/uk/cybersecurity/network-security/ (in English)

10. InfoSec. [Online]. – Available: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-information-security-infosec (in English)

11. Інформаційна безпека. [Online]. – Available: https://www.miyklas.com.ua/p/informatica/10-klas/informatciini-tekhnologiyi-v-suspilstvi-322205/informatciina-bezpeka-navchannia-v-interneti-321523/re-0cf3c5d6-6a11-458b-b39d-889f102e9e71 (in Ukrainian)

12. Токени аутентифікації. [Online]. – Available: https://training.qatestlab.com/blog/technical-articles/authentication-tokens/ (in Ukrainian)

13. 2FA. [Online]. – Available: https://experience.dropbox.com/uk-ua/resources/what-is-2fa (in English)

14. Що таке багатофакторна автентифікація та коли її доцільно використовувати. [Online]. – Available: https://yubikey.com.ua/shcho-take-bahatofaktorna-avtentyfikatsiia-ta-koly-dotsilno-ii-vykorystovuvaty (in Ukrainian)

15. TOTP. [Online]. – Available: https://www.twilio.com/docs/glossary/totp (in English)

16. .HOTP TOTP Differences [Online]. – Available: https://rublon.com/blog/hotp-totp-difference/ (in English)

17. OAuth2. [Online]. – Available: https://auth0.com/intro-to-iam/what-is-oauth-2 (in English)

18. OpenID Connect. [Online]. – Available: https://openid.net/developers/how-connect-works/ (in English)

19. OIDC. [Online]. – Available: https://cqr.company/ua/wiki/protocols/oidc-the-open-authentication-protocol/ (in English)

20. Bearer Auth. [Online]. – Available: https://swagger.io/docs/specification/authentication/bearer-authentication/ (in English)

21. JWT. [Online]. – Available: https://auth0.com/docs/secure/tokens/json-web-tokens (in English)

22. JWT. Плюси та мінуси . [Online]. – Available: https://highload.today/uk/plyusy-i-minusy-jwt-kratkij-obzor/ (in Ukrainian)

23. API Keys. [Online]. – Available: https://cloud.google.com/docs/authentication/api-keys (in English)

24. Дистанційна ідентифікація та верифікація клієнта. [Online]. – Available: https://bank.gov.ua/admin_uploads/article/2_pr_2020-11-20.pdf?v=4 (in Ukrainian)

