ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження алгоритмів аутентифікації за технологіями OAuth2 та JWT для користувача у WEB-додатках»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро КОРЧОВИЙ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Дмитро КОРЧОВИЙ
група ІСДМ-61

Керівник: Ігор СІНІЦИН
науковий ступінь, вчене звання к.т.н., професор

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем
Ступінь вищої освіти Магістр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма 126 Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою _Каміла СТОРЧАК__
Ім'я, ПРІЗВИЩЕ
«____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Корчовому Дмитру Юрійовичу_____
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження алгоритмів аутентифікації за технологіями OAuth2 та JWT для користувача у WEB-додатках
керівник кваліфікаційної роботи Сініцин І.П. КТН Професор,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. No 145

2. Строк подання кваліфікаційної роботи «28» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи:
   1. Науковотехнічна література.
   2. Аналіз систем автентифікації WEB - додатків

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Теоретичні основи аутентифікації в WEB - додатках
   2. Реалізація аутентифікації на практиці
   3. Аналіз і порівняння результатів

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково технічної літератури	20.10.23	Викон.
2	Теоретичні основи аутентифікації в WEB - додатках	31.10.23	Викон.
3	Реалізація аутентифікації на практиці	15.11.23	Викон.
4	Аналіз і порівняння результатів	29.11.23	Викон.
5	Висновки, вступ, реферат	11.12.23	Викон.
6	Розробка презентації	18.12.23	Викон.

Здобувач(ка) вищої освіти _____ Дмитро КОРЧОВИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ігор СІНІЦИН
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 67 стор., 17 рис., 30 дж., 2 додатка.

Мета роботи – вивчення та аналіз сучасних методів забезпечення безпеки, аутентифікації та авторизації веб-додатків на основі протоколу OAuth та токенів JWT.

Об'єкт дослідження – протокол OAuth та токени JWT.

Предмет дослідження – аутентифікації та авторизації веб-додатків.

Короткий зміст роботи: В магістерський роботі було проведено дослідження алгоритмів аутентифікації за технологіями JWT та OAuth2 і виявлено, що кожен підхід має свої сильні та слабкі сторони, які повинні бути враховані при розробці веб-додатків. Вибір між JWT та OAuth2 не є однозначним і залежить від специфіки та вимог проекту.

КЛЮЧОВІ СЛОВА: JSON WEB TOKENS, OAUTH, JWT, PYTHON, FLASK, TOKEN, SECURITY, ВЕБ ДОДАТОК, HTTPS, HTTP, WEB, АУТЕНТИФІКАЦІЯ, GOOGLE CLOUD, АВТОРИЗАЦІЯ, RESTFUL, CREDENTIALS, JAVA, API

Додаток
Анотація англійською мовою
для кваліфікаційної магістерської роботи

ABSTRACT

Text part of the master's qualification work: 67 pages, 17 pictures, 30 sources, 2 appendices.

The purpose of the work - and analyze modern methods of security, authentication and authorization of web applications based on the OAuth protocol and JWT tokens.

Object of research – OAuth protocol and JWT tokens.

Subject of research – authentication and authorization of web applications.

Summary of the work: In the master's thesis, a study of authentication algorithms using JWT and OAuth2 technologies was conducted and it was found that each approach has its own strengths and weaknesses that should be taken into account when developing web applications. The choice between JWT and OAuth2 is not unambiguous and depends on the specifics and requirements of the project.

KEYWORDS: JSON WEB TOKENS, OAUTH, JWT, PYTHON, FLASK, TOKEN, SECURITY, WEB APPLICATION, HTTPS, HTTP, WEB, AUTHENTICATION, GOOGLE CLOUD, AUTHORIZATION, RESTFUL, CREDENTIALS, JAVA, API

ЗМІСТ

ВСТУП

Веб-розробка є однією з найбільш активно розвиваючихся галузей інформаційної технології, і разом з цим росте і важливість забезпечення безпеки, аутентифікації та авторизації веб-додатків. З кожним днем кількість веб-злочинів та загроз для конфіденційності даних користувачів зростає, що робить це питання дедалі більш актуальним і важливим для розробників.

Однією з основних проблем є забезпечення безпеки під час обміну даними між веб-додатком та користувачем, а також впевненість в тому, що лише вповноважені користувачі мають доступ до конфіденційних ресурсів. Класичні методи аутентифікації та авторизації зачасту виявляються недостатньою захистом від сучасних загроз.

У цьому контексті, виникає необхідність вивчення та впровадження сучасних технологій, таких як OAuth та JWT, які дозволяють створювати захищені веб-додатки та забезпечувати надійну аутентифікацію та авторизацію користувачів.

Ця робота спрямована на вивчення та розуміння принципів функціонування OAuth та JWT, їх використання в розробці веб-додатків та забезпечення безпеки під час обміну даними.

Метою даного дослідження є вивчення та аналіз сучасних методів забезпечення безпеки, аутентифікації та авторизації веб-додатків на основі протоколу OAuth та токенів JWT (JSON Web Tokens). Робота спрямована на розуміння принципів їх роботи, переваг та недоліків використання в практичній веб-розробці.

Для досягнення поставленої мети були сформульовані такі основні завдання дослідження:

- Вивчення принципів роботи протоколу OAuth: Розглядання процесу аутентифікації та авторизації в системах, що використовують OAuth, з аналізом ключових понять і кроків.

- Розгляд схеми JWT: Вивчення основних концепцій JSON Web Tokens, включаючи структуру токенів та їх використання для забезпечення безпеки та аутентифікації.
- Аналіз переваг та недоліків: Оцінка переваг та обмежень використання OAuth та JWT в порівнянні з іншими методами забезпечення безпеки в веб-розробці.
- Практичне використання: Розробка прикладу веб-додатку, що використовує протокол OAuth та JWT для забезпечення безпеки та авторизації користувачів.
- Аналіз результатів: Оцінка ефективності та безпеки розробленого веб-додатку з використанням зазначених технологій.
- Формулювання рекомендацій: На основі отриманих результатів надати рекомендації щодо вибору та використання протоколу OAuth та токенів JWT у веб-розробці.

Актуальність теми дослідження полягає в тому, що в сучасному цифровому світі веб-додатки стали невід'ємною частиною життя користувачів. Збільшення обсягу онлайн-сервісів і додатків призводить до необхідності надійного забезпечення безпеки та приватності користувачів. Використання протоколу OAuth та токенів JWT стає все більш розповсюдженим способом забезпечення безпеки та авторизації.

Застосування протоколу OAuth і токенів JWT дозволяє:

- Забезпечити безпеку обміну даними між клієнтами та серверами.
- Впроваджувати потужну систему аутентифікації та авторизації без необхідності зберігання паролів користувачів на клієнтській стороні.
- Зменшити навантаження на сервер шляхом збереження даних авторизації в токенах.
- Підтримувати єдину сесію між різними серверами та додатками.

1 ТЕОРЕТИЧНІ ОСНОВИ АУТЕНТИФІКАЦІЇ В WEB-ДОДАТКАХ

1.1. Основні поняття аутентифікації

Основні поняття автентифікації в контексті інформаційної безпеки та авторизації є важливими для розуміння і дослідження технологій автентифікації, таких як OAuth2 та JWT.

Автентифікація − це термін, який означає процес доведення того, що певний факт або документ є справжнім. У комп'ютерних науках цей термін зазвичай асоціюється з підтвердженням особи користувача. Зазвичай користувач підтверджує свою особу, надаючи свої облікові дані, тобто узгоджену інформацію, якою обмінюються користувач і система. Комбінація імені користувача та пароля є найпопулярнішим механізмом автентифікації, який також відомий як парольна автентифікація.

Відомим прикладом є доступ до облікового запису користувача на веб-сайті або у постачальника послуг, наприклад, Facebook або Gmail. Перш ніж отримати доступ до свого облікового запису, користувач повинен довести, що володіє правильними обліковими даними для входу. Зазвичай сервіси представляють екран, який запитує ім'я користувача разом із паролем. Потім вони порівнюють введені користувачем дані зі значеннями, які раніше зберігалися у внутрішньому сховищі. Якщо ввести правильну комбінацію цих облікових даних, постачальник послуг дозволить продовжити і надасть доступ до облікового запису.

Авторизація − це процес надання комусь можливості доступу до ресурсу.

Хорошим прикладом є власність на будинок. Власник має повне право доступу до власності (ресурсу), але може надавати право доступу до нього іншим людям. Тобто власник уповноважує людей на доступ до нього. Цей простий приклад дозволяє нам ввести кілька понять у контексті авторизації.

Наприклад, доступ до будинку − це дозвіл, тобто дія, яку можна виконати на ресурсі. Іншими дозволами на будинок можуть бути його облаштування,

прибирання, ремонт тощо. Дозвіл стає привілеєм (або правом), коли його комусь призначають. Отже, якщо власник надає дозвіл на облаштування вашого будинку декоратору інтер'єру, він надає йому цей привілей. Іноді дозвіл дещо пов'язаний з ідентифікацією. Подумайте про процес посадки на літак. У пасажира є посадковий талон, який підтверджує, що пасажир має право летіти цим літаком. Однак цього недостатньо для того, щоб агент на виході на посадку дозволив пасажиру потрапити на борт. Пасажиру також потрібен паспорт, що посвідчує його особу. У цьому випадку агент на виході на посадку порівнює ім'я в паспорті з ім'ям у посадковому талоні і пропускає пасажира, якщо вони збігаються.

У контексті авторизації ім'я є атрибутом особи. Іншими атрибутами є ваш вік, мова, кредитна картка і все інше, що має значення в конкретному сценарії.

В описаних вище сценаріях можна побачити, що акт авторизації дозволяє суб'єктам виконувати завдання, які іншим суб'єктам не дозволено виконувати. Комп'ютерні системи, які використовують авторизацію, працюють подібним чином.

Облікові дані − це інформація, яка використовується для автентифікації, зазвичай у вигляді імені користувача та пароля. Інші форми облікових даних можуть включати біометричні дані, цифрові сертифікати або токени. Облікові дані надаються під час процесу автентифікації для підтвердження ідентичності суб'єкта.

Ідентичність представляє унікальну інформацію, пов'язану з користувачем чи суб'єктом, що дозволяє їх відрізняти від інших. В контексті автентифікації, ідентичність використовується для зв'язування облікових даних суб'єкта з їх особистою ідентичністю.

Багаторівнева автентифікація − це метод забезпечення безпеки, який вимагає від користувачів надати два або більше окремих видів автентифікації перед наданням доступу. Зазвичай це включає щось, що користувач знає (пароль), щось, що користувач має (секретний токен чи смартфон) і щось, що користувач є (біометричні дані).

Одноразовий вхід (англ. Single Sign-On) − це механізм, який дозволяє користувачу увійти один раз і отримати доступ до кількох пов'язаних систем чи

додатків без необхідності повторної автентифікації для кожного з них. SSO підвищує зручність для користувача, при цьому зберігаючи безпеку.

1.2. Технології аутентифікації в WEB-додатках

Повноцінна система веб-автентифікації важлива для впорядкованої роботи веб-сервісів. Будучи першою лінією захисту, вона є передумовою безпеки сервісів веб-додатків. По суті, автентифікація − це процес перевірки облікових даних. Мета автентифікації - ідентифікувати дозволеного користувача. Вона може гарантувати, що користувач, який використовує веб-сервіс з цифровою ідентичністю, є законним власником цієї цифрової ідентичності.

Ключовими елементами системи веб-автентифікації є облікові дані та стратегія автентифікації. При проектуванні системи автентифікації слід повністю враховувати вимоги конкретних прикладних сценаріїв. Стратегія автентифікації − це технічне рішення, яке запускається після всебічного розгляду міркувань безпеки, атрибутів облікових даних, витрат на впровадження та досвіду користувачів. Вона реалізує обмеження безпеки на облікові дані для автентифікації. Стратегія автентифікації визначає нижню межу зручності використання системи автентифікації. В умовах безперервної чорно-білої гри побудова системи веб-автентифікації стає більш досконалою, диверсифікована інформація про облікові дані, повна стратегія автентифікації та надійна мережева передача дають можливість людям отримувати більш високий рівень користувацького досвіду веб-автентифікації.

У системі веб-автентифікації облікові дані − це вид інформації, що підтверджує легітимність особи. Коли користувач, який претендує на цифрову ідентичність, звертається до веб-ресурсу, веб-автентифікація вимагає від користувача пред'явити певні облікові дані, які можуть довести легітимність його або її особи, і передає зібрані облікові дані на наступні етапи для верифікації. Розробники веб-додатків повинні враховувати ці вимоги при розробці процесу

сертифікації, вибирати правильні облікові дані і робити веб-сервіси більш безпечними і зручними.

Пароль є найбільш широко використовуваним обліковим записом у веб-автентифікації з часів Web 1.0. Хоча безпека пароля нижча, ніж у біометричних даних, цифрових сертифікатів та інших облікових даних, він має такі переваги, як простота реалізації, низька вартість і легкість розгортання. Пароль залишатиметься основним засобом веб-автентифікації протягом певного періоду часу.

Статичний пароль − це свого роду секретна інформація, що складається з символів, яка не має часових обмежень і може бути змінена і використана повторно. На сьогоднішній день більшість веб-сайтів, включаючи новостворені, все ще обирають текстовий пароль як один з основних способів автентифікації. Текстовий пароль - це рядок, який встановлюється користувачем відповідно до його звичок, запам'ятовується і зберігається користувачем. Однак людський мозок може запам'ятати лише від 5 до 7 паролів, а користувач часто використовує декілька веб-сервісів з паролями в якості облікових даних для автентифікації, в поєднанні з недостатньою увагою користувача до інформаційної безпеки в повсякденних справах, що призводить до активної генерації та використання вразливих команд з низькою інформаційною ентропією.

Зважаючи на все більш серйозні проблеми безпеки, спричинені дефектами текстових паролів, дослідники запропонували рішення з точки зору заміни, управління та вдосконалення. Графічний пароль є альтернативою текстовому паролю, він більше підходить для управління паролями декількох акаунтів, оскільки людський мозок може розпізнавати і запам'ятовувати зображення краще, ніж текст. Існує три способи встановлення графічного пароля: на основі графічного пароля малювання, репрезентативною схемою є DAS, запропонований Джерміном та іншими [11], на основі графічного пароля розташування, репрезентативною схемою є метод визначення послідовності позицій зображення, запропонований Блондером [12] на основі графічного пароля когніції, репрезентативною схемою є Passfaces. В аспекті управління паролями, широко використовуваний менеджер паролів який надає користувачам зручні та безпечні послуги генерації та зберігання

паролів, користувачеві потрібно лише запам'ятати основний пароль для входу в менеджер паролів, в той час як паролі інших веб-облікових записів можуть бути встановлені користувачами або автоматично згенеровані менеджером з високою інформаційною ентропією, і безпечно зберігати ці паролі в менеджері паролів.

Динамічний пароль, також відомий як одноразовий пароль (OTP) − це секретна інформація, що генерується терміналом користувача або сервером на основі алгоритму синхронізації подій/часу або алгоритму "запит-відповідь".

Через невизначеність факторів автентифікації, паролі, отримані в кожному процесі автентифікації, відрізняються, тому зловмисник не може отримати доступ до системи за допомогою атаки повторного відтворення.

Алгоритм HOTP (одноразовий пароль на основі HMAC) є найбільш класичним алгоритмом синхронізації подій Алгоритм OTP, псевдо-код алгоритму HOTP виглядає наступним чином (рис. 1.1):

HOTP(K, C) = Truncate(HMAC-SHA-1(K,C))

У початковому стані користувацький термінал уніфікує значення лічильника і розмір кроку з сервером, а потім кожного разу, коли обчислюється HOTP, значення лічильника з обох боків додають по одному кроку. K і C використовуються разом як початкові дані при генерації HOTP. Користувацький термінал спочатку отримує 20-байтовий рядок за допомогою алгоритму HMAC-SHA-1, а потім перетворює рядок, згенерований алгоритмом HMAC-SHA-1, в число десяткових цифр. HOTP, відправляє його на сервер, після чого лічильник подій збільшується на один крок. Після отримання HOTP, сервер генерує HOTP в тому ж процесі і виконує перевірку порівняння.

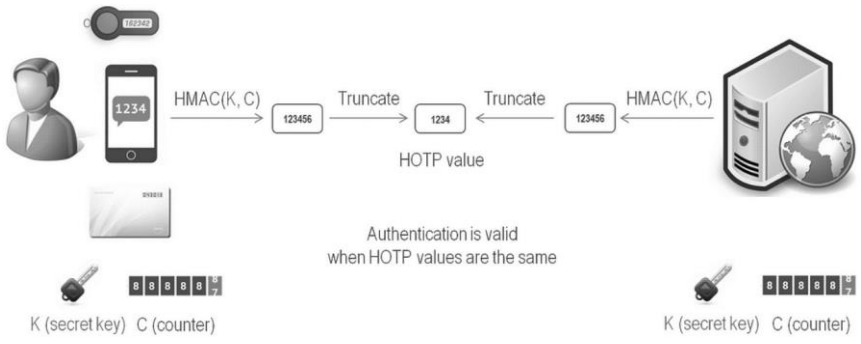


Рис. 1.1 Алгоритм HOTP

Алгоритм TOTP (time-based one-time password) є відкритим стандартом специфікації алгоритму ITEF RFC 6238. Оскільки структура алгоритму TOTP співпадає з HOTP, в цій роботі не буде детально описувати процес генерації TOTP.

На основі асинхронного динамічного пароля механізму "запит-відповідь" сторони, що спілкуються, заздалегідь домовляються про алгоритм шифрування та ключ. Коли користувач отримує доступ до системи, автентифікатор сервера випадковим чином генерує номер виклику та надсилає його користувачеві. Користувач вводить номер виклику на власному терміналі, після чого термінал обчислює номер відповіді за допомогою алгоритму шифрування відповідно до номера виклику та ключа, а потім надсилає його до системи для автентифікації. Коли система отримує відповідь, вона виконує ті ж самі дії, що й термінал користувача, щоб отримати результати обчислень, і порівнює їх. Якщо вони збігаються, то система пройде автентифікацію, в іншому випадку, ні.

Біологічні характеристики в основному включають фізіологічні та поведінкові характеристики. Фізіологічні характеристики включають відбитки голосу, пальців, райдужної оболонки ока, обличчя та ДНК. Поведінкові характеристики включають позу під час руху, рукописний підпис тощо. Завдяки своїм унікальним і стабільним характеристикам, біометричні дані важко підробити, одним словом, біометрія є чудовим сертифікатом автентифікації. Система веб-автентифікації, що використовує технологію біометричної автентифікації, в основному використовує відбитки пальців і обличчя в якості облікових даних.

Відбиток пальця − це весь або частина гребеня тертя пальця, виступаюча частина шкіри пальця, яка є унікальним вродженим посвідченням особи. Особливості відбитків пальців в основному включають глобальні та локальні особливості. Глобальні ознаки включають основні точки і точки трикутника. Локальні ознаки включають кінцеві точки і точки біфуркації дактилоскопічного гребеня. Виділення особливостей відбитків пальців − це процес вилучення місцезнаходження, типу, напрямку та іншої інформації про ці особливості та збереження їх у файлах особливостей.

Обличчя людини має велику кількість біометричної інформації. Риси обличчя людини, такі як брови, очі, вуха, ніс і рот, впливають на риси обличчя. Інформація про їх розмір, форму та розташування може бути використана як основа для розпізнавання обличчя. Існує два основних напрямки досліджень технології розпізнавання обличчя: вимірювання обличчя та власне обличчя. Ідея технології вимірювання обличчя полягає у визначенні положення очей, носа і рота та відстані між цими рисами, а потім створення шаблону обличчя після нормалізації. Метод власного обличчя заснований на ідеї власного вектора витягнутого зображення обличчя в наборі даних, який представляє власне обличчя через власний вектор, а обличчя є лінійною комбінацією власних облич, тому обличчя може бути перевірено за власним обличчям з максимальним власним значенням. Теоретично, біометрія буде найнадійнішим методом автентифікації за умови безпечної та надійної мережевої передачі та зберігання інформації.

Паролі та біометричні дані зазвичай використовуються для операцій входу в систему. Коли користувачі успішно ввійшли в систему і відвідують інші сторінки веб-сайту, якщо вони щоразу вводять облікові дані вручну, користувацький досвід буде незадовільним, а повторне введення збільшить ризик крадіжки облікових даних користувача. Однак, з точки зору безпеки, при відвідуванні інших сторінок веб-сайту необхідно вводити ідентифікаційні дані користувача. Для того, щоб вирішити цю проблему, при успішному вході користувача в систему, "ім'я користувача / пароль" буде замінено на прозорий для користувача обліковий запис.