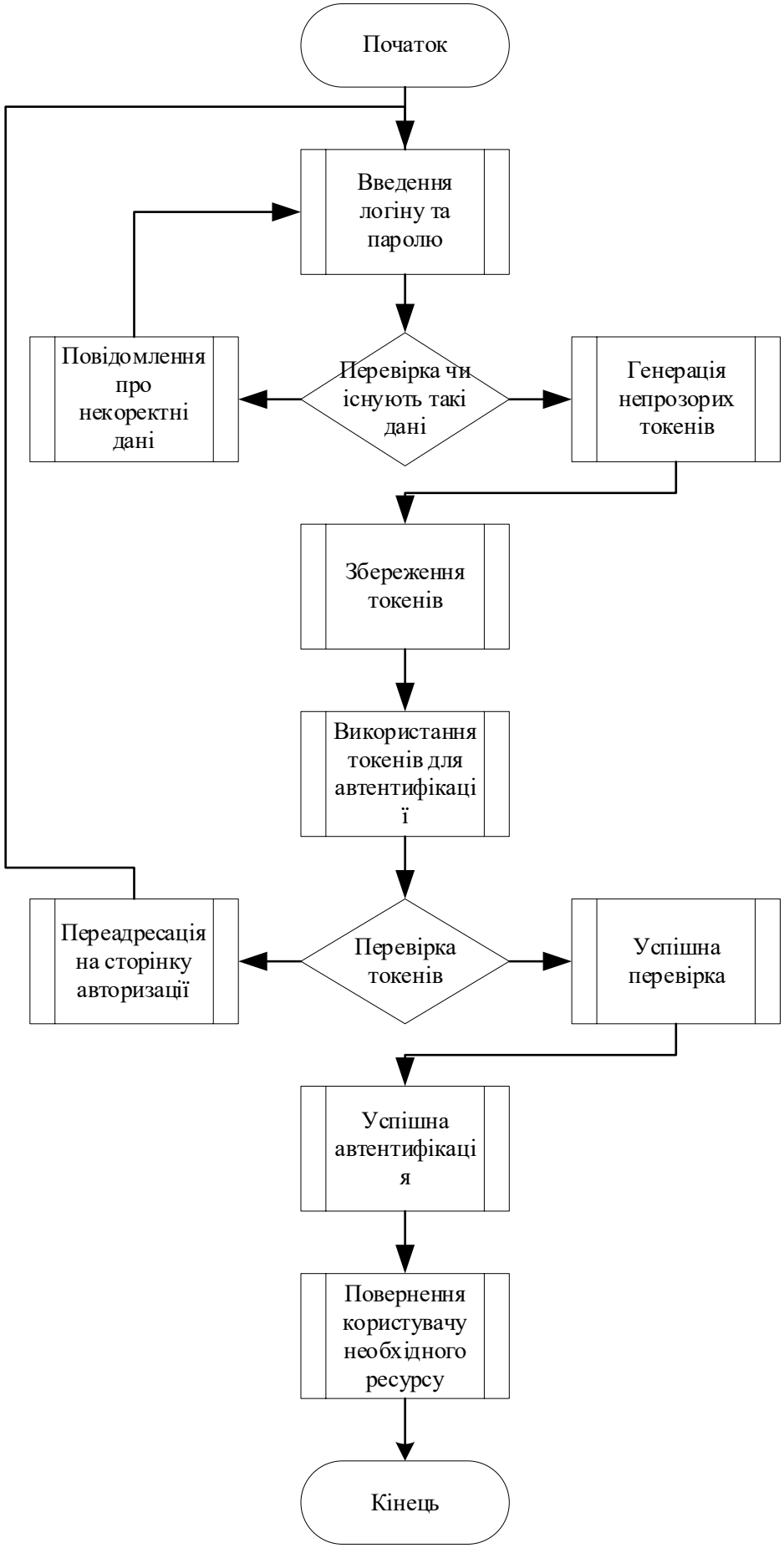
25. Підвищення рівня інформаційної безпеки за допомогою організаційних заходів накомерційних підприємствах. [Online]. – Available: https://ir.nmu.org.ua/bitstream/handle/123456789/1673/6.pdf?sequence=1 (in Ukrainian)

Додаток А. Коренева конфігурація maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.yakymchuk-eugene</groupId>
    <artifactId>spring-authorization-server-example</artifactId>
    <packaging>pom</packaging>
    <version>0.0.1</version>
    <name>auth-module-opaque-tokens</name>

    <properties>
        <java.version>17</java.version>
        <maven.compiler.target>17</maven.compiler.target>
        <maven.compiler.source>17</maven.compiler.source>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <lombok.version>1.18.26</lombok.version>
        <security-oauth2-server.version>1.1.1</security-oauth2-server.version>
        <security-oauth2-resource-server.version>3.1.1</security-oauth2-resource-server.version>
        <springdoc.version>2.1.0</springdoc.version>
        <google.guava.version>31.1-jre</google.guava.version>

        <frontend-maven-plugin.version>1.13.4</frontend-maven-plugin.version>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.1</version>
        <relativePath/>
    </parent>

    <modules>
        <module>j-sso</module>
        <module>j-service</module>
        <module>j-swagger-ui</module>
    </modules>

    <dependencyManagement>
```

```
    <dependencies>
      <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-oauth2-authorization-server</artifactId>
        <version>${security-oauth2-server.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
        <version>${security-oauth2-resource-server.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>${springdoc.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-api</artifactId>
        <version>${springdoc.version}</version>
      </dependency>
      <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>${google.guava.version}</version>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>com.github.eirslett</groupId>
          <artifactId>frontend-maven-plugin</artifactId>
          <version>${frontend-maven-plugin.version}</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

Додаток Б. Блок-схема програмного модулю



```
package yakymchuk.eugene.jsso.config.security;


import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler;
import org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHandler;
import yakymchuk.eugene.jsso.config.security.handler.CustomAuthenticationSuccessHandler;
import yakymchuk.eugene.jsso.service.CustomOAuth2UserService;
import yakymchuk.eugene.jsso.service.CustomUserDetailsService;


@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
@Configuration(proxyBeanMethods = false)
public class SecurityConfig {

    public static final String LOGIN_PAGE = "/login";
    public static final String[] PERMIT_ALL_PATTERNS = {
            LOGIN_PAGE,
            "/static/**"
    };
```

```
private final CustomOAuth2UserService customOAuth2UserService;
private final CustomUserDetailsService userDetailService;
private final PasswordEncoder passwordEncoder;
private final AuthorizationServerProperties authorizationServerProperties;

// handlers
private AuthenticationSuccessHandler oAuth2successHandler;
private AuthenticationSuccessHandler loginRequestSuccessHandler;
private AuthenticationFailureHandler failureHandler;

@Bean
public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
   SocialConfigurer socialConfigurer = new SocialConfigurer()
         .oAuth2UserService(customOAuth2UserService)
         .successHandler(oAuth2successHandler)
         .failureHandler(failureHandler)
         .formLogin(LOGIN_PAGE);
   http.apply(socialConfigurer);
   http.csrf(AbstractHttpConfigurer::disable);

   http.getSharedObject(AuthenticationManagerBuilder.class)
         .userDetailsService(userDetailService)
         .passwordEncoder(passwordEncoder);

   http.authorizeHttpRequests(authorize ->
         authorize
               .requestMatchers(PERMIT_ALL_PATTERNS).permitAll()
               .anyRequest().authenticated()
   );
   return http.formLogin(configurer -> {
      configurer.loginPage(LOGIN_PAGE)
            .loginProcessingUrl(LOGIN_PAGE)
            .successHandler(loginRequestSuccessHandler)
```

```
                .failureHandler(failureHandler);
        }).build();
    }

    @PostConstruct
    private void initializeHandlers() {
        // створюємо кастомний AuthenticationSuccessHandler для форми логіна
        this.loginRequestSuccessHandler = new CustomAuthenticationSuccessHandler(
                authorizationServerProperties.getAuthenticationSuccessUrl(),
                authorizationServerProperties.getCustomHandlerHeaderName()
        );

        // вказуємо стандартний AuthenticationSuccessHandler для OAuth2 Client
        this.oAuth2successHandler = new SimpleUrlAuthenticationSuccessHandler(
                authorizationServerProperties.getAuthenticationSuccessUrl()
        );
        this.failureHandler = new SimpleUrlAuthenticationFailureHandler();
    }
}
package yakymchuk.eugene.jsso.config.security;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import org.springframework.http.server.ServletServerHttpResponse;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.Authentication;
import org.springframework.security.oauth2.server.authorization.OAuth2Authorization;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationService;
```

```
import org.springframework.security.oauth2.server.authorization.OAuth2TokenIntrospection;

import org.springframework.security.oauth2.server.authorization.OAuth2TokenType;

import
org.springframework.security.oauth2.server.authorization.authentication.OAuth2TokenIntrospectionAuthenti
cationToken;

import
org.springframework.security.oauth2.server.authorization.config.annotation.web.configurers.OAuth2Authori
zationServerConfigurer;

import org.springframework.security.oauth2.server.authorization.settings.AuthorizationServerSettings;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint;

import org.springframework.security.web.util.matcher.RequestMatcher;

import yakymchuk.eugene.jsso.dto.AuthorizedUser;

import yakymchuk.eugene.jsso.dto.IntrospectionPrincipal;

import yakymchuk.eugene.jsso.dto.TokenInfoDto;


import java.io.IOException;


import static yakymchuk.eugene.jsso.config.security.SecurityConfig.PERMIT_ALL_PATTERNS;


@RequiredArgsConstructor

@Configuration(proxyBeanMethods = false)

public class AuthorizationServerConfig {


    private final static String principalAttributeKey = "java.security.Principal";


    private final MappingJackson2HttpMessageConverter mappingJackson2HttpMessageConverter;