Після успішного входу користувача веб-сервер генерує нову сесію. Сеанс можна розуміти як список, в якому записується статус користувача, його особистість та інша інформація у вигляді сеансових змінних. Кожна сесія має свій унікальний ідентифікатор сесії, який є єдиною інформацією про сесію, відомою користувачеві. Браузер автоматично надсилає ідентифікатор сесії на сервер щоразу, коли користувач відкриває нову сторінку. Сервер запитує сесію відповідно до ідентифікатора сесії, і якщо термін дії сесії не закінчився, вона пройде аутентифікацію.

З великою кількістю додатків розподілених серверів, через обмеження доменів cookie, сервер не може ділитися сесією з іншими серверами, що призводить до того, що стан сесії користувача не підтримується належним чином. Веб-токен може вирішити ці проблеми.

Коли користувач реєструється, сервер веб-автентифікації кодує ідентифікаційну інформацію користувача, щоб згенерувати рядок, і використовує його як маркер, коли клієнт запитує дані. Після цього користувач може пройти автентифікацію, лише пред'явивши токен при вході в систему. Інформація, що зберігається в токені, включає в себе ідентифікатор користувача, таблицю дозволів і т.д. Для того, щоб гарантувати, що інформація не буде підроблена зловмисниками, сервер зашифрує її, згенерує підпис, прикріпить підпис до токена і поверне його клієнту. Коли клієнт отримує токен, він зберігає його в локальному файлі. Коли користувач запитує веб-ресурс, поки легальний токен передається на сервер, автентифікація може бути завершена, тому йому не потрібно вводити пароль та інші облікові дані. Токен є видимим для користувачів та осіб без громадянства, сервер не зберігає та не записує токен, лише інформація та підпис у токені потребують перевірки під час автентифікації.

Веб-токен JSON (JWT) є основним засобом автентифікації в Інтернеті на основі токенів. У 2015 році IETF створила стандарт JSON-токенів (JWT) [14]. Веб-токен JSON − це компактний і самодостатній формат представлення декларації. Декларація кодується як JSON-об'єкт − корисне навантаження JSON Web Signature (JWS) або відкритий текст JSON Web encryption (JWE). JWT складається з трьох частин: заголовка, корисного навантаження і підпису. Кожна частина відокремлюється крапкою, заголовком, корисним навантаженням та підписом. JWT має багато переваг, він може підтримувати різні мови, має просту структуру і займає менше байт, тому дуже добре підходить для мережевої передачі; йому не потрібно зберігати сесію на сервері, тому його легко розширювати і він підходить для розподіленого середовища.

Цифровий сертифікат − це декларація електронного цифрового підпису, видана центром сертифікації ключів, яка використовується для автентифікації

особи користувачів в Інтернеті з метою забезпечення безпеки цифрової інформації. Цифровий сертифікат повинен містити: номер версії, серійний номер, алгоритм підпису, назву ЦСК, термін дії, інформацію про суб'єкта, відкритий ключ суб'єкта, алгоритм відкритого ключа суб'єкта, цифровий підпис ЦСК. Процес отримання цифрового сертифіката полягає в тому, що користувач спочатку подає власну ідентифікаційну інформацію до реєстраційного органу (РА), РА перевіряє кваліфікацію заявки користувача, а потім подає інформацію про заявку до центру сертифікації (ЦС); після отримання заявки та повторної перевірки справжньої ідентичності користувача, ЦС бере відкритий ключ, ідентифікаційну інформацію, термін дії сертифіката та іншу інформацію користувача як вихідне повідомлення для створення дайджесту повідомлення, потім шифрує дайджест повідомлення закритим ключем ЦС для створення цифрового підпису ЦС та видає цифровий сертифікат користувачеві.

Відповідна схема відкликання необхідна для знищення сертифіката та інформування всіх сторін, коли цифровий сертифікат більше не є дійсним через закінчення терміну дії або з інших причин. CRL є першою широко використовуваною схемою відкликання, яка підтримує список скасованих сертифікатів з відміткою часу та цифровим підписом. Revcast забезпечує своєчасну доставку списку відкликаних сертифікатів через канал FM-мовлення, що має переваги над схемами розповсюдження CRL з використанням мереж TCP/IP з точки зору своєчасності, вартості та захисту приватності. WebDAV визначає інформацію про відкликання цифрового сертифіката як окремий веб-ресурс, який зберігається з використанням архітектури REST, а сторона, що довіряє, використовує HTTP для доступу до статусу відкликання, щоб сертифікат можна було негайно відкликати.

Стратегія автентифікації становить основну структуру системи веб-автентифікації, і досконала стратегія автентифікації повинна точно ідентифікувати особу користувача, допускати легальних користувачів і виключати нелегальних. При розробці політики автентифікації слід виходити з безпеки як основної відправної точки і всебічно враховувати сценарії застосування веб-сервісів,

мережеве середовище, в якому вони знаходяться, вартість впровадження і досвід користувачів.

HTTP − це протокол без стану, в якому статус користувача, що успішно пройшов автентифікацію, не може зберігатися на рівні протоколу. Тому для додавання невизначених функцій управління станом в протокол HTTP необхідний сеанс управління файлами cookie. В даний час більшість веб-сайтів використовують сеансову аутентифікацію на основі файлів cookie (рис. 1.2).

Рис. 1.2 Аутентифікація на основі файлів cookie

Процес автентифікації на основі сесійних файлів cookie (рис. 1.2) виглядає наступним чином:

- Крок 1: Інтерфейсна форма отримує облікові дані користувача / пароля, а потім передає їх на сервер за допомогою повідомлення-запиту методом GET або POST через протокол HTTP;
- Крок 2: Сервер перевіряє ім'я користувача / пароль. Якщо він пройшов перевірку, він генерує сесію для запису статусу аутентифікації, а потім записує ідентифікатор сесії, що відповідає сесії, в поле Set-Cookie в заголовку пакета-відповіді і повертає його клієнту;
- Крок 3: Після отримання ідентифікатора сесії, надісланого сервером, клієнт зберігає його у вигляді Cookie локально. При наступному запиті на сервер браузер автоматично надсилає файл cookie з ідентифікатором

сеансу, а сервер розпізнає особу користувача, перевіряючи ідентифікатор сеансу.

Користувач вводить ім'я користувача та пароль, а сервер створює SQL-запит на основі введених імені користувача та пароля і підключається до бази даних для виконання запиту на основі SQL. Однак, деякі зловмисники вводять спеціальний SQL у форму, щоб дозволити серверу виконати автентифікацію. В цей час, якщо сервер не реалізує принцип "розділення даних і коду", зловмисники можуть успішно виконати SQL-ін'єкцію. Існує декілька способів належного захисту від атак SQL-ін'єкцій. Найкращим способом є використання попередньо скомпільованих операторів, зв'язування змінних безпосередньо, семантика SQL не зміниться, а оператори, вставлені зловмисниками, будуть запитуватися лише як імена користувачів або паролі; використання безпечних збережених процедур, ми можемо заздалегідь визначити набір збережених процедур в базі даних, які уникають використання динамічних операторів SQL або використовують функції суворої фільтрації та кодування вхідних даних для обробки вхідних даних користувача, наскільки це можливо; перевірка та обмеження типу вхідних даних може ефективно боротися з SQL-ін'єкціями; використання функцій кодування та фільтрації дозволить екранувати або видаляти деякі спеціальні символи; нарешті, дозволи, надані веб-додатку, повинні бути мінімізовані в звичайних випадках використання.

Життєвий цикл сеансу встановлюється сервером. Він починається з виклику серверною програмою оператора створення сесії. Сесія є дійсною протягом життєвого циклу. Після закінчення життєвого циклу сервер знищує сеанс. Зараз деякі веб-сервіси призначені для кращих користувачів. Досвід, якщо користувач все ще активний то наприкінці життєвого циклу сесії (виконує такі операції, як запит або відправлення), життєвий цикл сесії автоматично продовжується. Крім закінчення терміну дії сеансу, його також можна штучно завершити, наприклад, натиснувши кнопку виходу з системи. Тоді серверна програма викличе інструкцію destroy, щоб завершити сеанс.

Інфраструктура відкритих ключів (англ. Public Key Infrastructure, PKI) − це сукупність апаратного та програмного забезпечення, персоналу, політик і процедур,

що використовуються для реалізації генерації, управління, зберігання та розповсюдження ключів і сертифікатів на основі криптосистеми з відкритим ключем і функцією відкликання. Основна структура ІВК складається з центру сертифікації (ЦС), реєстраційного центру (РЦ), центру управління політикою (ЦУП), центру управління сертифікатами (ЦУС), сховища цифрових сертифікатів та резервного копіювання ключів, що включає в себе систему відновлення, і на сьогоднішній день вважається найповнішою системою автентифікації.

Розглянемо приклад, який ілюструє процес веб-автентифікації в системі PKI. Аліса − користувач, Боб − постачальник веб-послуг, а Єва − слухач. Аліса хоче отримати доступ до веб-сайту Боба через браузер, щоб скористатися його послугами, тому для Аліси, коли вона відвідує веб-сайт Боба, вона повинна підтвердити, що веб-сайт Боба є справжнім веб-сайтом, а не фішинговим веб-сайтом. У той же час, Боб повинен підтвердити особу законного користувача Аліси, щоб надавати подальші послуги. Боб заздалегідь отримав цифровий сертифікат, виданий центром сертифікації. Коли Боб отримує запит на доступ від Аліси, він надсилає цифровий сертифікат у відповідь. Аліса отримує цифровий сертифікат Боба і використовує авторитетний відкритий ключ ЦСК, попередньо встановлений у браузері, щоб розшифрувати цифровий підпис ЦСК у цифровому сертифікаті Боба. У той же час, відкритий ключ та ідентифікаційна інформація в цифровому сертифікаті Боба (наприклад, доменне ім'я) виконують хешування, щоб отримати дайджест повідомлення і порівняти його з розшифрованим цифровим підписом ЦСК. Якщо вони збігаються, це означає, що повідомлення легальне, в іншому випадку доступ припиняється. У цьому процесі, якщо Єва видає себе за Боба, перехоплює цифровий сертифікат Боба і позичає його, щоб відправити свій цифровий сертифікат на ім'я Аліси, оскільки сертифікат Єви не виданий авторитетним ЦС, відкритий ключ ЦС не може розшифрувати цифровий сертифікат Єви, браузер попередить Алісу про те, що сертифікат не виданий довіреним органом, попередить Алісу і припинить зв'язок; Крім того, навіть якщо Єва вкраде цифровий підпис ЦС у сертифікаті Боба, а потім додасть його до свого фальшивого сертифікату, через різницю між відкритим ключем Єви та доменним ім'ям Боба,

дайджест повідомлень не буде узгодженим, тому буде визначено, що Єва є нелегальною особою. На цьому кроки Аліси по перевірці автентичності Боба завершено. Після цього Боб може попросити Алісу підтвердити свою особу аналогічними методами. Ця операція практично не відрізняється від вищеописаних кроків, тому вона не буде повторюватись.

Єдиний вхід (Single sign-on, SSO) − це схема автентифікації в кластері мультисистемних сервісів. Вона дозволяє користувачам отримувати доступ до систем, відкритих для цього користувача у всіх кластерах, за допомогою єдиного входу в систему під кластером. З розвитком і популярністю розподіленої сфери все більше постачальників веб-сервісів надають послуги користувачам через розподілені системи. SSO встановлює механізм автентифікації та довіри на передумові збереження незалежності кожного додатка у веб-сервісах. Він використовує довірені файли cookie, токен-сертифікати та інші облікові дані для передачі ідентифікаційної інформації, щоб користувачі могли отримати доступ до всіх довірених веб-додатків після входу в систему.

Основним стандартним протоколом для Інтернету є OpenID, який є орієнтованою на користувача системою ідентифікації, що унікально ідентифікує користувачів за допомогою уніфікованої ідентичності (рис. 1.3).

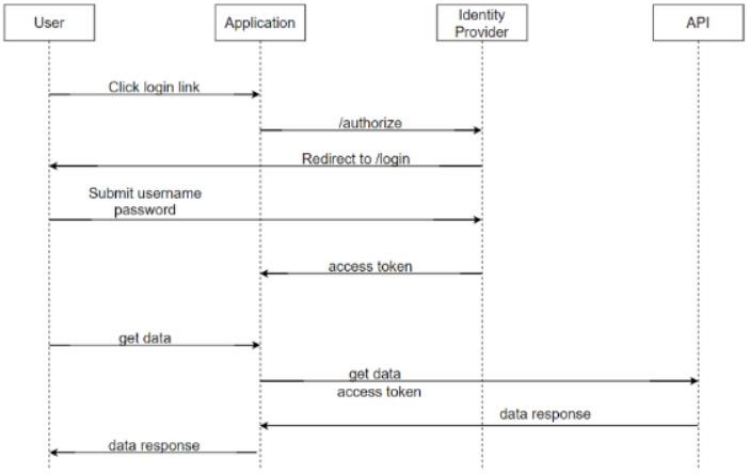


Рис. 1.3 Аутентифікація на основі файлів OpenID

Користувачі (ЄС) можуть довіряти OpenID. На веб-сайті сервісу (ВПО постачальника послуг OpenID) подається заявка на ексклюзивну реєстрацію OpenID. Після завершення реєстрації користувач формує зв'язок "один до одного" зі своїм унікальним URI. Коли користувач входить на веб-сайт веб-додатку (сторона, що покладається на OpenID, RP), йому потрібно лише ввести свій OpenID, веб-сайт веб-додатку автоматично виявить постачальника OpenID на основі OpenID і перенаправить користувача на веб-сайт постачальника OpenID для автентифікації. Після проходження аутентифікації провайдер генерує маркер (наприклад, JWT) і перенаправляє його назад на веб-сайт веб-додатку. OpenID Connect 1.0 встановлює рівень ідентичності на основі OAuth 2.0, який дозволяє клієнтам перевіряти ідентичність кінцевих користувачів на основі автентифікації, виконаної сервером авторизації, і отримувати основну профільну інформацію про користувачів в інтероперабельний і REST-подібний спосіб (рис. 1.3). З моменту створення стандарту OIDC 1.0 у листопаді 2014 року він швидко розвивається. Google, Amazon, PayPal, Oracle, Microsoft, Symantec, IBM та інші компанії використовували OIDC 1.0 для розгортання своїх систем автентифікації та авторизації.

У 2012 році було створено альянс FIDO (Fast Identity Online), метою якого є створення набору відкритих стандартних протоколів надійної автентифікації для видалення паролів, зміна ситуації, коли більшість сучасних стратегій автентифікації все ще використовують паролі в якості облікових даних, а також усунення або зменшення залежності користувачів від паролів. У грудні 2014 року альянс FIDO випустив протокол UAF і протокол U2F, що ознаменувало офіційну появу стратегії автентифікації ідентичності FIDO 1.0. У вересні 2015 року була запропонована концепція FIDO 2.0. Веб-аплікатор використовувався службами веб-додатків для доступу до надійних парольних облікових даних, сумісних з FIDO 2.0. У березні 2019 року робоча група з веб-автентифікації W3C та альянс FIDO спільно опублікували офіційний рекомендаційний стандарт веб-автентифікації: API для доступу до облікових даних з відкритим ключем рівня 1, або WebAuthn. Це означає, що FIDO2.0 офіційно введено в експлуатацію.

Стратегія автентифікації ідентичності FIDO застосовує важливу технічну концепцію: довірчі обчислення, тобто широке використання довірених обчислювальних платформ, заснованих на апаратній ізоляції і криптографічних алгоритмах, в системах автентифікації для підвищення загальної безпеки системи. Використовуючи довірені обчислення, FIDO і WebAuthn відокремлюють захищений апаратний модуль на кінцевій стороні, встановлюють якір безпеки для зберігання облікових даних і моніторингу їх потоку, а також створюють наскрізний протокол безпеки, який базується на технології криптографії з відкритим ключем.

FIDO 2.0 базується на технічному фундаменті FIDO 1.0. Два протоколи, запропоновані WebAuthn API і CTAP (Client To Authenticator Protocol), роблять FIDO придатним для більшої кількості сценаріїв, і технічні деталі не розглядаються в цій статті. Наразі веб-браузери, такі як Windows 10, Android, Chrome, Firefox, Edge і Safari, вже підтримують стандарт WebAuthn. Користувачі можуть використовувати вбудований в термінальний пристрій автентифікатор для виклику служби FIDO через інтерфейс WebAuthn, щоб досягти надійної автентифікації особи для веб-додатків. Ми вважаємо, що WebAuthn матиме широкі перспективи застосування в Інтернеті.

1.3. OAuth2: принципи та структура

OAuth 2.0, що розшифровується як "Відкрита авторизація" − це стандарт, розроблений для того, щоб дозволити веб-сайту або додатку отримувати доступ до ресурсів, розміщених на інших веб-додатках, від імені користувача. Він прийшов на зміну OAuth 1.0 у 2012 році і зараз є де-факто галузевим стандартом для онлайн-авторизації. OAuth 2.0 забезпечує узгоджений доступ і обмежує дії, які клієнтська програма може виконувати на ресурсах від імені користувача, ніколи не передаючи облікові дані користувача.

Хоча Інтернет є основною платформою для OAuth 2, специфікація також описує, як обробляти такий вид делегованого доступу для інших типів клієнтів

(браузерні додатки, веб-додатки на стороні сервера, нативні/мобільні додатки, підключені пристрої і т.д.).

OAuth 2.0 − це протокол авторизації, а НЕ протокол автентифікації. Як такий, він розроблений в першу чергу як засіб надання доступу до набору ресурсів, наприклад, віддалених API або користувацьких даних.

OAuth 2.0 використовує маркери доступу. Токен доступу − це фрагмент даних, який представляє дозвіл на доступ до ресурсів від імені кінцевого користувача. OAuth 2.0 не визначає конкретного формату для маркерів доступу. Однак у деяких контекстах часто використовується формат JSON Web Token (JWT). Це дозволяє емітентам токенів включати дані в сам токен. Крім того, з міркувань безпеки, токени доступу можуть мати дату закінчення терміну дії.

Ідея ролей є частиною основної специфікації фреймворку авторизації OAuth2.0. Вони визначають основні компоненти системи OAuth 2.0 і є наступними:

- Власник ресурсу: користувач або система, яка володіє захищеними ресурсами і може надавати доступ до них.
- Клієнт: Клієнт − це система, якій потрібен доступ до захищених ресурсів. Щоб отримати доступ до ресурсів, клієнт повинен мати відповідний маркер доступу.
- Сервер авторизації: Цей сервер отримує запити від клієнта на маркери доступу і видає їх після успішної автентифікації та згоди власника ресурсу. Сервер авторизації має дві кінцеві точки: кінцеву точку авторизації, яка обробляє інтерактивну автентифікацію та згоду користувача, і кінцеву точку токенів, яка бере участь у взаємодії між машинами.
- Сервер ресурсів: Сервер, який захищає ресурси користувача і отримує запити на доступ від клієнта. Він приймає і перевіряє маркер доступу від клієнта і повертає йому відповідні ресурси.

Області видимості є важливим поняттям в OAuth 2.0. Вони використовуються для точного визначення причини, з якої може бути надано доступ до ресурсів.

Допустимі значення діапазонів і ресурси, до яких вони відносяться, залежать від сервера ресурсів.

Сервер авторизації OAuth 2 може не повертати маркер доступу безпосередньо після того, як власник ресурсу авторизував доступ. Замість цього, для кращої безпеки, може повертатися код авторизації, який потім обмінюється на маркер доступу. Крім того, сервер авторизації може також видати маркер поновлення разом з маркером доступу. На відміну від токенів доступу, токени поновлення зазвичай мають тривалий термін дії і можуть бути замінені на нові токени доступу, коли термін дії останніх закінчується. Оскільки токени поновлення мають такі властивості, клієнти повинні зберігати їх у безпеці.

На найпростішому рівні, перш ніж OAuth 2.0 може бути використаний, клієнт повинен отримати свої власні облікові дані, ідентифікатор клієнта і секрет клієнта, від сервера авторизації, щоб ідентифікувати та автентифікувати себе при запиті токену доступу (рис. 1.4).