    private final AuthorizationServerProperties authorizationServerProperties;

    private final OAuth2AuthorizationService oAuth2AuthorizationService;


    @Bean

    @Order(Ordered.HIGHEST_PRECEDENCE)

    public SecurityFilterChain authServerSecurityFilterChain(HttpSecurity http) throws Exception {

        OAuth2AuthorizationServerConfigurer authorizationServerConfigurer = new
OAuth2AuthorizationServerConfigurer();

        authorizationServerConfigurer.tokenIntrospectionEndpoint((config) -> {
```

```
            config.introspectionResponseHandler(this::introspectionResponse);
        });

        RequestMatcher endpointsMatcher = authorizationServerConfigurer.getEndpointsMatcher();
        http.securityMatcher(endpointsMatcher)
                .authorizeHttpRequests(authorize ->
                        authorize
                                // ендпоїнти які винесемо з під security
                                .requestMatchers(PERMIT_ALL_PATTERNS).permitAll()
                                .anyRequest().authenticated()
                )
                .csrf(csrf -> csrf.ignoringRequestMatchers(endpointsMatcher))
                .exceptionHandling(exceptions -> exceptions.authenticationEntryPoint(new
LoginUrlAuthenticationEntryPoint("/login")))
                .apply(authorizationServerConfigurer);
        return http.build();
    }

    @Bean
    public AuthorizationServerSettings authorizationServerSettings() {
        return AuthorizationServerSettings.builder()
                .issuer(authorizationServerProperties.getIssuerUrl())
                .tokenIntrospectionEndpoint(authorizationServerProperties.getIntrospectionEndpoint())
                .build();
    }

    private void introspectionResponse(HttpServletRequest request, HttpServletResponse response,
Authentication authentication) throws IOException {
        OAuth2TokenIntrospectionAuthenticationToken introspectionAuthenticationToken =
(OAuth2TokenIntrospectionAuthenticationToken) authentication;
        TokenInfoDto.TokenInfoDtoBuilder tokenInfoDtoBuilder = TokenInfoDto.builder().active(false);
// створюємо білдер об'єкта відповіді
        if (introspectionAuthenticationToken.getTokenClaims().isActive()) {                    //
якщо токен активний, то заповнюємо всі параметри інформації про токен і далі намагаємося
отримати інформацію про користувача
            OAuth2TokenIntrospection claims = introspectionAuthenticationToken.getTokenClaims();
```

```
        tokenInfoDtoBuilder.active(true)
                .sub(claims.getSubject())
                .aud(claims.getAudience())
                .nbf(claims.getNotBefore())
                .scopes(claims.getScopes())
                .iss(claims.getIssuer())
                .exp(claims.getExpiresAt())
                .iat(claims.getIssuedAt())
                .jti(claims.getId())
                .clientId(claims.getClientId())
                .tokenType(claims.getTokenType());



        String token = introspectionAuthenticationToken.getToken();                    //
отримуємо значення токена, який перевіряється

        OAuth2Authorization tokenAuth = oAuth2AuthorizationService.findByToken(token,
OAuth2TokenType.ACCESS_TOKEN);     // припускаючи, що це ACCESS TOKEN, намагаємося
отримати об'єкт OAuth2Authorization з OAuth2AuthorizationService

        if (tokenAuth != null) {

            Authentication attributeAuth = tokenAuth.getAttribute(principalAttributeKey);
// Якщо знайдено цей об'єкт OAuth2Authorization, то отримуємо з нього об'єкт Authentication
наступним чином

            if (attributeAuth != null) {

                if (attributeAuth.getPrincipal() instanceof AuthorizedUser authorizedUser) {          //
Якщо отриманий об'єкт Authentication не пустий, то перевіряємо чи є його principal екземпляром
класса AuthorizedUser

                    tokenInfoDtoBuilder.principal(IntrospectionPrincipal.build(authorizedUser));
// Створюємо IntrospectionPrincipal на його основі

                } else {                                                    // Інакше викидаємо
виняток, що інші типи principal ми не підтримуємо

                    throw new RuntimeException("Principal class = " +
attributeAuth.getPrincipal().getClass().getSimpleName() + " is not supported");

                }

            }

        }

    }
```

```
        ServletServerHttpResponse httpResponse = new ServletServerHttpResponse(response);
        mappingJackson2HttpMessageConverter.write(tokenInfoDtoBuilder.build(), null, httpResponse);
// Перетворюємо наш TokenInfoDto в json рядок і відправляємо його через ServletServerHttpResponse
    }
}
package yakymchuk.eugene.jsso;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }

}

package yakymchuk.eugene.jsso.config.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.server.authorization.InMemoryOAuth2AuthorizationService;
import org.springframework.security.oauth2.server.authorization.OAuth2AuthorizationService;

@Configuration(proxyBeanMethods = false)
public class SecurityConfigUtilities {

    @Bean
    public OAuth2AuthorizationService oAuth2AuthorizationService() {
        return new InMemoryOAuth2AuthorizationService();
```

```
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(10);
    }

}
```

```
package yakymchuk.eugene.jsso.mapper;

import lombok.experimental.UtilityClass;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import yakymchuk.eugene.jsso.dao.entity.AuthorityEntity;
import yakymchuk.eugene.jsso.dao.entity.RoleEntity;
import yakymchuk.eugene.jsso.dao.entity.UserEntity;
import yakymchuk.eugene.jsso.dto.AuthorizedUser;
import yakymchuk.eugene.jsso.type.AuthProvider;

import java.util.List;
import java.util.stream.Collectors;

@UtilityClass
public class AuthorizedUserMapper {

    public AuthorizedUser map(UserEntity entity, AuthProvider provider) {
        List<GrantedAuthority> authorities = getUserAuthorities(entity);
        return AuthorizedUser.builder(entity.getEmail(), entity.getPasswordHash(), authorities)
                .id(entity.getId())
                .firstName(entity.getFirstName())
                .lastName(entity.getLastName())
                .middleName(entity.getMiddleName())
                .birthday(entity.getBirthday())
```

```
            .avatarUrl(entity.getAvatarUrl())
            .build();
    }

    public List<GrantedAuthority> getUserAuthorities(UserEntity entity) {
        return entity.getRoles().stream()
            .filter(RoleEntity::getActive)
            .flatMap(role -> role.getAuthorities().stream())
            .filter(AuthorityEntity::getActive)
            .map(authority -> new SimpleGrantedAuthority(authority.getCode()))
            .collect(Collectors.toList());
    }
}
```

```
package yakymchuk.eugene.jsso.service;

import lombok.RequiredArgsConstructor;
import
org.springframework.security.oauth2.client.userinfo.DefaultOAuth2UserService;
import org.springframework.security.oauth2.client.userinfo.OAuth2UserRequest;
import org.springframework.security.oauth2.core.OAuth2AuthenticationException;
import org.springframework.security.oauth2.core.user.OAuth2User;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import yakymchuk.eugene.jsso.type.AuthProvider;

@Service
@RequiredArgsConstructor
public class CustomOAuth2UserService extends DefaultOAuth2UserService {

    private final UserService userService;
```

```
    @Override
    @Transactional(readOnly = true)
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
OAuth2AuthenticationException {
        OAuth2User oAuth2User = super.loadUser(userRequest);
        String clientRegId = userRequest.getClientRegistration().getRegistrationId();
        AuthProvider provider = AuthProvider.fingByName(clientRegId);
        return userService.saveAndMap(oAuth2User, provider);
    }
}
```

```
package yakymchuk.eugene.jsso.service;

import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import yakymchuk.eugene.jsso.dao.entity.UserEntity;
import yakymchuk.eugene.jsso.dao.repository.UserRepository;
import yakymchuk.eugene.jsso.mapper.AuthorizedUserMapper;

@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;
```

```
    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        UserEntity entity = userRepository.findByEmail(username);
        if (entity == null) {
            throw new UsernameNotFoundException("User with username = " +
username + " not found");
        }
        return AuthorizedUserMapper.map(entity, null);
    }
}
```

```
package yakymchuk.eugene.jsso.service;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.oauth2.core.user.OAuth2User;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import yakymchuk.eugene.jsso.dao.entity.UserEntity;
import yakymchuk.eugene.jsso.dao.repository.RoleRepository;
import yakymchuk.eugene.jsso.dao.repository.UserRepository;
import yakymchuk.eugene.jsso.dto.AuthorizedUser;
import yakymchuk.eugene.jsso.exception.AuthException;
import yakymchuk.eugene.jsso.mapper.AuthorizedUserMapper;
import yakymchuk.eugene.jsso.type.AuthErrorCode;
import yakymchuk.eugene.jsso.type.AuthProvider;
```