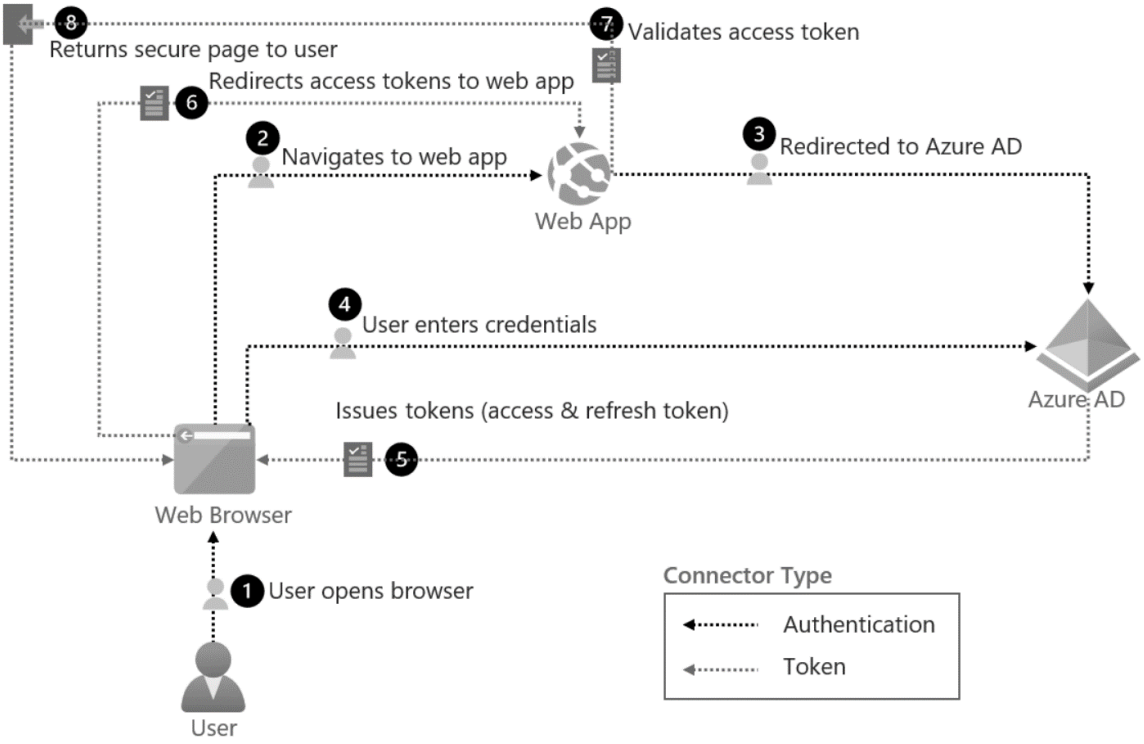


Рис. 1.4 Аутентифікація на основі файлів OpenID

Використовуючи OAuth 2.0, запити на доступ ініціюються клієнтом, наприклад, мобільним додатком, веб-сайтом, додатком для смарт-телебачення, десктопним додатком тощо. Запит токену, обмін та відповідь відбуваються за такою схемою:

Клієнт запитує авторизацію (запит на авторизацію) у сервера авторизації, надаючи ідентифікатор клієнта та секретний ключ для ідентифікації; він також надає діапазон та URI кінцевої точки (URI перенаправлення) для надсилання маркеру доступу або коду авторизації.

Сервер авторизації автентифікує клієнта і перевіряє, чи дозволені запитувані області доступу.

Власник Ресурсу взаємодіє з сервером авторизації, щоб надати доступ.

Сервер авторизації перенаправляє назад клієнту або код авторизації, або маркер доступу, залежно від типу доступу, як буде пояснено в наступному розділі. Також може бути повернутий маркер поновлення. За допомогою маркера доступу клієнт запитує доступ до ресурсу у сервера ресурсів.

В OAuth 2.0 гранти − це набір кроків, які повинен виконати клієнт, щоб отримати дозвіл на доступ до ресурсу. Система авторизації надає кілька типів грантів для різних сценаріїв:

Грант на основі коду авторизації: Сервер авторизації повертає клієнту одноразовий код авторизації, який потім обмінюється на маркер доступу. Це найкращий варіант для традиційних веб-додатків, де обмін може безпечно відбуватися на стороні сервера. Потік коду авторизації може використовуватися в односторінкових додатках (SPA) і мобільних/нативних додатках. Однак у цьому випадку секрет клієнта не може бути надійно збережений, і тому автентифікація під час обміну обмежується використанням лише ідентифікатора клієнта. Кращою альтернативою є код авторизації з PKCE-грантом, описаний нижче.

Неявний грант це спрощений потік, в якому токен доступу повертається безпосередньо клієнту. У неявному потоці сервер авторизації може повернути токен доступу як параметр в URI зворотного виклику або як відповідь на повідомлення у формі. Перший варіант зараз застарілий через потенційний витік токенів.

Надання коду авторизації з ключем підтвердження для обміну кодами (PKCE): Цей процес авторизації схожий на надання коду авторизації, але з додатковими кроками, які роблять його більш безпечним для мобільних/нативних додатків і SPA.

Тип гранту "Облікові дані власника ресурсу" вимагає, щоб клієнт спочатку отримав облікові дані власника ресурсу, які передаються на сервер авторизації. Таким чином, він обмежується клієнтами, яким повністю довіряють. Його перевага полягає в тому, що не відбувається перенаправлення на сервер авторизації, тому він застосовується в тих випадках, коли перенаправлення неможливе.

Тип надання облікових даних клієнта використовується для неінтерактивних додатків, наприклад, автоматизованих процесів, мікросервісів тощо. У цьому випадку програма автентифікується сама по собі за допомогою ідентифікатора клієнта та секрету.

Потік авторизації пристрою це грант, який дозволяє використовувати додатки на пристроях з обмеженим введенням даних, таких як смарт-телевізори.

Грант на оновлення токенів це потік, який передбачає обмін токена оновлення на новий токен доступу.

1.4. JWT: структура та переваги

JSON Web Token (JWT) представляє собою стандарт, що базується на відкритій специфікації RFC 7519. Цей високофункціональний і водночас надзвичайно компактний інструмент визначає ефективний і надійний спосіб обміну інформацією між різними системами та додатками у форматі JSON-об'єкта. Що особливо важливо, ця передана інформація може бути перевірена і надійно аутентифікована завдяки наявності цифрового підпису.

Серед ключових переваг JWT варто відзначити його здатність до підписування за допомогою секретного ключа, використовуючи різні алгоритми, такі як HMAC (зашифрування з використанням алгоритму HMAC), або ж за

допомогою пари відкритого/закритого ключа, використовуючи алгоритми RSA або ECDSA.

Цей функціональний інструмент може бути використаний у багатьох сферах, де потрібна безпека і передача даних від однієї сторони до іншої, і він надає впевненість у тому, що передані дані залишаються цілими і безпечними, забезпечуючи довіру до цієї інформації.

JWT є універсальним інструментом, який може бути використаний для безпечного обміну даними між різними системами та додатками у форматі JSON. Однією з його ключових можливостей є здатність до зашифрування, що забезпечує повну конфіденційність даних між сторонами. Однак у цьому тексті ми зосередимося на підписаних токенах, які, хоча не надають шифрування, забезпечують важливий рівень захисту та автентифікації (рис. 1.5).

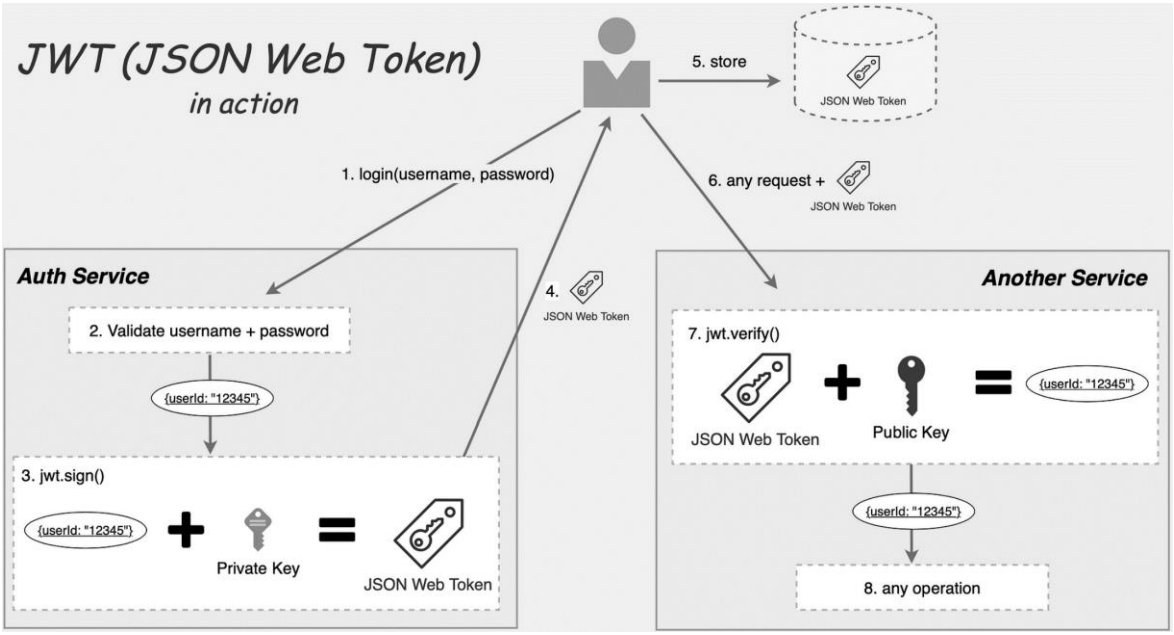


Рис. 1.5 Аутентифікація на основі файлів OpenID

Підписані токени є особливо важливими, оскільки вони дозволяють перевіряти цілісність тверджень, що містяться у токені. Це означає, що отримувач може переконатися, що інформація в токені не була змінена після підпису. Коли токени підписуються за допомогою пар відкритих/закритих ключів, підпис також виконує функцію засвідчення автентичності сторони, яка підписала токен. Це

означає, що тільки особа чи система, яка володіє закритим ключем, може бути відповідальною за підпис токена.

Розглянемо докладніше кілька сценаріїв, в яких веб-токени JWT можуть виявитися надзвичайно корисними та прогресивними засобами для різноманітних використань:

- Авторизація**:** Один із найпоширеніших та важливих сценаріїв, де JWT приходить на допомогу, це авторизація користувачів. Після успішного входу в систему кожний наступний запит включатиме JWT, надаючи користувачеві право доступу до маршрутів, сервісів та ресурсів, які визнано дозволеними для даного токену. Цей метод авторизації досить популярний через свою ефективність та здатність працювати без проблем в різних доменах завдяки низьким накладним витратам.
- Обмін інформацією**:** Використання JWT стає необхідністю у випадках, коли потрібно безпечно обмінюватися даними між сторонами. Діяльність також підтверджується цими токенами, оскільки JWT можна підписувати, використовуючи, наприклад, пари відкритих/закритих ключів. Це дозволяє впевнитися, що відправники інформації є тими, за кого себе видають, і що дані не піддалися підробці. Підпис токена обчислюється на основі інформації у заголовку і корисному навантаженні, надаючи ще один рівень надійності та цілісності інформації, що обмінюється.

Отже, відомості про використання JWT наразі дуже актуальні, бо цей механізм не лише спрощує авторизацію користувачів, але й забезпечує безпечний обмін даними між системами, додатками та сервісами.

У спрощеному та лаконічному вигляді, JSON Web Tokens (JWT) складаються з трьох основних компонентів, які розділені між собою крапками:

- Заголовок**:** Перший компонент JWT − це його заголовок. В цьому заголовку містяться метадані, які описують формат та алгоритм підпису токена. Важливою частиною цього заголовку є інформація про тип токена та використаний алгоритм шифрування. Завдяки цій інформації,

отримувач може коректно обробити токен та визначити, яким чином він повинен бути розкодований і перевірений.

- Корисне навантаження**:** Другий компонент − це корисне навантаження (payload). Ця частина токена містить конкретні дані або твердження, які відправник додав до токена. Інформація у корисному навантаженні може бути різноманітною і включати, наприклад, ідентифікатор користувача, час створення токена, обмеження дії токена та інші корисні дані. Важливо зауважити, що ця частина токена не є зашифрованою, тому інформація в ній може бути прочитана кожним, хто має доступ до токена.
- Підпис**:** Останній компонент JWT − це підпис (signature). Після створення заголовку та корисного навантаження, вони об'єднуються та підписуються за допомогою певного алгоритму та секретного ключа. Цей підпис дозволяє перевірити цілісність токена та визначити, чи був токен змінений після його створення. Призначення підпису − забезпечити впевненість, що токен був створений відповідно до правил та не був підроблений.

Отже, стандартний JWT зазвичай має вигляд, xxxxx.yyyyyy.zzzzz, що показаний на рис. 1.6. Це компактний спосіб представлення важливої інформації у форматі, який забезпечує безпеку та цілісність даних.

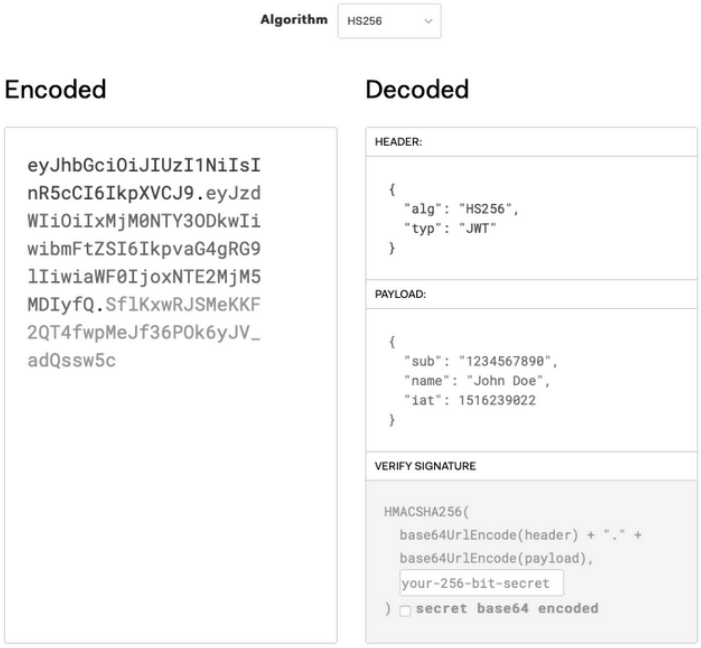


Рис. 1.6 Приклад стандартного JWT

Під час процесу автентифікації, коли користувач успішно пройшов процедуру входу в систему, використовуючи свої облікові дані, система генерує і видаватиме користувачеві JSON-токен у форматі JSON. Значущим є те, що ці токени насправді є критичними обліковими даними, які потрібно обережно зберігати та обробляти для запобігання можливим проблемам безпеки. Важливо мати на увазі, що токени не повинні залишатися активними довше, ніж це є необхідним, і уникати зберігання конфіденційних даних сеансу в сховищі браузера через відомі ризики з точки зору безпеки.

Іншим ключовим моментом є той факт, що при кожній спробі користувача отримати доступ до захищеного маршруту або ресурсу, агент користувача зазвичай повинен надсилати JWT. Цей JWT, як правило, передається у заголовку запиту авторизації, використовуючи певну схему пред'явника. Цей механізм може слугувати механізмом авторизації без необхідності зберігання стану (stateless). При використанні захищених маршрутів на стороні сервера відбувається перевірка наявності дійсного JWT в заголовку Authorization. Якщо JWT вказаний та вірний, користувачеві надається доступ до захищених ресурсів. Важливо відзначити, що наявність певних даних в JWT може сприяти зменшенню необхідності у виконанні запитів до бази даних для виконання певних операцій, хоча це не є завжди обов'язковим чинником.

Завжди слід пам'ятати, що правильна обробка та управління JWT грають ключову роль у забезпеченні безпеки і здійсненні авторизації, і ця техніка має важливе значення в розробці захищених додатків та систем.

2 РЕАЛІЗАЦІЯ АУТЕНТИФІКАЦІЇ НА ПРАКТИЦІ

2.1. Популярні фреймворки та бібліотеки

На сьогоднішній день існує безліч популярних фреймворків та бібліотек для різних мов програмування та областей розробки. Вибір конкретного фреймворку або бібліотеки залежить від задачі та специфіки проекту. Ось деякі з популярних фреймворків та бібліотек у різних областях Веб розробки:

Django (Python) − це високорівневий фреймворк для розробки веб-додатків на мові програмування Python. Він надає зручність та продуктивність для швидкої розробки веб-сайтів та веб-додатків.

Ruby on Rails (Ruby), також відомий як Rails, − це фреймворк для розробки веб-додатків на мові програмування Ruby. Він відомий своєю простотою та швидкістю розробки.

Express.js (JavaScript/Node.js) − це мінімальний та гнучкий фреймворк для створення веб-додатків на мові програмування JavaScript з використанням Node.js.

React (JavaScript) − це бібліотека для створення інтерфейсів користувача (UI) на JavaScript. Вона використовується для розробки односторінкових додатків та додатків з багатьма сторінками.

Angular (JavaScript/TypeScript) − це фреймворк для створення веб-додатків на JavaScript або TypeScript. Він надає інструменти для розробки складних веб-додатків з великим обсягом коду.

Spring Boot (Java) − це фреймворк для розробки веб-додатків на мові програмування Java. Він надає потужні засоби для створення високопродуктивних веб-сервісів та додатків з меншим обсягом конфігурації.

Flask (Python) − це легкий фреймворк для розробки веб-додатків на мові програмування Python. Він дозволяє створювати веб-сервери та RESTful API.

jQuery (JavaScript) − це бібліотека JavaScript, яка спрощує взаємодію з DOM-деревом сторінки, обробку подій та виконання AJAX-запитів.

Bootstrap (HTML/CSS/JavaScript) − це набір CSS-фреймворків та JavaScript-компонентів, які допомагають швидко створювати стильні та відзначені веб-сайти та додатки.

2.2. Налаштування середовища розробки

Для демонстрації роботи WEB-додатку з використанням аутентифікації буде написано код на мові програмування Python з використанням фреймворку Flask.

Python було обрано через його читабельність та широкі можливості, а Flask часто використовується для створення веб-додатків, прототипів, малих та середніх проектів, RESTful API та багатьох інших сценаріїв веб-розробки. Він надає розробникам великий контроль та гнучкість у розробці веб-додатків та допомагає швидко реалізовувати ідеї.

Легкість використання − Flask призначений для простоти. Можна швидко створити веб-додаток або RESTful API за допомогою всього кількох рядків коду.

Маршрутизація Flask − дозволяє визначити URL-шляхи та пов'язати їх з функціями-обробниками. Це робить проект більш організованим та легким для розуміння.

Інтеграція з іншими бібліотеками − фреймворк може легко інтегруватися з іншими популярними бібліотеками, такими як SQLAlchemy для роботи з базами даних, WTForms для обробки форм та Flask-SocketIO для реального часу.

Розширення − Flask має широкий вибір розширень, які дозволяють розширити функціональність додатку. Наприклад, Flask-Login для автентифікації користувачів або Flask-RESTful для створення RESTful API.

Спрощена конфігурація − Flask дозволяє налаштувати додаток з використанням файлів конфігурації або змінних середовища, що спрощує управління параметрами.

Спільнота та документація - Flask має активну спільноту користувачів, готових відповідати на питання та надавати допомогу. Крім того, існує велика кількість документації, яка допоможе вам вивчити фреймворк.

Для розробки обрав редактор Visual Studio Code від компанії Microsoft, який виявився ідеальним інструментом завдяки своїй гнучкості та інтуїтивно зрозумілому інтерфейсу.

Visual Studio Code підтримує Python та пропонує вбудовані функції, такі як підсвітка синтаксису та автодоповнення коду, що значно спрощує процес програмування. Цей редактор також дозволяє легко використовувати різні розширення для Flask, що є корисним для розробки веб-додатків. Завдяки вбудованому терміналу та інтеграції з Git, VS Code забезпечує ефективне середовище для розробки, тестування та управління версіями коду.

Під час розробки увагу було зосереджено на створенні маршрутів, шаблонів та обробників запитів. Використання Visual Studio Code значно спростило цей процес, дозволяючи швидко вносити зміни в код та перевіряти їх в дії.