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;



@Service
@RequiredArgsConstructor
public class DefaultUserService implements UserService {

    private final UserRepository userRepository;
    private final RoleRepository roleRepository;



    @Override
    @Transactional
    public UserEntity save(OAuth2User userDto, AuthProvider provider) {
        return switch (provider) {
            case GITHUB -> this.saveUserFromGithab(userDto);
            case GOOGLE -> this.saveUserFromGoogle(userDto);
        };
    }



    @Override
    public AuthorizedUser saveAndMap(OAuth2User userDto, AuthProvider
provider) {
        UserEntity entity = this.save(userDto, provider);
        return AuthorizedUserMapper.map(entity, provider);
```

```
}



private UserEntity saveUserFromGithab(OAuth2User userDto) {
    String email = userDto.getAttribute("email");
    UserEntity user = this.getEntityByEmail(email);

    if (userDto.getAttribute("name") != null) {
        String[] splitted = ((String) userDto.getAttribute("name")).split(" ");
        user.setFirstName(splitted[0]);
        if (splitted.length > 1) {
            user.setLastName(splitted[1]);
        }
        if (splitted.length > 2) {
            user.setMiddleName(splitted[2]);
        }
    } else {
        user.setFirstName(userDto.getAttribute("login"));
        user.setLastName(userDto.getAttribute("login"));
    }

    if (userDto.getAttribute("avatar_url") != null) {
        user.setAvatarUrl(userDto.getAttribute("avatar_url"));
    }
    return userRepository.save(user);
}
```

```
private UserEntity saveUserFromGoogle(OAuth2User userDto) {
    String email = userDto.getAttribute("email");
    UserEntity user = this.getEntityByEmail(email);

    if (userDto.getAttribute("given_name") != null) {
        user.setFirstName(userDto.getAttribute("given_name"));
    }

    if (userDto.getAttribute("family_name") != null) {
        user.setLastName(userDto.getAttribute("family_name"));
    }

    if (userDto.getAttribute("picture") != null) {
        user.setAvatarUrl(userDto.getAttribute("picture"));
    }

    return userRepository.save(user);
}

private UserEntity getEntityByEmail(String email) {
    if (email == null) {
        throw new AuthException(AuthErrorCode.EMAIL_IS_EMPTY);
    }
    UserEntity user = this.userRepository.findByEmail(email);
    if (user == null) {
        user = new UserEntity();
        user.setEmail(email);
        user.setActive(true);
        user.setRoles(List.of(roleRepository.getDefaultRole()));
```

```
        }
        return user;
    }
}

package yakymchuk.eugene.jsso.exception;

import lombok.Getter;
import org.springframework.security.core.AuthenticationException;
import yakymchuk.eugene.jsso.type.AuthErrorCode;

@Getter
public class AuthException extends AuthenticationException {

    private final AuthErrorCode errorCode;

    public AuthException(AuthErrorCode errorCode, String msg, Throwable cause) {
        super(msg, cause);
        this.errorCode = errorCode;
    }

    public AuthException(String msg, AuthErrorCode errorCode) {
        super(msg);
        this.errorCode = errorCode;
    }

    public AuthException(AuthErrorCode errorCode) {
        super(null);
        this.errorCode = errorCode;
```

```
    }
}
```

```
package yakymchuk.eugene.jsso.dto;

import lombok.Getter;
import lombok.Setter;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.oauth2.core.user.OAuth2User;

import java.time.LocalDate;
import java.util.Collection;
import java.util.Map;
import java.util.UUID;

@Getter
@Setter
public class AuthorizedUser extends User implements OAuth2User {

    private UUID id;
    private String firstName;
    private String lastName;
    private String middleName;
    private LocalDate birthday;
    private String avatarUrl;

    private Map<String, Object> oauthAttributes;
```

```
    public AuthorizedUser(String username, String password, Collection<? extends
GrantedAuthority> authorities) {
        super(username, password, authorities);
    }

    public AuthorizedUser(
            String username,
            String password,
            boolean enabled,
            boolean accountNonExpired,
            boolean credentialsNonExpired,
            boolean accountNonLocked,
            Collection<? extends GrantedAuthority> authorities
    ) {
        super(username, password, enabled, accountNonExpired,
credentialsNonExpired, accountNonLocked, authorities);
    }

    public static AuthorizedUserBuilder builder(String username, String password,
Collection<? extends GrantedAuthority> authorities) {
        return new AuthorizedUserBuilder(username, password, authorities);
    }

    public static AuthorizedUserBuilder builder(
            String username,
            String password,
            boolean enabled,
            boolean accountNonExpired,
            boolean credentialsNonExpired,
            boolean accountNonLocked,
```

```
        Collection<? extends GrantedAuthority> authorities
    ) {
        return new AuthorizedUserBuilder(username, password, enabled,
accountNonExpired, credentialsNonExpired, accountNonLocked, authorities);
    }

    public String getEmail() {
        return this.getUsername();
    }

    @Override
    public Map<String, Object> getAttributes() {
        return oauthAttributes;
    }

    @Override
    public String getName() {
        return this.getUsername();
    }

    public static class AuthorizedUserBuilder {

        private final AuthorizedUser entity;

        AuthorizedUserBuilder(String username, String password, Collection<? extends
GrantedAuthority> authorities) {
            if (password == null) {
                password = "";
            }
```

```
        this.entity = new AuthorizedUser(username, password, authorities);
    }

    AuthorizedUserBuilder(
            String username,
            String password,
            boolean enabled,
            boolean accountNonExpired,
            boolean credentialsNonExpired,
            boolean accountNonLocked,
            Collection<? extends GrantedAuthority> authorities
    ) {
        this.entity = new AuthorizedUser(username, password, enabled,
accountNonExpired, credentialsNonExpired, accountNonLocked, authorities);
    }

    public AuthorizedUserBuilder id(UUID id) {
        this.entity.setId(id);
        return this;
    }

    public AuthorizedUserBuilder firstName(String firstName) {
        this.entity.setFirstName(firstName);
        return this;
    }

    public AuthorizedUserBuilder lastName(String lastName) {
        this.entity.setLastName(lastName);
        return this;
```

```
        }

        public AuthorizedUserBuilder middleName(String middleName) {
            this.entity.setMiddleName(middleName);
            return this;
        }

        public AuthorizedUserBuilder birthday(LocalDate birthday) {
            this.entity.setBirthday(birthday);
            return this;
        }

        public AuthorizedUserBuilder avatarUrl(String avatarUrl) {
            this.entity.setAvatarUrl(avatarUrl);
            return this;
        }

        public AuthorizedUserBuilder oauthAttributes(Map<String, Object>
userSasInfo) {
            this.entity.setOauthAttributes(userSasInfo);
            return this;
        }

        public AuthorizedUser build() {
            return this.entity;
        }
    }
}
```

```
package yakymchuk.eugene.jsso.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import org.springframework.security.core.GrantedAuthority;

import java.time.LocalDate;
import java.util.Collections;
import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;

@Getter
@Builder
@AllArgsConstructor
public class IntrospectionPrincipal {

    private UUID id;
    private String firstName;
    private String lastName;
    private String middleName;
    private LocalDate birthday;
    private String avatarUrl;
    private String username;
    private String email;
    private List<String> authorities;

    public static IntrospectionPrincipal build(AuthorizedUser authorizedUser) {
```

```
        if (authorizedUser == null) {
            return null;
        }

        List<String> authorities = Collections.emptyList();
        if (authorizedUser.getAuthorities() != null) {
            authorities = authorizedUser.getAuthorities()
                    .stream()
                    .map(GrantedAuthority::getAuthority)
                    .collect(Collectors.toList());
        }

        return IntrospectionPrincipal.builder()
                .id(authorizedUser.getId())
                .firstName(authorizedUser.getFirstName())
                .lastName(authorizedUser.getLastName())
                .middleName(authorizedUser.getMiddleName())
                .birthday(authorizedUser.getBirthday())
                .avatarUrl(authorizedUser.getAvatarUrl())
                .username(authorizedUser.getUsername())
                .email(authorizedUser.getEmail())
                .authorities(authorities)
                .build();
    }
}
```