Загалом, Visual Studio Code від Microsoft відмінний вибір для розробки веб-додатків на Python із використанням Flask. Його гнучкість, розширюваність і численні корисні функції роблять його популярним серед розробників різного рівня.

Після встановлення редактору коду та інтерпретатора Python версії 3.7.6 виконалось встановлення фреймворку Flask за допомогою команди (pip install Flask). Також послідовно було виконано встановлення бібліотеки Flask-OAuthlib та jwt (pip install Flask-OAuthlib jwt).

Flask-OAuthlib − це розширення для фреймворку Flask, яке надає підтримку для реалізації OAuth 1.0 та OAuth 2.0 аутентифікації WEB -додатку.

Jwt − це бібліотека для роботи з JWT (JSON Web Tokens) в мові програмування Python. Вона дозволяє створювати, перевіряти та розкодовувати JWT токени, що використовуються для автентифікації та авторизації в веб-додатках. Бібліотека python-jwt дозволяє легко створювати та перевіряти цифрові підписи JWT токенів, забезпечуючи безпеку та достовірність даних.

2.3. Розробка WEB-додатка з використанням аутентифікації OAuth2

Для початку роботи було розроблено потік процесу автентифікації в додатку, зображено на рисунку 2.1, який ілюструє процес OAuth2 автентифікації у веб-додатку.

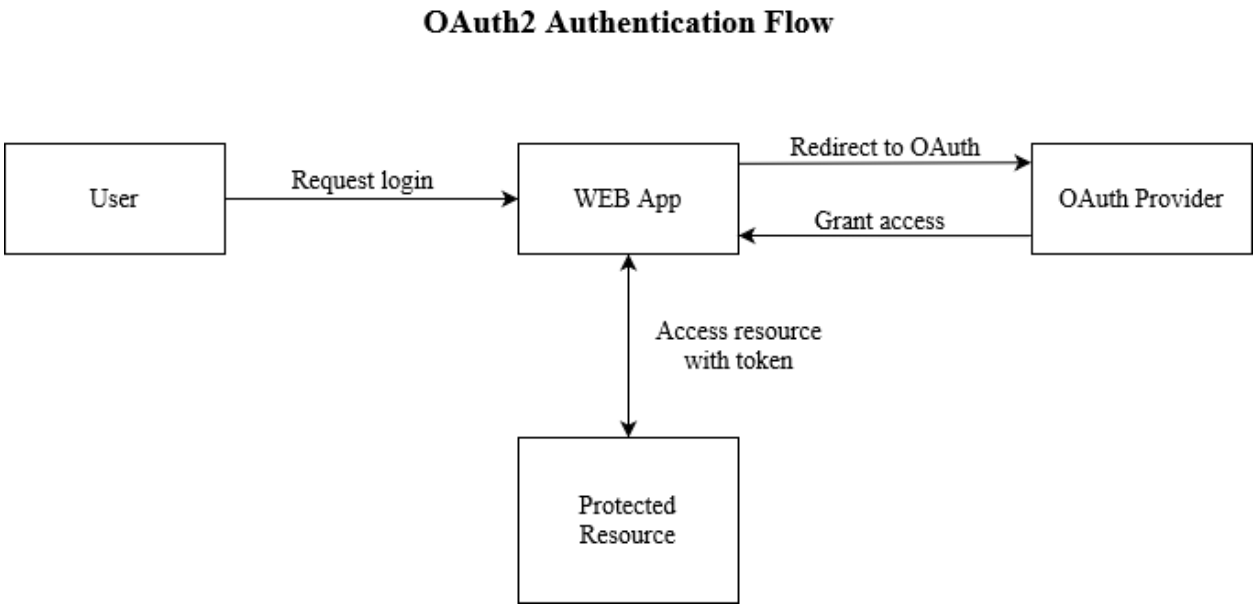


Рис. 2.1 Потік OAuth2 автентифікації в додатку

Діаграма відображає ключові кроки та взаємодію між користувачем, веб-додатком і OAuth2 провайдером:

- Користувач ініціює запит на вхід у веб-додатку.
- Веб-додаток перенаправляє користувача до сторінки авторизації OAuth провайдера.
- Користувач надає доступ і OAuth провайдер видає токен доступу.
- Веб-додаток отримує токен доступу від провайдера.
- Веб-додаток використовує токен доступу для отримання захищених ресурсів.

Для деталізування і глибшого розуміння процесу налаштування OAuth2 провайдера, розглянемо цей процес на конкретному прикладі Google. Важливо зазначити, що налаштування провайдера OAuth2 може варіюватися залежно від

конкретного провайдера, і в даному випадку розглянуто приклад налаштування для Google.

Спершу, потрібно перейти до консолі Google Cloud (рис. 2.2) та обрати або створити проєкт, для якого буде налаштовуватись OAuth2. Наприклад, створено проєкт з назвою "OAuth for DUT" для цієї роботи.

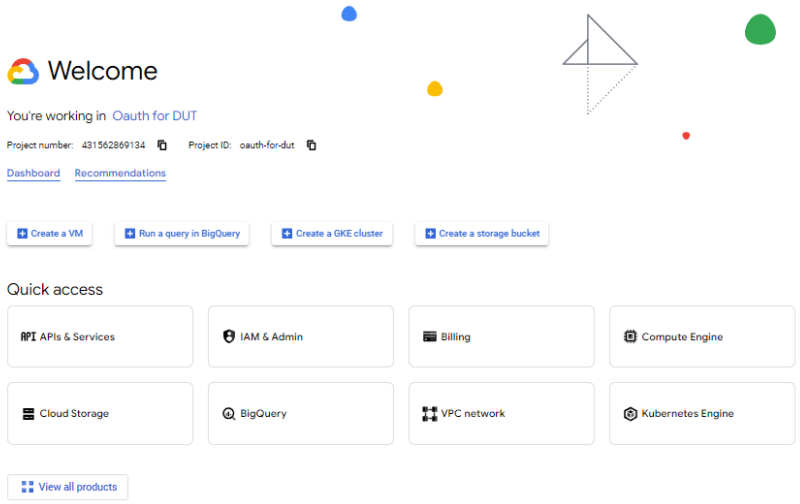


Рис. 2.2 Головна сторінка Google Cloud

Потім перейти до розділу "APIs & Services" і перейти на вкладку "OAuth consent screen". Тут можна обрати тип доступу OAuth до додатку. Можливі варіанти включають доступ для користувачів однієї організації (internal) або доступ для будь-якого користувача (external), який має обліковий запис Google. У прикладі обрано "external" (рис. 2.3), оскільки для тестування достатньо одного користувача, якого не потрібно включати в організацію.

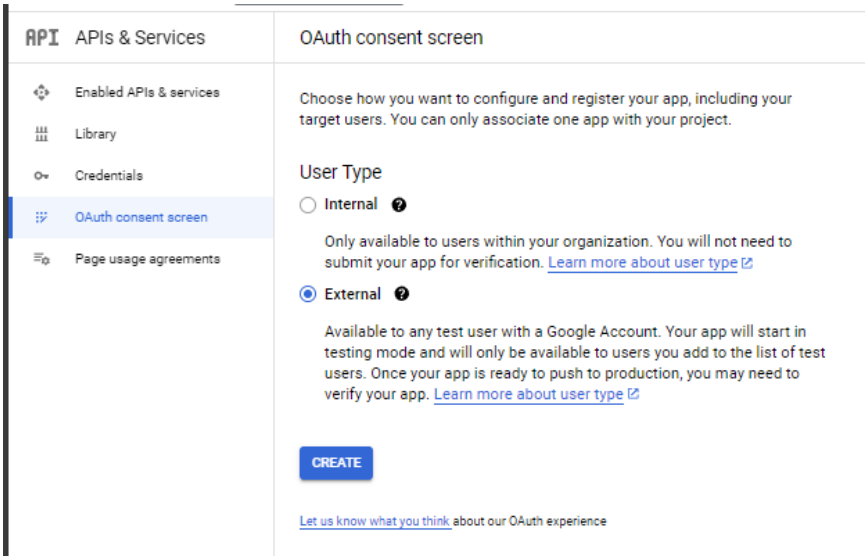


Рис. 2.3 Вкладка "OAuth consent screen"

Після цього надається доступ до реєстрації додатку (рис. 2.4), де потрібно буде ввести назву додатку та контактний email для підтримки користувачів, які проходитимуть аутентифікацію.

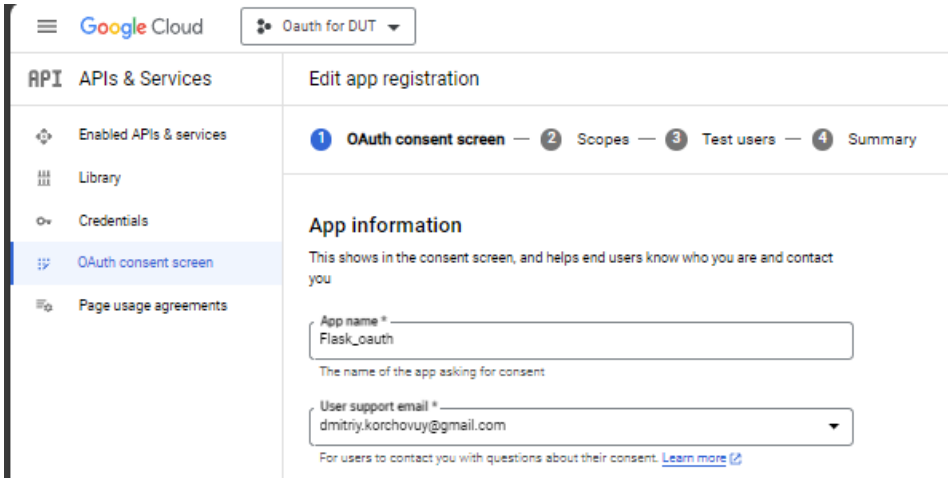


Рис. 2.4 Реєстрація додатку

Наступним кроком буде додавання користувачів до тестового переліку. Це користувачі, яким буде надано доступ до виконання аутентифікації в тестовому режимі. Цей крок дозволяє визначити, хто може взаємодіяти з додатком під час тестування.

Фінальним і важливим кроком для налаштування зв’язку з провайдером OAuth2 є створення секретного ключа для підключення WEB-додатку. Для цього потрібно перейти у вкладку "Credentials" і натиснути "Create Credentials" з типом "OAuth client ID", обравши тип додатку "Web application" (рис. 2.5).

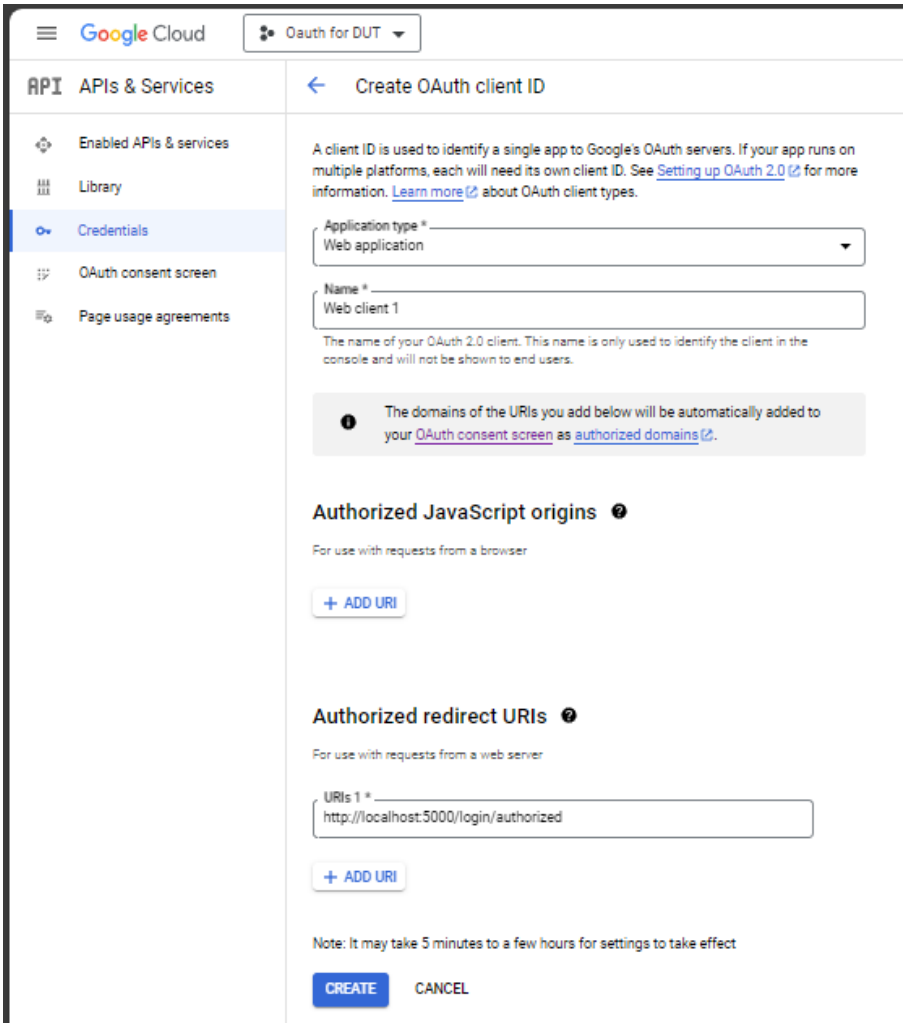


Рис. 2.5 Вікно конфігурації зв’язку додатку з Google OAuth2

У відкритому екрані потрібно буде ввести посилання для переадресації, на яке користувачі будуть переспрямовані після автентифікації в Google. Важливою частиною цього посилання є код авторизації для доступу, який повинен відповідати протоколу. Код авторизації не може містити фрагменти URL-адреси, відносні шляхи або символи узагальнення, і також не може бути загальнодоступною IP-адресою.

В закінчення процесу підключення, буде надано можливість завантажити JSON-файл з даними для підключення додатку. Цей файл міститиме інформацію, таку як client_id, auth_uri, token_uri та client_secret, яка необхідна для забезпечення зв’язку додатку з провайдером OAuth2, в даному випадку Google.

Перейдемо до покрокового розгляду показаних секції Python коду з описом та аналізом функцій за які вони відповідають:

1. Імпорт бібліотек і створення Flask додатку:

```
python
from flask import Flask, request, redirect, session, url_for
from flask_oauthlib.client import OAuth
import os

app = Flask(__name__)
app.secret_key = os.urandom(24)
```

Тут імпортуються необхідні бібліотеки, створюється екземпляр Flask, і генерується секретний ключ для сесій.

2. Конфігурація OAuth з Google:

```
python
oauth = OAuth(app)

google = oauth.remote_app(
    'google',
    consumer_key='YOUR_CONSUMER_KEY',
    consumer_secret='client_secret',
    request_token_params={
        'scope': 'email',  # Можна додати додаткові scopes
    },
```

```
    base_url='https://www.googleapis.com/oauth2/v1/',
    request_token_url=None,
    access_token_method='POST',
    access_token_url='https://accounts.google.com/o/oauth2/token',
    authorize_url='https://accounts.google.com/o/oauth2/auth',
)
```

В цій частині конфігурується OAuth клієнт для Google з усіма необхідними параметрами та URL-адресами що були отримані на етапі реєстрації додатку в Google. В якості request_token_params буде запитуватись email, так як він не може бути змінений для користувача ні в якому разі, і він є унікальним параметром.

3. Маршрути Flask додатку:

```
python
@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login')
def login():
    return google.authorize(callback=url_for('authorized', _external=True))

@app.route('/logout')
def logout():
    session.pop('google_token', None)
    return 'Logged out'

@app.route('/login/authorized')
def authorized():
    response = google.authorized_response()
```

```
    if response is None or response.get('access_token') is None:
        return 'Access denied: reason={} error={}'.format(
            request.args['error_reason'],
            request.args['error_description']
        )
    session['google_token'] = (response['access_token'], '')
    user_info = google.get('userinfo')
    return 'Logged in as: ' + user_info.data['email']
```

Тут визначаються маршрути для основних дій у додатку: вхід, вихід, авторизація та перевірка результату авторизації.

4. Отримання токена доступу:

```
python
@google.tokengetter
def get_google_oauth_token():
    return session.get('google_token')
```

Ця функція використовується для отримання поточного токена доступу з сесії користувача.

5. Запуск додатку:

```
python
if __name__ == '__main__':
    app.run(debug=True)
```

Якщо файл виконується як основний, додаток запускається в дебаг режимі.

Цей лістинг демонструє, як інтегрувати аутентифікацію через Google у веб-додаток на Flask, використовуючи OAuth 2.0. Повний код застосунку наведений в Додатку А.

2.4. Розробка WEB-додатка з використанням аутентифікації JWT

Для початку роботи було розроблено потік процесу автентифікації в додатку, зображено на рисунку 2.6, який ілюструє процес JWT автентифікації у веб-додатку.

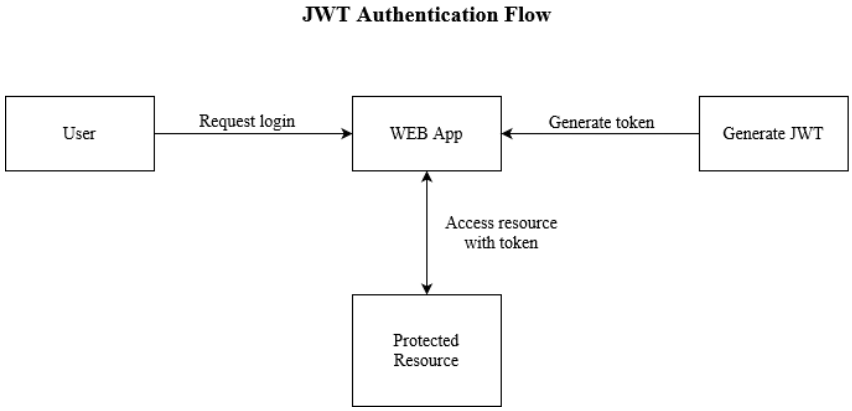


Рис. 2.6 Потік JWT автентифікації в додатку

Діаграма (рис. 2.6) ілюструє процес аутентифікації за допомогою JWT у веб-додатку, розробленому на Flask. Діаграма показує основні етапи взаємодії між користувачем, веб-додатком та *процесом* генерації та використання JWT:

1. Користувач ініціює запит на вхід у веб-додатку.

2. Веб-додаток обробляє запит та генерує JWT.

3. JWT передається назад користувачу.

4. Користувач використовує отриманий JWT для доступу до захищених ресурсів веб-додатку.

Ця діаграма візуалізує процес аутентифікації за допомогою JWT, показуючи, як користувачі можуть отримувати та використовувати токени для безпечного доступу до веб-додатку.

Цей код демонструє використання JSON Web Tokens (JWT) для аутентифікації у веб-додатку на базі Flask.

1. Імпорт бібліотек і створення Flask додатку:

```python
from flask import Flask, request, jsonify
```

```
import jwt
import datetime


app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'  # Замініть на секретний ключ за вашим вибором
```

Тут імпортуються необхідні бібліотеки, створюється екземпляр Flask, і задається секретний ключ для підпису JWT.

2. Функція для генерації JWT-токена:

```python
def generate_token(username):
    payload = {
        'username': username,
        'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=1)  # Токен дійсний протягом 1 години
    }
    token = jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')
    return token
```

Ця функція генерує JWT, який містить інформацію про користувача та час його дійсності.

3. Маршрут для аутентифікації і отримання JWT-токена:

```
python
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    # Тут налаштовується логіка для аутентифікації користувача на основі переданих даних
    if username == 'your_username' and password == 'your_password':
        token = generate_token(username)
        return jsonify({'access_token': token})
    else:
        return jsonify({'message': 'Authentication failed'}), 401
```

У цьому маршруті користувач вводить свої дані, які перевіряються, і в разі успішної аутентифікації генерується JWT.

4. Маршрут для захищених ресурсів:

```
python
@app.route('/protected', methods=['GET'])
def protected_resource():
    token = request.headers.get('Authorization')

    if not token:
        return jsonify({'message': 'Token is missing'}), 401

    try:
        payload =
```

```
        jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
            return jsonify({'message': 'Welcome, ' + payload['username']})
        except jwt.ExpiredSignatureError:
            return jsonify({'message': 'Token has expired'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'message': 'Invalid token'}), 401
```

Цей маршрут перевіряє наявність та дійсність JWT у запиті. Якщо токен дійсний, користувач отримує доступ до захищеного ресурсу.