```
package yakymchuk.eugene.jsso.dao.repository;

import lombok.RequiredArgsConstructor;
```

```
import org.springframework.security.oauth2.core.AuthorizationGrantType;
import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
import
org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import
org.springframework.security.oauth2.server.authorization.client.RegisteredClientRep
ository;
import
org.springframework.security.oauth2.server.authorization.settings.OAuth2TokenFor
mat;
import
org.springframework.security.oauth2.server.authorization.settings.TokenSettings;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import yakymchuk.eugene.jsso.dao.entity.SystemOauth2Client;

import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.stream.Collectors;

@Repository
@RequiredArgsConstructor
public class DaoRegisteredClientRepository implements RegisteredClientRepository
{
    private final SystemOauth2ClientRepository systemOauth2ClientRepository;

    @Override
    @Transactional
    public void save(RegisteredClient dto) {
        SystemOauth2Client entity = new SystemOauth2Client();
        if (dto.getId() != null) {
```

```
            entity =
systemOauth2ClientRepository.getReferenceById(Long.parseLong(dto.getId()));
        }
        this.map(dto, entity);
        systemOauth2ClientRepository.save(entity);
    }
    @Override
    @Transactional(readOnly = true)
    public RegisteredClient findById(String id) {
        SystemOauth2Client entity =
systemOauth2ClientRepository.getReferenceById(Long.parseLong(id));
        return this.map(entity);
    }
    @Override
    @Transactional(readOnly = true)
    public RegisteredClient findByClientId(String clientId) {
        SystemOauth2Client entity =
systemOauth2ClientRepository.getByClientId(clientId);
        return this.map(entity);
    }
    private RegisteredClient map(SystemOauth2Client entity) {
        return RegisteredClient.withId(entity.getId().toString())
                .clientId(entity.getClientId())
                .clientSecret(entity.getClientSecret())
                .clientIdIssuedAt(entity.getClientIdIssueAt().toInstant(ZoneOffset.UTC))

.clientSecretExpiresAt(entity.getClientSecretExpiresAt().toInstant(ZoneOffset.UTC))
                .clientName(entity.getClientName())
                .clientAuthenticationMethods(clientAuthenticationMethods ->
clientAuthenticationMethods.addAll(entity.clientAuthenticationMethods()))
```

```
            .authorizationGrantTypes(authorizationGrantTypes ->
authorizationGrantTypes.addAll(entity.authorizationGrantTypes()))
            .redirectUris(redirectUris -> redirectUris.addAll(entity.redirectUris()))
            .scopes(scopes -> scopes.addAll(entity.scopes()))

.tokenSettings(TokenSettings.builder().accessTokenFormat(OAuth2TokenFormat.RE
FERENCE).build())
            .build();
    }
    private void map(RegisteredClient dto, SystemOauth2Client entity) {
        entity.setClientId(dto.getClientId())
            .setClientIdIssueAt(dto.getClientIdIssuedAt() != null ?
LocalDateTime.ofInstant(dto.getClientIdIssuedAt(), ZoneOffset.UTC) : null)
            .setClientSecret(dto.getClientSecret())
            .setClientSecretExpiresAt(dto.getClientSecretExpiresAt() != null ?
LocalDateTime.ofInstant(dto.getClientSecretExpiresAt(), ZoneOffset.UTC) : null)
            .setClientName(dto.getClientName())

.setClientAuthenticationMethods(dto.getClientAuthenticationMethods().stream()
                .map(ClientAuthenticationMethod::getValue)
                .collect(Collectors.joining(","))
            )
            .setAuthorizationGrantTypes(dto.getAuthorizationGrantTypes().stream()
                .map(AuthorizationGrantType::getValue)
                .collect(Collectors.joining(","))
            )
            .setRedirectUris(String.join(",", dto.getRedirectUris()))
            .setScopes(String.join(",", dto.getScopes()));
    }
}
```