5. Запуск додатку:

```
python
if __name__ == '__main__':
    app.run(debug=True)
```

Якщо файл виконується як основний, додаток запускається в дебаг режимі.

Цей код є прикладом використання JWT для аутентифікації у Flask додатку. Він демонструє, як генерувати токени, а також як їх перевіряти при доступі до захищених ресурсів. Повний код застосунку наведений в Додатку Б.

2.5. Порівняння реалізацій та результати тестування

Порівняння OAuth 2.0 та JWT (JSON Web Tokens) за їх призначенням та способом використання допоможе краще зрозуміти ключові відмінності між цими двома механізмами.

У процесі авторизації OAuth 2.0, користувач надає додатку "токен доступу" від посередника (наприклад, Google або Facebook), який дозволяє додатку доступатися до певних інформаційних ресурсів на відповідному сервері. Цей токен набагато більш компактний і має одну основну функцію − надавати доступ до

ресурсів. Токен не містить інформації про користувача і використовується як ключ для доступу до ресурсів.

З іншого боку, JWT містить інформацію (payload), яка може включати дані про користувача і додаткові метадані. Цей токен підписується секретним ключем або публічним/приватним ключем і може містити значиму інформацію про користувача та інші деталі. JWT використовується для перевірки того, що запит відправлено автентифікованим користувачем, і може бути використаний для передачі даних між клієнтом та сервером.

Основна відмінність полягає у тому, що OAuth 2.0 зосереджений на делегуванні доступу до ресурсів без розкриття облікових даних користувача, тоді як JWT використовується для аутентифікації та забезпечення обміну інформацією між сторонами. OAuth 2.0 дозволяє додаткам отримувати доступ до ресурсів в ім'я користувача без передачі облікових даних, тоді як JWT надає засіб для передачі корисної інформації разом з підтвердженням ідентичності.

Токени OAuth 2.0 є засобом доступу до ресурсів і не містять інформації про користувача, тоді як JWT містить інформацію про користувача і використовується для підтвердження його ідентичності, а також для обміну корисною інформацією між сторонами. Отже, вибір між цими двома механізмами повинен враховувати конкретні потреби та цілі вашого проекту.

Хоча JWT і OAuth служать різним цілям, можливість їх поєднання в одному проекті відкриває безмежні можливості для забезпечення безпеки та розширення функціональності. Основна відмінність полягає в тому, що OAuth стандартизує процес авторизації та делегування доступу до ресурсів, тоді як JWT використовується для забезпечення цілісності даних та ідентифікації користувачів.

OAuth не накладає жодних обмежень на формат токенів, що використовуються для представлення прав доступу. Це означає, що ви можете використовувати JWT як один із форматів токенів в OAuth. Наприклад, сервер аутентифікації OAuth2 може видавати токен доступу, який включає JWT як частину свого корисного навантаження. Цей JWT може містити додаткову інформацію, таку як роль користувача або додаткові властивості, що покращують продуктивність. Це

може суттєво зменшити кількість обмінів даними між сервером автентифікації та сервером ресурсів.

Більш того, поєднання JWT і OAuth2 може відбуватися за допомогою підходу з двома маркерами. В такому випадку OAuth2 видаватиме два окремих маркери: access_token та JWT. Останній міститиме додаткову ідентифікаційну інформацію про користувача та надаватиме більше можливостей для контролю доступу та обміну даними.

Необхідно пам'ятати про важливість використання OpenID Connect, коли на виборі стратегія з двома токенами. OpenID Connect базується на OAuth2 і надає стандартизовані поля токенів, що полегшує обмін інформацією та ідентифікацію користувачів.

Використання JWT замість OAuth2 може значно полегшити і прискорити виконання певних завдань у проекті. Однак це також може зробити розробку більш складною. Тому приймаючи рішення про використання JWT разом з OAuth2, важливо обдумати, чи виправдовується витратами на розробку можливий приріст продуктивності та безпеки проекту.

На рис. 2.7 приклад роботи OAuth2 зі створеного Python додатку, який відображає перенаправлену сторінку для входу в додаток через стороннього провайдера, в даному випадку Google.

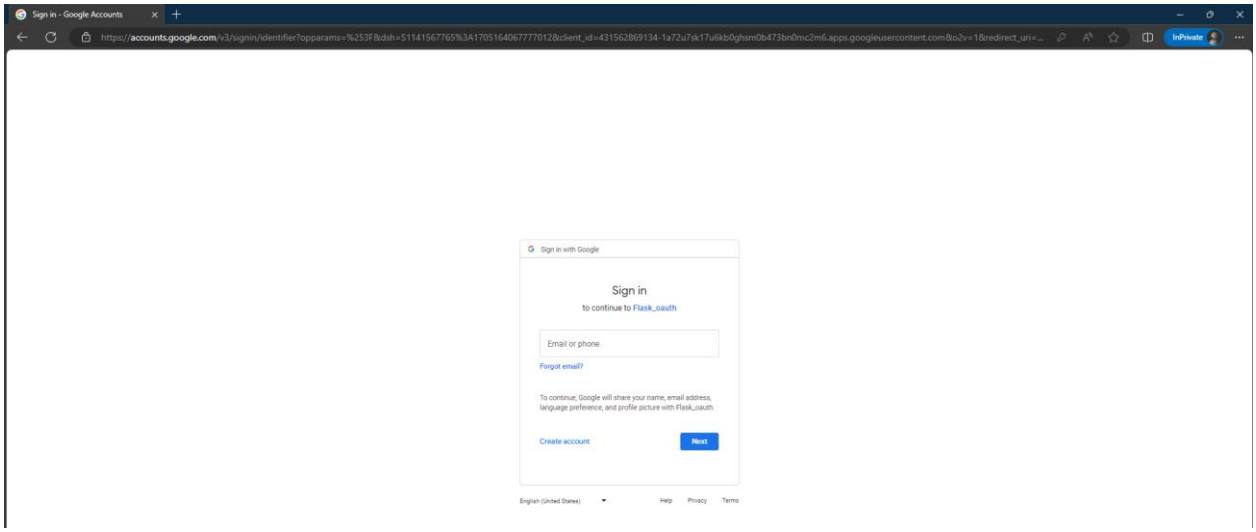


Рис. 2.7 Екран входу в додаток за допомогою OAuth2 Google

Після вводу автентифікаційних даних користувача відбувається перевірка чи він є в списку «дозволених» для тестування і далі при отриманні токена додаток запрошує у провайдера для прикладу e-mail клієнта (рис 2.8).

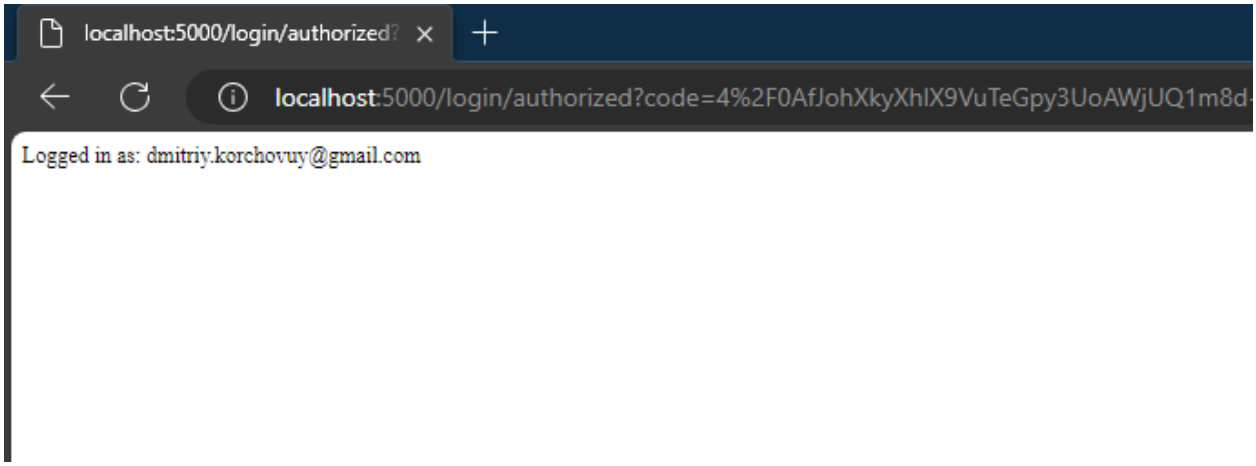


Рис. 2.8 Результат після автентифікації

Далі показано тестування Python застосунку з JWT автентифікацією клієнта. На першому скріншоті (рис. 2.9) показано процес виконання HTTP POST запиту для отримання токену. В середині, в тілі POST запиту передаються облікові дані клієнта для базової авторизації. Додаток після отримання запиту і перевірки облікових даних, в разі успіху передає відповідь (рис. 2.10) з JWT токеном (access_token). В разі передачі невірного паролю отримано відповідь "message": "Authentication failed", що означає що в доступі відмовлено.

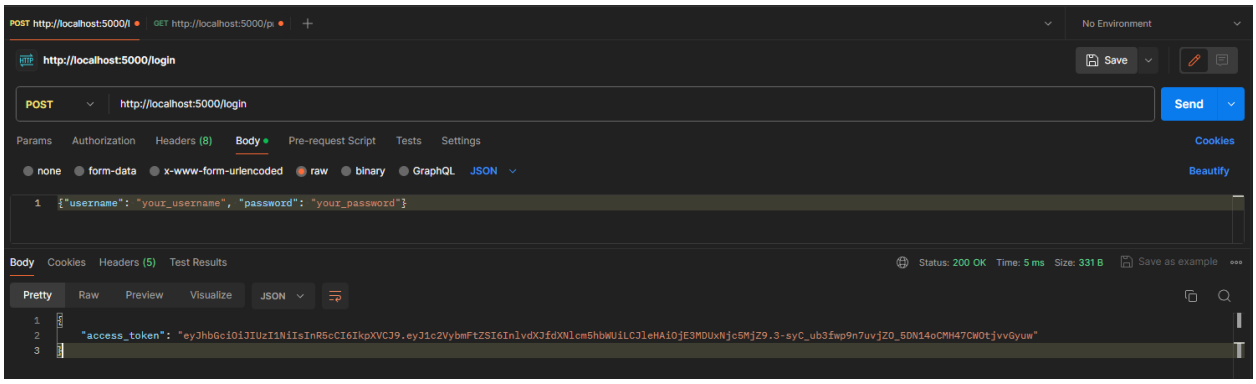


Рис. 2.9 Успішне виконання запиту для отримання JWT токену

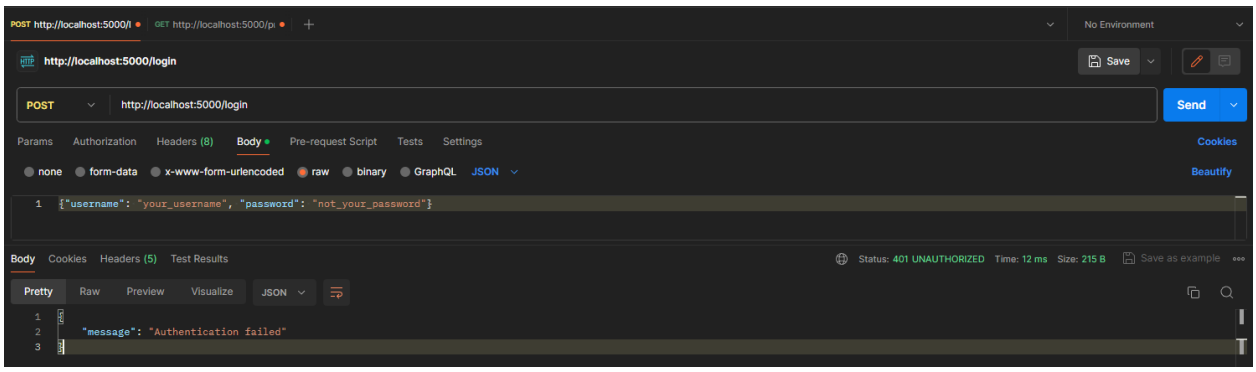


Рис. 2.10 Не успішне виконання запиту для отримання JWT токену

Після отримання токену, виконується HTTP GET запит за захищеним шляхом (/protected), для цього додано заголовок "Authorization" з токеном в середині заголовку, заголовок має такий вигляд:

--header 'Authorization: xxxxx.yyyyyy.zzzzz'

В разі отримання додатком невірного токену буде повернена відповідь "message": "Invalid token", а якщо токен буде протермінований то отримана відповідь "message": "Token has expired". В обох випадках статус відповіді буде 401(UNAUTHORIZED)

При отриманні валідного токену додаток повертає відповідь "message": "Welcome, your_username" зі статусом 200(OK), як відображено на скріншоті (рис. 2.11).

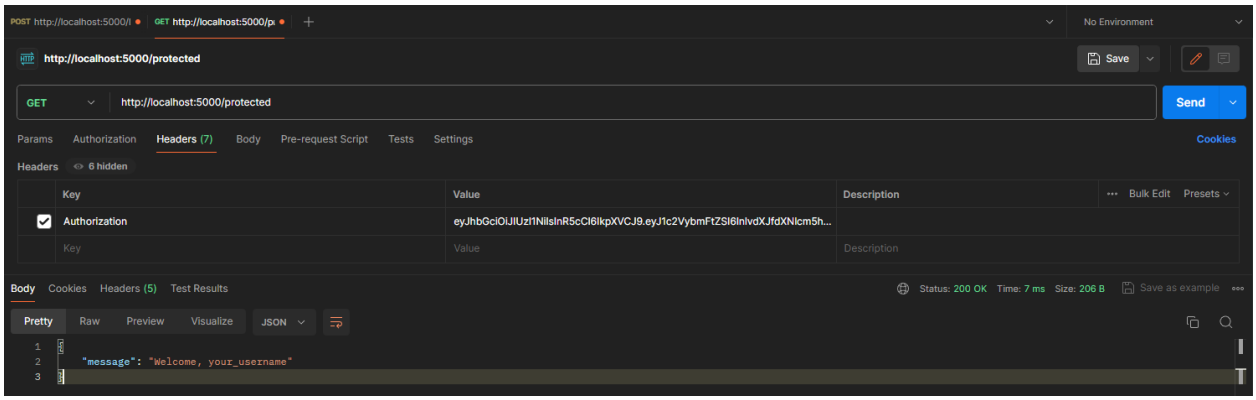


Рис. 2.11 Результат використання валідного JWT токенавв

3 АНАЛІЗ І ПОРІВНЯННЯ РЕЗУЛЬТАТІВ

3.1. Аналіз сучасних рішень з використанням OAuth2 та JWT

У даній частині, оглянемо популярні реалізацій протоколі OAuth2 та JWT, які стали важливою частиною сучасної інтернет-екосистеми. Огляд функціональності та безпеки кожної з них допоможе краще зрозуміти їх особливості та використання.

- Google OAuth2: Ця реалізація OAuth2 від Google є однією з найпоширеніших і добре відомих серед користувачів. Вона дозволяє використовувати обліковий запис Google для увіходу в різноманітні додатки та сервіси. Google OAuth2 забезпечує високу ступінь безпеки та зручності, що робить його важливим для багатьох розробників.
- Facebook OAuth2: Інша відома реалізація OAuth2 від Facebook, яка надає можливість користувачам авторизуватися через свій обліковий запис на цій популярній соціальній мережі. Facebook OAuth2 часто використовується у соціальних додатках та веб-сервісах для спрощення процесу входу користувачів.
- GitHub OAuth2: GitHub використовує OAuth2 для авторизації та надання доступу до свого API. Це робить GitHub OAuth2 важливим для розробників, які працюють з репозиторіями та кодом на цій платформі. Він дозволяє стороннім додаткам взаємодіяти з GitHub і використовувати його ресурси.
- Auth0: Сервіс Auth0 надає інструменти для реалізації OAuth2 в різних додатках та сервісах. Його гнучкість і можливості роблять його важливим інструментом для розробників, які шукають швидке та безпечне вирішення авторизації та аутентифікації.
- Firebase Authentication**:** Надає розширені можливості автентифікації, включаючи підтримку різних методів автентифікації, таких як електронна

пошта, соціальні мережі, номери телефонів тощо. Він також має інструменти для управління користувачами та дозволами.

- Auth0**:** Надає розширені можливості авторизації та аутентифікації, включаючи підтримку соціальних мереж, двофакторну аутентифікацію, управління дозволами, аналітику та інші функції. Він може бути інтегрований в різні типи додатків.
- Node.js бібліотеки**:** Існують різні бібліотеки для роботи з JWT. Найпоширеніша з них - "jsonwebtoken", яка дозволяє створювати, перевіряти та розшифровувати JWT-токени. Функціональність може варіюватися в залежності від конкретної бібліотеки та потреб вашого проекту.
- Django REST**:** REST Framework − це фреймворк для розробки RESTful API на Python, який має вбудовану підтримку JWT. Він надає можливість автентифікації користувачів, створення та оновлення JWT-токенів для різних потреб додатків.
- Spring Security**:** Це фреймворк для автентифікації та авторизації в додатках на платформі Java. Він також має підтримку JWT для автентифікації та авторизації користувачів.

Ці реалізації OAuth2 та JWT різняться за своєю функціональністю, специфічними особливостями та масштабом використання.

Порівняємо основні реалізацій безпекових функцій протоколу OAuth2 та JWT, щоб краще зрозуміти їхні відмінності та переваги:

- Google OAuth2: Враховує важливі аспекти безпеки, такі як захист від CSRF-атак та перехоплення токенів. Також вона надає можливість обмеження дозволів та аудиту доступу.
- Facebook OAuth2: Вживає заходів безпеки для захисту користувачів від атак та недозволених доступів. Вона також підтримує шифрування даних та має механізми обмеження доступу.

- GitHub OAuth2: Забезпечує безпеку взаємодії з репозиторіями та ресурсами GitHub. Вона використовує заходи для запобігання атак та перехоплення токенів.
- Auth0: Високо цінує безпеку та надає інструменти для захисту додатків, що використовують OAuth2. Вона включає в себе різноманітні заходи безпеки, такі як двофакторна аутентифікація та моніторинг безпеки.
- Firebase Authentication: Забезпечує захист від CSRF-атак та надає можливість обмежувати дозволи користувачів. Токени зазвичай зберігаються на стороні клієнта.
- Auth0: Забезпечує захист від CSRF-атак, перехоплення токенів та шифрування даних. Він має багатошарову систему безпеки.
- Node.js бібліотеки: Безпека JWT залежить від правильної імплементації та використання. Бібліотеки можуть надавати основні заходи безпеки, але їхнє використання варто ретельно перевіряти.
- Django REST: Має вбудовані заходи безпеки, включаючи захист від CSRF-атак і обмеження доступу. Використання JWT з цим фреймворком може забезпечувати високий рівень безпеки.
- Spring Security: Надає широкі можливості для налаштування заходів безпеки, включаючи обробку JWT-токенів. Це дозволяє забезпечити високий рівень безпеки в додатках на Java.

Цей детальний порівняльний аналіз допоможе визначити, яка реалізація OAuth2 та найкраще відповідає конкретним потребам та вимогам з огляду на їхню функціональність та безпеку.

Загалом, вибір між реалізаціями OAuth2 та JWT повинен залежати від конкретних потреб вашого проекту, його масштабу та вимог до безпеки та функціональності. Обидва протоколи мають свої унікальні переваги і обмеження, і правильний вибір залежатиме від контексту вашого застосунку.

Таким чином, розуміння потреб вашого проекту і вимог до безпеки є вирішальними при виборі між OAuth2 та JWT. Важливо провести ретельний аналіз і обрати той протокол, який найкраще відповідає вашим потребам, а також

врахувати можливість їх комбінування для досягнення більш високого рівня безпеки та функціональності в вашому проекті.

3.2. Переваги та недоліки використання OAuth2 та JWT

OAuth і JWT представляють собою два засадничі стандарти, які відіграють ключову роль у забезпеченні авторизації та автентифікації в інтернет-середовищі. Кожен з цих стандартів має свої унікальні особливості та застосування, які розглянемо більш докладно.

OAuth, як стандарт авторизації, використовується для делегування авторизаційних даних користувачів між різними сторонніми додатками та платформами. Його основна мета полягає у наданні дозволу користувача на доступ до його ресурсів, не вимагаючи від користувача надавати свої облікові дані на інших веб-сайтах. OAuth стає незамінним інструментом для веб-додатків, які потребують авторизації через сторонні сервіси, і дозволяє ефективно управляти сеансами та дозволами користувачів.

З іншого боку, JWT (JSON Web Tokens) − це стандарт для створення та передачі токенів, які можуть бути використані для автентифікації користувачів і забезпечення безпеки у веб-додатках та сервісах. Його сфера застосування включає бездержавні додатки, автентифікацію API та навіть авторизацію між серверами. JWT здатний зберігати інформацію в структурованому форматі JSON та підписувати її цифровим підписом для підтвердження її цілісності. Цей стандарт використовується для забезпечення автентичності даних та авторизації у розподілених системах.

Важливо відзначити, що обидва стандарти виконують важливі функції у сфері безпеки та авторизації в інтернеті. Вибір між ними залежатиме від конкретних потреб та вимог вашого проекту. Розуміння ключових відмінностей між OAuth та JWT допоможе вам зробити правильний вибір для вашого додатка або сервісу.

Давайте подивимось на низку важливих відмінностей між OAuth і JWT, щоб краще зрозуміти їхню роль та застосування:

- Основна функція: OAuth використовується для авторизації, дозволяючи здійснювати делегування прав доступу, тоді як JWT використовується для автентифікації та обміну інформацією для підтвердження ідентичності користувача або клієнта.
- Безпека: OAuth слугує безпечним механізмом керування потоками авторизації, водночас JWT є легким та автономним токеном, який не забезпечує безпеку сам по собі, але може бути безпечним у контексті добре продуманої системи автентифікації.
- Незалежність від держави: JWT є незалежним від держави, що означає, що він не потребує зовнішнього джерела для підтвердження претензій. Він не вимагає централізованого сервера або бази даних для зберігання токенів. Однак OAuth залежить від держави, оскільки він вимагає з'єднання з сервером авторизації для отримання та перевірки токенів.
- Застосування: JWT ідеально підходить для бездержавних додатків, де не потрібно зберігати стан сеансу на сервері. З іншого боку, OAuth підтримує стан сеансу на сервері, і його застосування відмінно підходить для роботи з різними сторонніми додатками та ресурсами.
- Токени: JWT − це токен, який містить твердження про користувача або клієнта та може використовуватись для автентифікації. З іншого боку, OAuth використовує унікальний токен для надання доступу до ресурсів користувача. Токени OAuth є токенами безпеки, наданими постачальником токенів OAuth, і їх можна перевірити тільки тим самим постачальником токенів OAuth. JWT може використовуватись як інший вид токенів OAuth.

Ці два стандарти, OAuth і JWT, можна інтегрувати разом для створення максимально ефективної системи авторизації та автентифікації. Наприклад, при автентифікації користувача сервер автентифікації може використовувати OAuth для передачі даних користувача клієнтському додатку через безпечний механізм авторизації.

Переваги OAuth:

- Широке поширення: Однією з основних переваг OAuth є його широке поширення. Більшість служб автентифікації розуміють і підтримують OAuth 2.0, що робить його стандартом для багатьох онлайн-сервісів та додатків.
- Готові рішення: Велика кількість готових рішень для OAuth дозволяє вам швидко та легко інтегрувати його в ваші додатки. Популярні соціальні мережі, такі як Google та Facebook, підтримують OAuth, що робить можливим створення різноманітних опцій автентифікації для користувачів, таких як "увійти за допомогою Google" або "увійти за допомогою Facebook".
- Готові бібліотеки: OAuth має добре протестовані клієнтські бібліотеки для практично всіх мов програмування та веб-фреймворків. Це дає можливість розробникам використовувати OAuth з будь-якою технологією, яка їм зручна.
- Ізоляція коду: Однією з ключових переваг OAuth є можливість ізолювати код клієнтської програми від коду авторизації. Це забезпечує безпеку та спрощує процес розробки. OAuth є надійним і добре протестованим стандартом, який довів свою ефективність у польових умовах.

Недоліки OAuth:

- Складність для початківців: Для новачків OAuth може бути складним у розумінні, оскільки існує кілька різних потоків та процесів авторизації. Вибір правильного потоку для конкретного випадку може бути викликом.
- Зайвість для простих сценаріїв: Для деяких додатків, особливо тих, які мають один фронтенд і один бекенд, використання протоколу OAuth може здаватися зайвим та перевантажливим.
- Конфіденційність користувачів: OAuth може створювати питання з приводу конфіденційності користувачів. Наприклад, при використанні "Увійти за допомогою Google" Google може відстежувати активність

користувачів на різних веб-сайтах, що може порушити приватність користувачів.

Враховуючи ці переваги та недоліки, важливо ретельно обирати використання стандарту OAuth у вашому проекті та враховувати його специфіку та вимоги для досягнення найкращого результату.

Переваги JWT**:**

- Передача даних користувача: Однією з ключових переваг JWT є їх здатність передавати дані користувача без необхідності запитувати цю інформацію в базі даних або на сервері автентифікації при кожному запиті. Це робить обмін даними між клієнтом і сервером більш ефективним.
- Ефективна перевірка і швидкість: JWT можна перевіряти ефективно і швидко, оскільки вони містять всю необхідну інформацію для перевірки в самому токені. Це дозволяє уникнути звернень до бази даних або інших зовнішніх джерел для перевірки валідності.
- Збереження на стороні клієнта: JWT зберігаються лише на стороні клієнта, що робить їх більш легкими для обробки серверами. Сервер генерує JWT і передає їх клієнту, який потім включає їх у кожен запит. Це дозволяє заощадити ресурси та місце в базі даних.
- Гарантії безпеки: JWT забезпечують надійні гарантії безпеки, оскільки вони підписуються цифровим підписом. Це означає, що вони не можуть бути змінені клієнтами або зловмисниками, інакше підпис буде недійсним.

Недоліки JWT**:**

- Відсутність відкликання без додаткових зусиль: Один з недоліків JWT полягає в тому, що їх важко або навіть неможливо відкликати без значної додаткової інженерної роботи. Такий механізм відкликання зазвичай відсутній в стандарті.
- Складність негайного відкликання: Якщо потрібно негайно відкликати JWT (наприклад, в разі компрометації), це може виявитися часом,

споживаючим завданням. Це вимагає створення чорного списку JWT, що може бути затратним і часо- та ресурсозатратним процесом.

- Ризик в разі компрометації ключа підпису: Якщо ключ підпису JWT був скомпрометований зловмисником, то він може використовувати його для створення дійсних JWT. Це може призвести до підміни особи користувача і порушення безпеки системи.

Ураховуючи ці переваги та недоліки, важливо правильно використовувати JWT у вашому проекті та ураховувати їхні особливості для забезпечення безпеки та ефективності системи автентифікації та авторизації.

Обидва протоколи широко використовуються і підтримуються, але вони мають різні цілі та варіанти використання.

JWT є надзвичайно корисним інструментом для безпечної передачі інформації між різними сторонами в формі JSON-об'єкта. Цей стандарт здебільшого використовується для забезпечення безпеки та конфіденційності при обміні даними між різними компонентами системи.

Однією з головних функцій JWT є забезпечення надійної аутентифікації користувачів і надання їм авторизованого доступу до різних ресурсів, таких як особисті дані та файли. Для цього використовується процес генерації та перевірки токенів, які містять інформацію про користувача та їхні права.

Один із важливих аспектів JWT полягає в його компактності та автономності. Тобто весь необхідний контекст інформації міститься безпосередньо в токені, що робить зайвими додаткові запити до бази даних або серверу для перевірки інформації. Це дозволяє знизити накладні витрати на обмін даними та спрощує процес автентифікації.

JWT особливо підходить для розробки додатків без статусу. Це означає, що програма може легко автентифікувати користувачів і надавати доступ до ресурсів без необхідності зберігання стану сеансу на сервері. Кожний запит включає JWT, який містить інформацію про користувача та його права, і це дозволяє забезпечити безпеку і авторизований доступ до ресурсів на основі цього токена.

Узагальнюючи, JWT є потужним інструментом для забезпечення безпеки та автентифікації в веб-додатках та API. Він дозволяє ефективно передавати дані та забезпечувати авторизований доступ, зменшуючи накладні витрати та спрощуючи процес автентифікації.

OAuth, в порівнянні з JWT, відображає іншу філософію та має свої унікальні особливості. Один з найважливіших аспектів OAuth полягає в тому, що він зберігає стан сеансу на сервері, що робить його іншим від JWT.

OAuth використовує унікальний токен для надання доступу до ресурсів користувача на сторонньому веб-сайті або додатку. Це означає, що користувач може дозволити сторонньому додатку доступ до своїх ресурсів, таких як особисті дані, без необхідності надавати своє ім'я користувача та пароль.

Наприклад, OAuth дозволяє користувачам увійти в сторонні додатки, використовуючи обліковий запис на іншому веб-сайті. Наприклад, ви можете увійти в потоковий музичний сервіс, використовуючи свій обліковий запис Google, без необхідності надавати свій пароль потоковому сервісу.

Один з головних аспектів безпеки OAuth полягає в тому, що користувачеві надається безпечний спосіб дати дозвіл сторонньому додатку на доступ до своїх ресурсів, при цьому не розкриваючи свої облікові дані для входу. Це дозволяє забезпечити захист конфіденційності та безпеки користувача при використанні сторонніх додатків і сервісів.

Отже, OAuth є потужним інструментом для забезпечення безпеки та надання доступу до ресурсів користувача на сторонніх веб-сайтах та додатках, забезпечуючи при цьому захист конфіденційності та безпеки користувача.

JWT і OAuth2 - це два різних стандарти, які служать різним цілям, проте вони можуть спільно існувати та використовуватися разом, розширюючи можливості автентифікації та авторизації в сучасних додатках та сервісах.

Протокол OAuth2 зосереджений на делегуванні авторизації, керуванні доступом до ресурсів і управлінні сеансами користувачів. Однак OAuth2 не визначає формат токенів, які використовуються для надання доступу. Це відкриває можливості для використання JWT як формату токенів в рамках OAuth2.

Наприклад, сервер авторизації OAuth2 може видавати токен access_token у форматі JWT, який містить додаткову інформацію в корисному навантаженні. Це може призвести до оптимізації продуктивності, зменшивши кількість запитів між сервером ресурсів і сервером авторизації.

Іншим популярним сценарієм використання JWT з OAuth2 є видача двох різних токенів як access_token: токену доступу і JWT, який містить ідентифікаційну інформацію, додатково до токену доступу. Цей підхід може бути корисним в ситуаціях, коли потрібно забезпечити додаткову інформацію про користувача разом із токеном доступу. Проте для таких випадків рекомендується розглядати можливість використання OpenID Connect, розширення OAuth2, яке забезпечує додаткову нормалізацію шляхом включення полів access_token і id_token, спрощуючи і нормалізуючи обмін інформацією між ресурсами та авторизаційним сервером.

Таким чином, використання JWT і OAuth2 разом може допомогти розширити функціональність та забезпечити більше можливостей в сфері автентифікації та авторизації в сучасних додатках і сервісах.

Важливо зрозуміти, що використання JWT разом з OAuth2 не завжди гарантує підвищену безпеку додатків. JWT − це потужний і корисний механізм для передачі даних між сторонами у вигляді JSON-об'єкта, проте, як і будь-який інший стандарт, він не є непроникним до потенційних загроз безпеці.

Система безпеки OAuth2 побудована на визначенні суб'єктів, що беруть участь у процесі авторизації, і конкретних кроків, які необхідно виконати для здійснення цього процесу в різних випадках використання. Проблеми безпеки OAuth2 найкраще вирішуються шляхом вибору правильного потоку авторизації OAuth2 для вашого додатку на основі конкретного сценарію використання, а не за типом токену.

Однією з переваг використання JWT є покращена продуктивність та зниження складності певних процесів. JWT може бути швидшим і ефективнішим у випадках, коли потрібно передавати дані між сторонами без необхідності обробки кожного запиту до сервера авторизації.

Проте важливо пам'ятати, що використання JWT може також ускладнити розробку і вимагати більше зусиль для забезпечення безпеки. Тому перед вибором використання JWT з OAuth2 рекомендується ретельно розглянути, чи варто здійснювати додаткові зусилля з розробки вашого додатку на користь покращеної продуктивності та ефективності.

В остаточному підсумку вибір між використанням JWT та OAuth2 повинен базуватися на специфікації вашого додатку і його потребах у безпеці та продуктивності.

3.3. Порівняння ефективності та безпеки

Аспекти безпеки в протоколі OAuth2 є важливими для забезпечення захисту як облікових даних користувачів, так і самого процесу авторизації та доступу до ресурсів. Давайте розглянемо докладніше кожен з вказаних аспектів.

Авторизація та делегування доступу, які забезпечуються протоколом OAuth2, є важливими аспектами з точки зору безпеки та конфіденційності. Головна ідея полягає в тому, що користувачі можуть надавати додаткам обмежений доступ до своїх ресурсів без ризику розкриття своїх облікових даних.

Провайдер авторизації (наприклад, Google або Facebook) відповідає за аутентифікацію користувача і видавання токенів доступу додаткам. Основна перевага цього підходу полягає в тому, що облікові дані користувача залишаються на стороні провайдера авторизації, а не передаються додаткам. Це дозволяє уникнути ризику витоку чутливих даних і паролів, оскільки додатки отримують лише обмежений токен доступу.

З точки зору безпеки це важливо, оскільки додатки можуть бути нечесними або не надійними. Якщо користувач передавав свої облікові дані кожному додатку, який він використовує, це створювало б серйозні загрози для його безпеки. Однак завдяки OAuth2, користувачі можуть контролювати, які додатки мають доступ до їхніх ресурсів і можуть скасувати доступ в будь-який момент, не розкриваючи свої облікові дані.

Використання токенів в OAuth2 є ключовим аспектом забезпечення безпеки та контролю доступу. OAuth2 використовує два типи токенів: access_token та refresh_token, кожен з яких відповідає за певні аспекти авторизації та безпеки:

- access_token: Цей токен надає доступ до ресурсів і є одноразовим. Його безпека дуже важлива, оскільки в разі його перехоплення зловмисник може отримати доступ до ресурсів, на які має право користувач. Тому access_token повинен бути доглянутим і захищеним від перехоплення або несанкціонованого використання. Використання HTTPS для передачі токенів і їхнє зберігання в безпечному місці є ключовими практиками безпеки в цьому випадку.
- refresh_token: Цей токен використовується для оновлення access_token без необхідності повторної авторизації користувача. Він також має бути належним чином захищений, оскільки в разі його перехоплення зловмисник може безкінечно оновлювати access_token. Зазвичай refresh_token має довший термін дії, і його обмін з сервером авторизації має відбуватися по безпечному каналу і з використанням належного забезпечення.

Основна перевага використання різних типів токенів полягає в тому, що це дозволяє зберігати більший контроль над безпекою в різних сценаріях. Такий підхід допомагає уникнути несанкціонованого доступу і зберігає конфіденційність даних користувачів, що є ключовим аспектом забезпечення безпеки в OAuth2.

Обмін ідентифікаторами в протоколі OAuth2 − це важливий процес, який дозволяє різним сторонам взаємодіяти між собою без розкриття облікових даних користувачів. Цей обмін ідентифікаторами повинен бути належним чином захищений для забезпечення безпеки процесу авторизації та обміну токенами. Деякі ключові аспекти безпеки в контексті обміну ідентифікаторами включають:

1. Захист від перехоплення: Дані, які передаються під час обміну ідентифікаторами, повинні бути зашифрованими та передаватися через безпечний канал, такий як HTTPS. Це допомагає уникнути перехоплення ідентифікаторів зловмисниками.

2. Валідація ідентифікаторів: Під час обміну ідентифікаторами, сторони повинні правильно валідувати отримані ідентифікатори, щоб переконатися, що вони не були змінені або підроблені. Ця перевірка допомагає попередити атаки на цілісність даних.
3. Захист від фішингу: Система повинна бути налаштована таким чином, щоб запобігти атакам типу фішинг, коли атакуючий намагається обманом отримати доступ до ідентифікаторів. Важливо навчити користувачів розпізнавати легітимні запити авторизації.
4. Забезпечення аутентифікації: Під час обміну ідентифікаторами, система повинна підтвердити автентичність сторін, що беруть участь у процесі. Це допомагає уникнути атак імперсонації та зберігає безпеку системи.
5. Використання стандартів безпеки: Для обміну ідентифікаторами краще використовувати стандартизовані протоколи та методи забезпечення безпеки, які перевірені часом і визнані безпечними.

Забезпечення безпеки обміну ідентифікаторами − це критичний аспект протоколу OAuth2, оскільки невірний обмін може призвести до ризику витоку конфіденційних даних і зловмисних атак.

Усі ці аспекти безпеки грають важливу роль в забезпеченні надійності протоколу OAuth2 та захисту як користувачів, так і системи в цілому від потенційних загроз безпеки. Організації повинні дотримуватися найкращих практик та використовувати належні методи захисту, щоб забезпечити безпеку в рамках протоколу OAuth2.

Аспекти безпеки у контексті JWT є ключовими для забезпечення цілісності і конфіденційності даних, які містяться у токенах. Давайте розглянемо кожен з них докладніше.

Підпис і шифрування − це дві ключові аспекти безпеки, пов'язані з використанням JWT (JSON Web Tokens).

Підпис JWT використовується для перевірки цілісності токену. Коли JWT створюється, до нього додається підпис, який генерується на основі секретного ключа (у випадку симетричного підпису) або приватного ключа (у випадку

асиметричного підпису). При перевірці JWT, отримувач використовує відповідний ключ для перевірки підпису, щоб переконатися, що токен не було змінено після його створення. Це допомагає уникнути атак, які спрямовані на зміну даних в JWT.

Шифрування JWT забезпечує конфіденційність даних внутрішнього навантаження токену. Токен може бути зашифрованим з використанням публічного ключа (у випадку асиметричного шифрування) або секретного ключа (у випадку симетричного шифрування). Таким чином, навантаження токену буде незрозумілим для будь-якого, хто не має відповідного ключа для розшифрування. Це дозволяє передавати чутливі дані, такі як облікові записи користувачів або фінансова інформація, зашифрованими у JWT.

Використання підпису та шифрування робить JWT надійним інструментом для передачі та зберігання чутливих даних. Однак важливо відзначити, що ці механізми також додають обчислювальну складність та можуть збільшувати розмір токену, що може вплинути на продуктивність і мережевий обсяг. Таким чином, при використанні шифрування та підпису важливо розглянути баланс між безпекою та продуктивністю в залежності від конкретних потреб вашого додатку.

Захист від підроблення є важливим аспектом безпеки JWT (JSON Web Tokens) та використовується для забезпечення автентичності і цілісності токенів. Це важливо, оскільки підроблення токенів може призвести до серйозних безпекових проблем, таких як несанкціонований доступ до ресурсів або підробка ідентифікації користувача.

Один із способів досягнення захисту від підроблення в JWT − це використання цифрового підпису. Під час створення JWT підпис обчислюється на основі вмісту токену і секретного ключа (у випадку симетричного підпису) або приватного ключа (у випадку асиметричного підпису). Цей підпис додається до токену і використовується для перевірки його автентичності після передачі.

Однак важливо, щоб ключі для підпису були належним чином збережені та оброблені. Ключі повинні бути дуже сильними і захищеними від несанкціонованого доступу. В ідеальному випадку, секретні ключі мають бути збережені на сервері і недоступними ззовні. Публічні ключі (якщо використовуються асиметричні ключі)

можуть бути розповсюджені, оскільки вони використовуються для перевірки підпису, а не для генерації його.

Після приймання JWT отримувач може використовувати відповідний ключ для перевірки підпису. Якщо підпис вірний, це свідчить про те, що токен не був підроблений або змінений після його створення. У випадку виявлення неправильного підпису JWT відкидається, і це допомагає запобігти використанню підроблених токенів для незаконного доступу.

Важливим аспектом є також регулярне оновлення ключів та вдосконалення методів підпису для забезпечення відповідності сучасним стандартам безпеки.

Захист від атак є ще одним важливим аспектом безпеки при використанні JWT (JSON Web Tokens). Такі токени можуть бути підробленими або викраденими з вимиканням безпеки, що може призвести до різних видів атак. Ось кілька способів, які допомагають захистити JWT від атак:

1. Використання HTTPS: Всі передачі JWT повинні здійснюватися через безпечне з'єднання HTTPS. Це забезпечує конфіденційність та цілісність даних під час їх передачі між клієнтом і сервером. Атаки, такі як перехоплення токенів під час передачі через незахищене HTTP, можуть бути запобіжені використанням HTTPS.
2. Безпечне сховище ключів: Ключі, які використовуються для підпису та/або шифрування JWT, повинні бути збережені в безпечному сховищі на стороні сервера. Недоступність ключів для потенційних зловмисників важлива, оскільки вони дозволяють підписувати та перевіряти токени. Ключі повинні бути належним чином захищені та керовані.
3. Валідація та перевірка підпису: При отриманні JWT, отримувач повинен валідувати токен, перевіряти його підпис і гарантувати, що дані в токені не були змінені після його створення. Тільки валідні і непошкоджені токени повинні бути прийняті і використані.
4. Обмеження терміну дії: JWT має обмежений термін дії (час життя). Після закінчення цього терміну токен автоматично стає недійсним. Це

допомагає запобігти використанню старих токенів, які можуть бути викрадені або підроблені.

5. Передача мінімальних даних: JWT може містити різну інформацію, включаючи додаткові дані про користувача. Проте для забезпечення безпеки рекомендується передавати лише необхідну і мінімальну кількість даних в токені, щоб зменшити ризик витоку конфіденційної інформації.

Ці практики та перевірки допомагають захистити JWT від різних атак і забезпечують безпеку при його використанні в додатках і системах.

Всі ці аспекти допомагають зробити JWT надійним інструментом для автентифікації та авторизації, і вони дозволяють передавати чутливі дані без ризику їхнього витоку або підроблення. Однак важливо дотримуватися належних стандартів безпеки та ретельно налаштовувати систему для забезпечення найвищого рівня захисту.

Порівнюючи OAuth2 та JWT, важливо враховувати їхні специфічні властивості та призначення.

OAuth2 спрямований на делегований доступ до ресурсів та в основному використовується для авторизації і обміну токенами. Він служить чудовим інструментом для забезпечення безпеки, контролю доступу та обміну ресурсами між додатками, при цьому не розкриваючи облікових даних користувачів. OAuth2 є стандартним протоколом для авторизації і використовується багатьма великими провайдерами послуг, такими як Google, Facebook, і Twitter.

З іншого боку, JWT відкриває ширший спектр можливостей. Він дозволяє включати додаткові дані в токен, такі як інформація про користувача та додаткові метадані. Це робить JWT більш гнучким і розширює функціональність, але вимагає додаткового контролю та обережності для забезпечення безпеки. JWT може бути підписаним і зашифрованим, щоб забезпечити цілісність та конфіденційність даних, що робить його корисним для передачі чутливих інформаційних даних між сторонами.

Вибір між OAuth2 та JWT повинен залежати від конкретних потреб проекту та рівня безпеки, який вимагається. Якщо основна мета - забезпечити делегований доступ до ресурсів без розкриття облікових даних користувачів, OAuth2 може бути відмінним вибором. З іншого боку, якщо потрібно передавати додаткову інформацію про користувачів і контролювати їхню ідентифікацію, JWT може бути більш відповідним варіантом.

Також важливо враховувати, що у багатьох випадках може бути корисним поєднувати обидва протоколи − OAuth2 і JWT. Це дозволяє поєднувати переваги обох підходів і забезпечити більший контроль над безпекою та функціональністю додатку. В кінцевому підсумку вибір між цими двома підходами або їх поєднанням повинен бути зроблений з урахуванням конкретних вимог проекту та рівня безпеки, який прагнете досягти.

3.4. Рекомендації щодо вибору алгоритму аутентифікації

Якщо потреби полягають у наданні зовнішнім додаткам доступу до ресурсів, які належать проекту, і при цьому ви бажаєте зберегти облікові дані користувачів в безпеці, то протокол OAuth2 може бути відмінним вибором. Основним перевагою OAuth2 є його здатність делегувати доступ до ресурсів без передачі фактичних облікових даних користувача сторонньому додатку. Замість цього, сторонні додатки отримують токени доступу, які дають їм право взаємодіяти з ресурсами від імені користувача.

OAuth2 використовується широко в інтернет-екосистемі, і він має стандартизовані специфікації, що роблять його добре підходящим для використання в різних сценаріях. Це дає можливість стороннім розробникам легко інтегрувати свої додатки з ресурсами, не змушуючи їх запитувати користувачів про їхні облікові дані. Крім того, OAuth2 забезпечує механізми керування доступом, які дозволяють обмежувати права сторонніх додатків і встановлювати умови для доступу до ресурсів.

Таким чином, якщо основна мета − забезпечити безпеку облікових даних користувачів та надати стороннім додаткам зручний та стандартизований спосіб доступу до ресурсів, OAuth2 може бути оптимальним вибором. Однак важливо також ретельно налаштувати і керувати реалізацією OAuth2 для забезпечення максимальної безпеки.

JWT (JSON Web Tokens) може стати більш відповідним варіантом, коли необхідно передавати додаткову інформацію про користувачів та контролювати їхню ідентифікацію. Ця особливість JWT полягає у тому, що він дозволяє включати корисні дані, такі як інформація про користувача, ролі, дозволи або будь-які інші метадані, безпосередньо в токен.

Завдяки цьому можна покращити функціональність додатку, роблячи JWT більш гнучким і здатним до адаптації під конкретні потреби. Наприклад, можна включити в JWT інформацію про рівень доступу користувача, що дозволить додатку надавати різний функціонал в залежності від прав користувача. Це особливо корисно в системах з різними ролями користувачів або в управлінні доступом до певних ресурсів.

JWT також забезпечує можливість аутентифікації, оскільки токен підписується секретним ключем або публічним/приватним ключем, забезпечуючи підтвердження ідентичності відправника запиту. Це дозволяє перевіряти, що запити надсилаються аутентифікованим користувачем, що додає рівень безпеки до додатків.

Отже, використання JWT може бути вигідним, коли ви прагнете не лише надавати доступ до ресурсів, але і розширювати функціональність додатку, контролювати ідентифікацію та забезпечувати додатковий рівень безпеки. Однак важливо також ретельно керувати і забезпечувати безпеку JWT, оскільки неправильна імплементація може призвести до потенційних уразливостей.

Поєднання протоколів OAuth2 і JWT може бути корисним у випадках, коли потрібно комбінувати переваги обох підходів, щоб забезпечити більший контроль над безпекою та функціональністю додатку.

OAuth2, зорієнтований на делегування доступу до ресурсів без розкриття облікових даних користувача, може бути використаний для забезпечення авторизації сторонніх додатків та надання їм доступу до обмеженого набору ресурсів. В той же час JWT дозволяє включати додаткову інформацію про користувача та розширювати функціональність додатку завдяки корисному навантаженню (payload).

Поєднання цих двох протоколів може вигідно використовуватися для створення більш гнучких та функціональних додатків. Наприклад, сервер автентифікації OAuth2 може видавати токен доступу, який містить JWT з додатковою інформацією про користувача або дозволами. Це дозволяє стороннім додаткам отримувати доступ до ресурсів та одночасно отримувати інформацію про користувача без додаткових запитів.

Крім того, можна використовувати підхід з двома маркерами, коли OAuth2 видаватиме два окремих токени: access_token для доступу до ресурсів і JWT для ідентифікації та передачі додаткових даних. Цей підхід надає більше контролю над доступом користувачів та даними, що передаються між сторонами.

Проте важливо пам'ятати, що поєднання цих двох протоколів може зробити архітектуру додатку складнішою і вимагати додаткового рівня безпеки та управління. Важливо правильно імплементувати і налагодити це поєднання, а також забезпечити безпеку обох протоколів для захисту додатку від потенційних загроз.

Перед вибором алгоритму автентифікації для проекту варто провести ретельний аналіз вимог та особливостей системи. Ключовими аспектами, які варто враховувати, є наступні речі.

Вимоги проекту: Розглянути, які саме функціональність та ресурси необхідно захищати і керувати доступом до них. Які види інформації потрібно передавати між користувачами та додатками. Наприклад, якщо потрібно надавати доступ до конфіденційних даних користувачів, можна віддати перевагу більш безпечним методам автентифікації, таким як OAuth2 з використанням JWT.

Особливості користувачів: Врахувати, хто саме буде використовувати додаток. Якщо є різні типи користувачів, які мають різний рівень доступу та сфери відповідальності, важливо вибрати автентифікаційний метод, який дозволяє керувати доступом та ідентифікувати користувачів точно за їхніми ролями:

- Рівень безпеки: Залежно від чутливості інформації та потенційних загроз безпеці, вибрати алгоритм, який надає відповідний рівень захисту. Якщо обрізкуєте важливі дані, JWT може бути більш підходящим, але вимагатиме додаткових заходів безпеки.
- Спрощення розробки: Розглянути, наскільки легко можна імплементувати обраний алгоритм автентифікації в додатку. Існують готові бібліотеки та інструменти для багатьох алгоритмів, які можуть спростити розробку.

Зробивши цей аналіз і зваживши всі варіанти, ви зможете обрати алгоритм автентифікації, який найкраще відповідає потребам проекту та забезпечує необхідний рівень безпеки та функціональності. Нехай ця обдумана стратегія стане основою для ефективної реалізації аутентифікації у додатку.

Завжди важливо дотримуватися рекомендацій та кращих практик з безпеки при реалізації обраного алгоритму автентифікації. Це має вирішальне значення для забезпечення захисту конфіденційної інформації та запобігання можливим загрозам безпеці. Деякі важливі аспекти безпеки, які слід враховувати:

1. Захист ключів та секретів: Для більшості алгоритмів автентифікації ключі та секрети є критично важливими компонентами. Вони повинні бути зберігатися в безпечному місці і доступ до них має бути обмежений тільки на необхідному рівні.
2. Оновлення та моніторинг: Після реалізації автентифікаційного алгоритму важливо регулярно оновлювати його. Це дозволяє уникнути використання застарілих версій алгоритмів, які можуть містити вразливості. Також необхідно вести моніторинг системи на предмет можливих загроз та несправностей.
3. Захист від атак: Реалізація автентифікації повинна враховувати заходи безпеки, такі як захист від атак на перехоплення сесії, захист від SQL-

ін'єкцій та інших типових атак. Використання стандартизованих бібліотек та рекомендацій з безпеки допоможе запобігти багатьом загрозам.

4. Тестування безпеки: Важливо проводити тестування безпеки автентифікаційного механізму, включаючи тестування на проникнення. Це допоможе виявити можливі вразливості та виправити їх перед тим, як зловмисники зможуть їх використовувати.

Враховуючи ці аспекти безпеки і дотримуючись кращих практик, можна створити надійний і безпечний механізм автентифікації для проекту. Регулярне оновлення та моніторинг допоможуть підтримувати високий рівень безпеки протягом усього життєвого циклу системи.

ВИСНОВКИ

Дослідження алгоритмів аутентифікації за технологіями JWT та OAuth2 виявило, що кожен підхід має свої сильні та слабкі сторони, які повинні бути враховані при розробці веб-додатків. Вибір між JWT та OAuth2 не є однозначним і залежить від специфіки та вимог проекту. Наприклад, JWT може бути більш підходящим для ситуацій, де потрібна висока швидкість обробки та мінімальна залежність від зовнішніх сервісів, тоді як OAuth2 є кращим рішенням для додатків, які вимагають більш складної системи дозволів і взаємодії з різними зовнішніми платформами.

Важливо підкреслити, що вибір конкретної технології має базуватися на конкретних вимогах та контексті проекту. Розробники повинні враховувати такі фактори, як масштабованість, безпека, вартість реалізації, а також очікування та потреби кінцевих користувачів.

Забезпечення безпеки повинно бути пріоритетом у розробці будь-якої системи аутентифікації. Це означає використання надійних методів шифрування, захист від різних видів атак та ретельне тестування системи на вразливості. Розробники повинні бути обізнані щодо потенційних ризиків кожної технології та використовувати кращі практики для мінімізації вразливостей.

Окрім технічних аспектів, не менш важливим є забезпечення хорошого користувацького досвіду. Система аутентифікації має бути інтуїтивно зрозумілою та зручною для користувачів, не створюючи зайвих перешкод при взаємодії з веб-додатком.

Враховуючи швидкий розвиток технологій у сфері веб-розробки, важливо залишатися в курсі останніх трендів та інновацій. Це допоможе вибирати найбільш ефективні та актуальні рішення для конкретних проектів, а також адаптуватися до змінних вимог ринку та користувацьких очікувань. Розробники повинні бути готові до непередбачених викликів та гнучко підходити до вибору технологій, враховуючи не лише поточні, але й майбутні потреби додатку.

У підсумку, вибір між JWT та OAuth2 повинен базуватися на глибокому розумінні їх характеристик та здатності відповідати специфічним вимогам проекту. Важливо забезпечити баланс між безпекою, продуктивністю, гнучкістю та користувацьким досвідом, щоб створити ефективну та надійну систему аутентифікації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Intro to IAM. URL: https://auth0.com/intro-to-iam
2. What is OAuth 2.0. URL: https://auth0.com/intro-to-iam/what-is-oauth-2
3. JWT Authorization: How It Works and Implementing in Your Application. URL: https://frontegg.com/guides/jwt-authorization
4. OAuth vs. JWT: What Is the Difference? Can You Use Them Together. URL: https://frontegg.com/blog/oauth-vs-jwt
5. Authentication in Web Applications. URL: https://www.theseus.fi/bitstream/handle/10024/813453/Wu_I-En.pdf
6. JWT vs. OAuth: Which Is Good for Ultimate Web Security. URL: https://geekflare.com/jwt-vs-oauth/
7. OAuth 2.0 Authorization Framework. URL: https://auth0.com/docs/authenticate/protocols/oauth
8. JWT Framework. URL: https://web-token.spomky-labs.com/
9. Zikai Wang and Wei Sun, “Review of Web Authentication”, Journal of Physics: Conference Serie, 2020.
10. Rahmatulloh A., “Performance comparison of signed algorithms on JSON Web Token”, IOP Conference Series: Materials Science and Engineering, 2019.
11. Jermyn I., Mayer A., Monrose F., Reiter M. and Rubin A., “The design and analysis of graphical password”, USENIX Association, 1999.
12. Blonder G., “Graphical password”, U.S. Patent 5,559,961, 1996.
13. Keith M, Shao B, and Steinbart P, “The usability of passphrases for authentication: An empirical field study”, Int. J. Man-Mach, 2007.
14. Weir M, Aggarwal S, De Medeiros B, “Password cracking using probabilistic context-free grammars”, IEEE. Symp. Security and Privacy, 2009.
15. Yan J, Blackwell A, Anderson R and Grant A, “Password memorability and security: Empirical results”, IEEE Security & privacy, 2004.

16. Biddle R, Chiasson S and Van Oorschot P, “Graphical passwords: Learning from the first twelve years”, J. ACM. CSUR, 2012.
17. Brostoff S, Sasse M, “Are Passfaces more usable than passwords? A field trial investigation People and computers XIV—usability or else”, London Springer, 2000.
18. Silver D, Jana S, Boneh D, “Password managers: Attacks and defenses”, Conf. Security Symp, 2014.
19. M’Raihi D, Machani S, Pei M, “Totp: Time-based one-time password algorithm”, J. Int. RFC, 2011.
20. Zheng F, Rozi A, Wang R and Li L, “Overview of biometric recognition Technology”, CHN. J. Inf. Sec. Res., 2016.
21. Tico M and Kuosmanen P, “Fingerprint matching using an orientation-based minutia descriptor”, J. IEEE. Trans. on Pattern Anal. Mach. Intell., 2003.
22. Turk M A and Pentland A, “Face recognition using eigenfaces”, IEEE. Conf. on Comput Vision., 1991.
23. Ranjan R, Alisherov F and Choi M, “Biometric authentication: A review”, Int. U-E-Serv. Sci. Tech., 2009.
24. Breau J, Belser J E, Doerr R and Rieschick G, “Unique session ID sharing”, U.S. Patent 8,213,423, 2012.
25. Jones M, Bradley J and Sakimura N, “Json web token (jwt)”, J. IETF, 2015.
26. Cooper D, Santesson S, Farrell S, “Public Key Infrastructure Certificate and Certificate Revocation List (CRL)”, RFC5280, 2008.
27. Chadwick D W, Antony S and Bjerk R, “Instant certificate revocation and publication using WebDAV”, Comput. Sec., 2010.
28. Internet Engineering Taskforce (IETF). JSON Web Token (JWT). URL: https://www.rfc-editor.org/rfc/rfc7519
29. OAuth2.0. The Industry-standard Protocol for Authorization. URL: https://oauth.net/2/
30. 10 Passkeys. Safer and Easier Replacement for Passwords. URL: https://www.passkeys.com/

ДОДАТОК А

ДОДАТОК ДЛЯ ДЕМОНСТРАЦІЇ РОБОТИ OAUTH 2.0

```
from flask import Flask, request, redirect, session, url_for
from flask_oauthlib.client import OAuth
import os

app = Flask(__name__)
app.secret_key = os.urandom(24)

# Configuration for OAuth 2.0 with Google
oauth = OAuth(app)

google = oauth.remote_app(
    'google',
    consumer_key='431562869134-
1a72u7sk17u6kb0ghsm0b473bn0mc2m6.apps.googleusercontent.com',
    consumer_secret='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',
    request_token_params={
        'scope': 'email',  # You can specify additional scopes here
    },
    base_url='https://www.googleapis.com/oauth2/v1/',
    request_token_url=None,
    access_token_method='POST',
    access_token_url='https://accounts.google.com/o/oauth2/token',
    authorize_url='https://accounts.google.com/o/oauth2/auth',
)

@app.route('/')
def index():
```

ДОДАТОК А

```
 return redirect(url_for('login'))

@app.route('/login')
def login():
    return google.authorize(callback=url_for('authorized', _external=True))

@app.route('/logout')
def logout():
    session.pop('google_token', None)
    return 'Logged out'

@app.route('/login/authorized')
def authorized():
    response = google.authorized_response()
    if response is None or response.get('access_token') is None:
        return 'Access denied: reason={} error={}'.format(
            request.args['error_reason'],
            request.args['error_description']
        )

    session['google_token'] = (response['access_token'], '')
    user_info = google.get('userinfo')
    return 'Logged in as: ' + user_info.data['email']
@google.tokengetter
def get_google_oauth_token():
    return session.get('google_token')

if __name__ == '__main__':
    app.run(debug=True)
```

ДОДАТОК Б

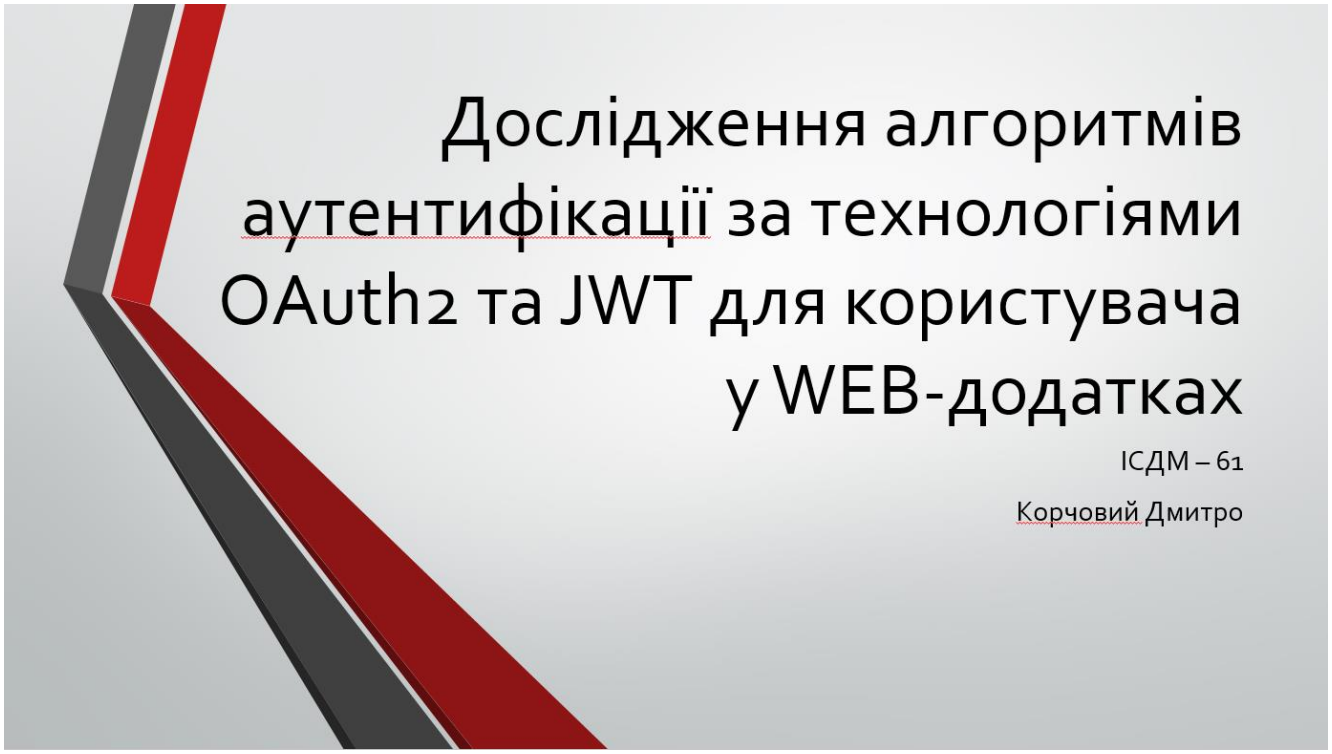
ДОДАТОК ДЛЯ ДЕМОНСТРАЦІЇ РОБОТИ JWT

```
from flask import Flask, request, jsonify
import jwt
import datetime

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'  # Замініть на секретний ключ за вашим вибором

# Функція для генерації JWT-токена
def generate_token(username):
    payload = {
        'username': username,
        'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=1)  # Токен дійсний протягом 1 години
    }
    token = jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')
    return token

# Роут для аутентифікації і отримання JWT-токена
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')

    # Тут ви можете додати логіку для аутентифікації користувача на основі переданих даних
```

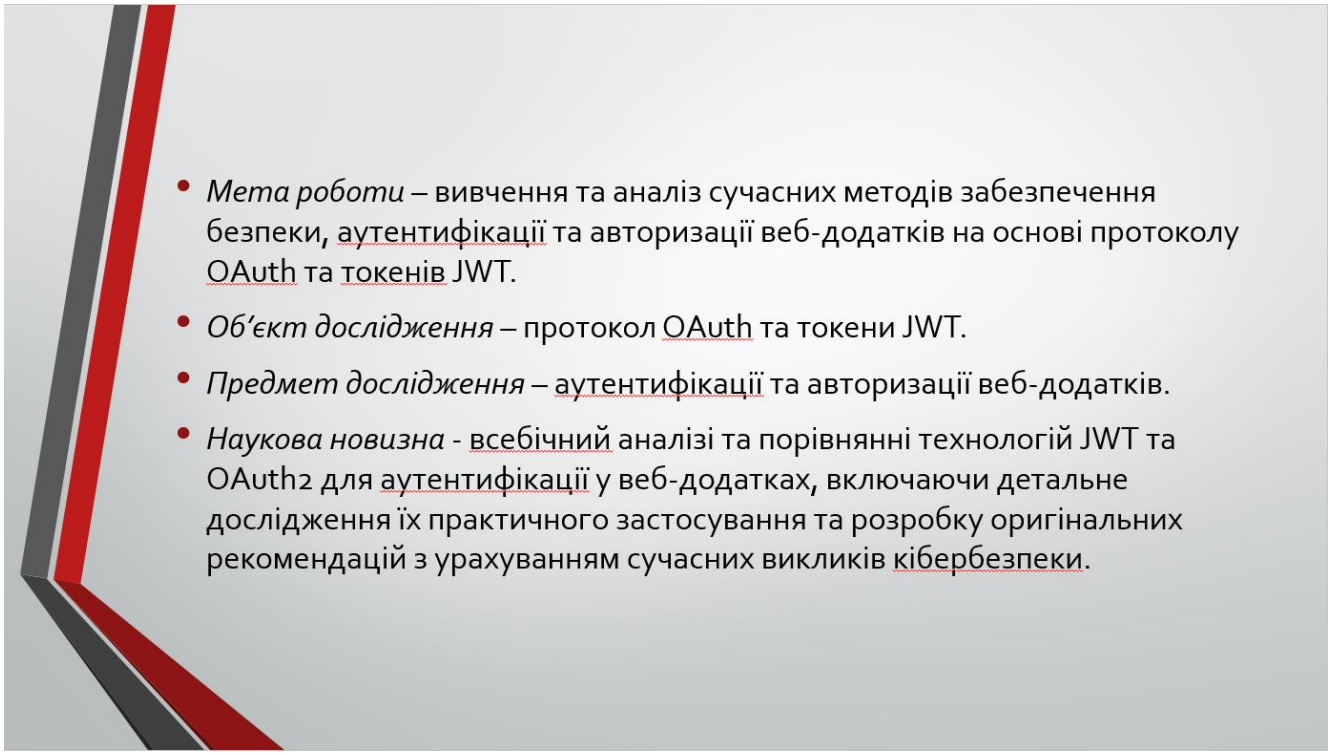
ДОДАТОК Б

```
    # Приклад простої перевірки користувача (перевірка пароля)
    if username == 'your_username' and password == 'your_password':
        token = generate_token(username)
        return jsonify({'access_token': token})
    else:
        return jsonify({'message': 'Authentication failed'}), 401

# Роут для захищених ресурсів, які можуть бути доступні лише з дійсним JWT-токеном
@app.route('/protected', methods=['GET'])
def protected_resource():
    token = request.headers.get('Authorization')

    if not token:
        return jsonify({'message': 'Token is missing'}), 401

    try:
        payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        return jsonify({'message': 'Welcome, ' + payload['username']})
    except jwt.ExpiredSignatureError:
        return jsonify({'message': 'Token has expired'}), 401
    except jwt.InvalidTokenError:
        return jsonify({'message': 'Invalid token'}), 401

if __name__ == '__main__':
    app.run(debug=True)
```

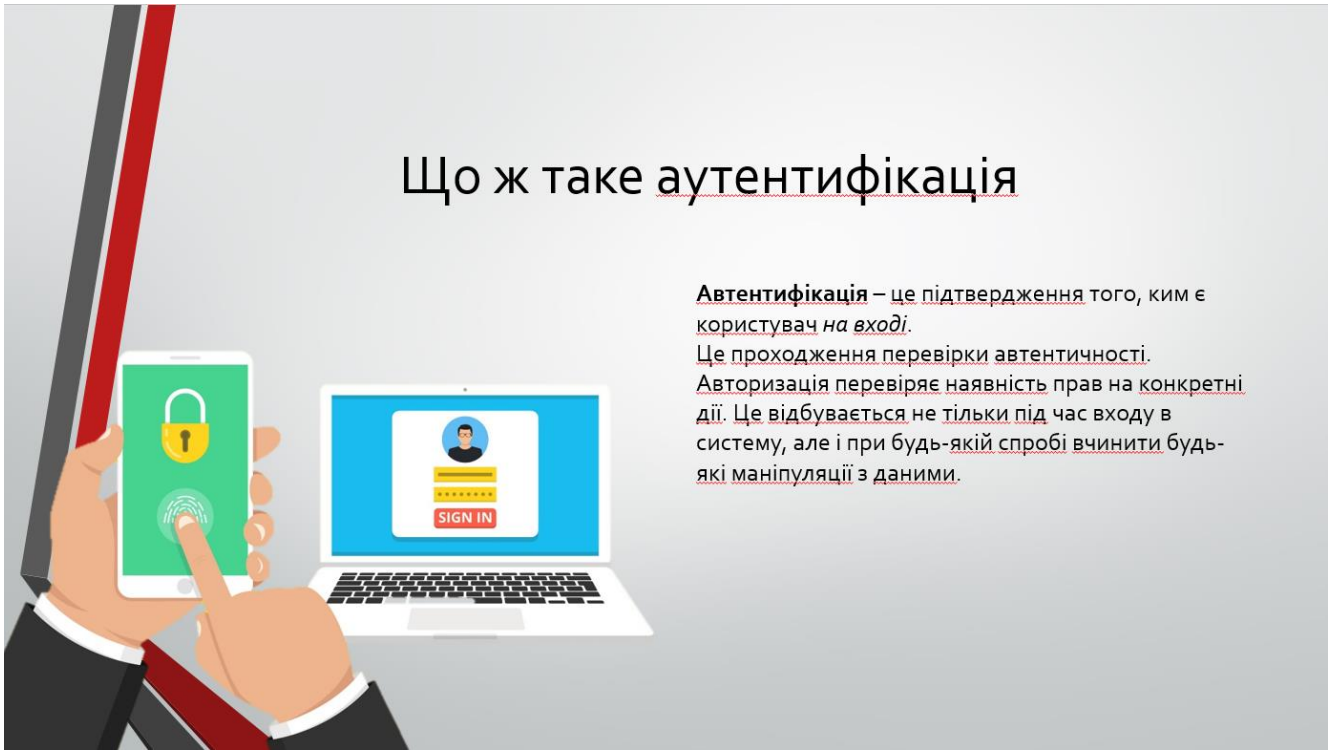
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



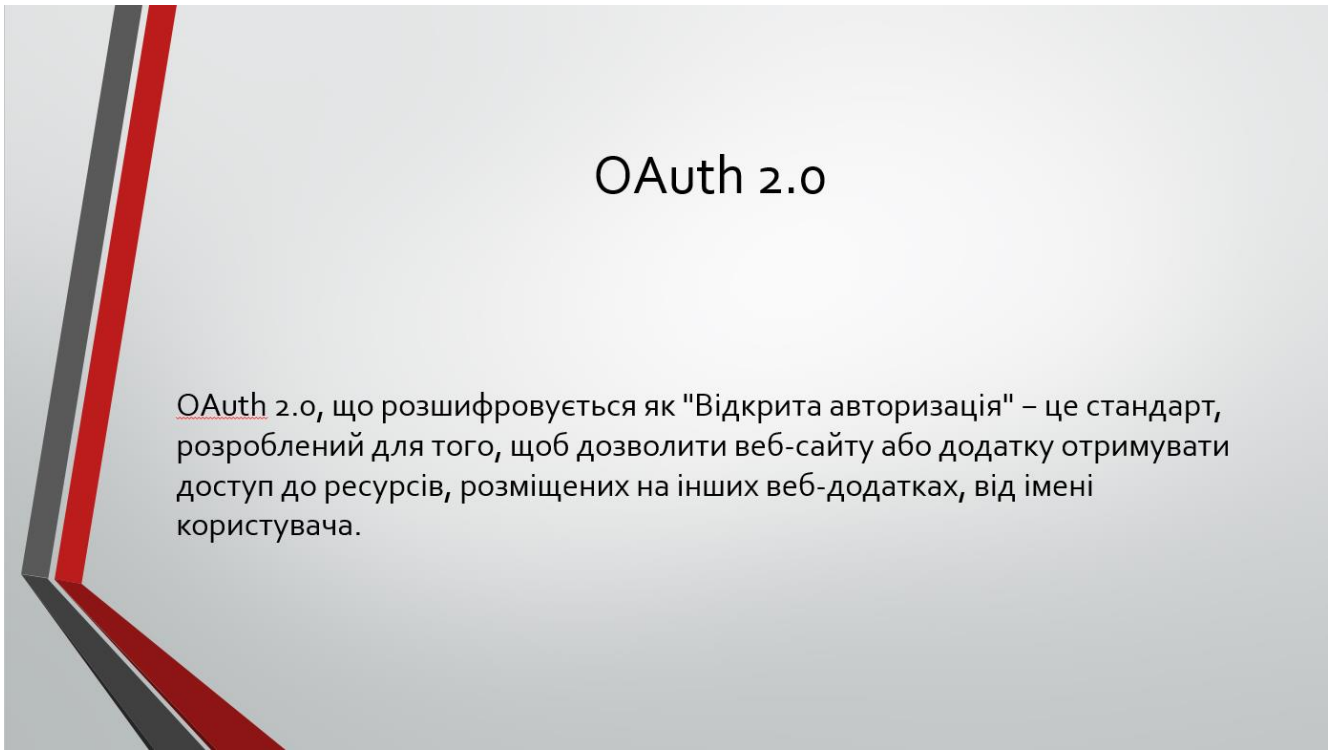
Слайд 1



Слайд 2

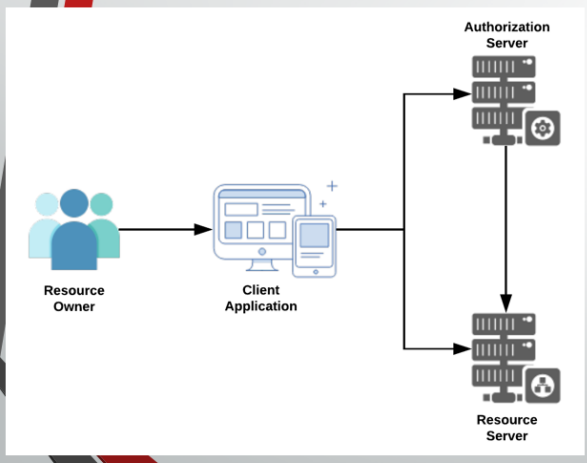


Слайд 3



Слайд 4

Рольова схема OAuth 2.0



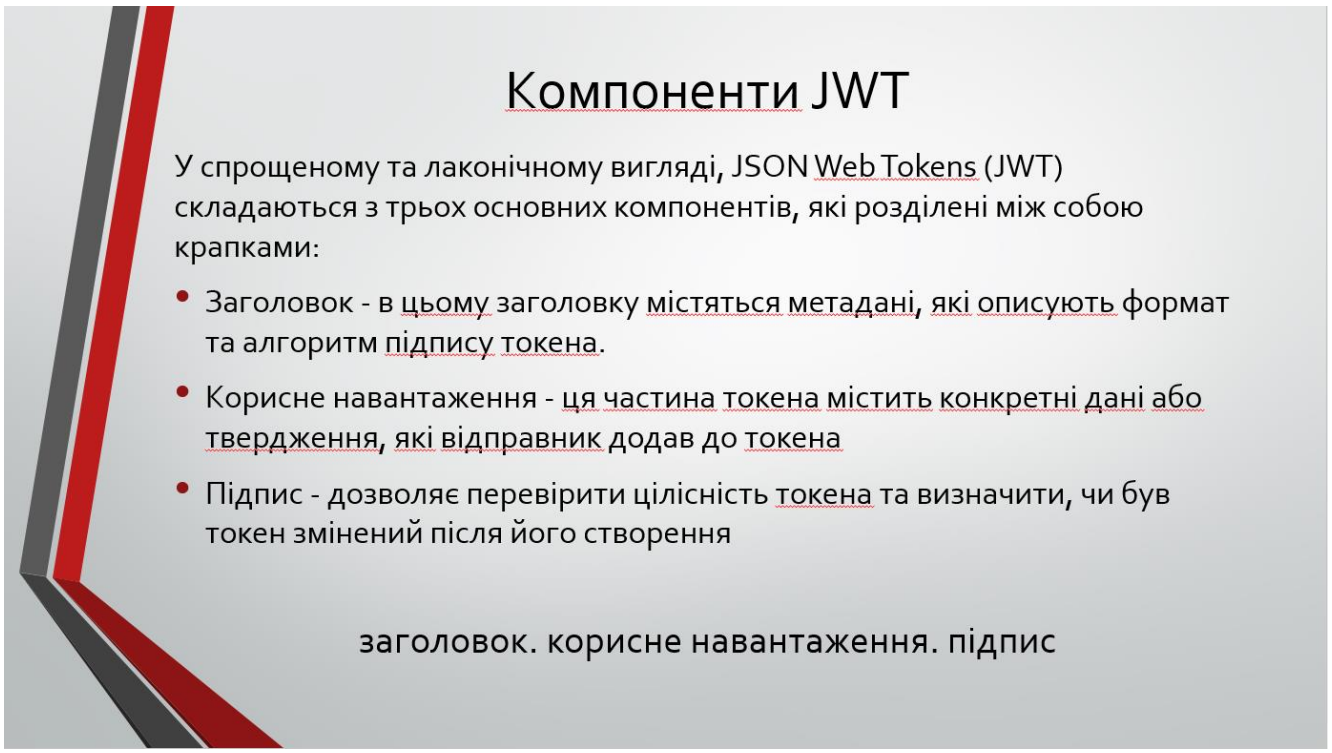
Власник ресурсу: користувач або система, яка володіє захищеними ресурсами і може надавати доступ до них.
Клієнт: Клієнт – це система, якій потрібен доступ до захищених ресурсів. Щоб отримати доступ до ресурсів, клієнт повинен мати відповідний маркер доступу.
Сервер авторизації: Цей сервер отримує запити від клієнта на маркери доступу і видає їх після успішної автентифікації та згоди власника ресурсу. Сервер авторизації має дві кінцеві точки: кінцеву точку авторизації, яка обробляє інтерактивну автентифікацію та згоду користувача, і кінцеву точку токенів, яка бере участь у взаємодії між машинами.
Сервер ресурсів: Сервер, який захищає ресурси користувача і отримує запити на доступ від клієнта. Він приймає і перевіряє маркер доступу від клієнта і повертає йому відповідні ресурси.

Слайд 5

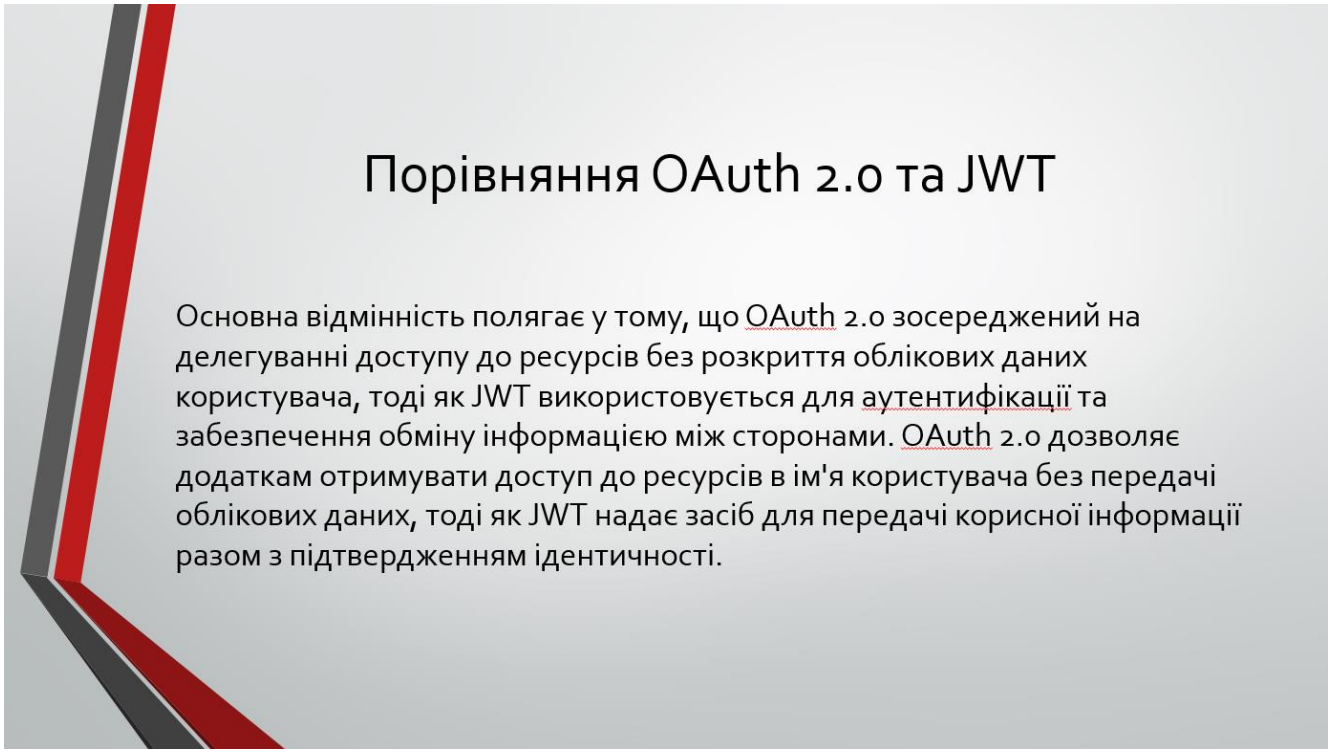
JWT
(JSON Web Token)

представляє собою стандарт, що базується на відкритій специфікації RFC 7519. Цей високофункціональний і водночас надзвичайно компактний інструмент визначає ефективний і надійний спосіб обміну інформацією між різними системами та додатками у форматі JSON-об'єкта. Що особливо важливо, ця передана інформація може бути перевірена і надійно аутентифікована завдяки наявності цифрового підпису.

Слайд 6

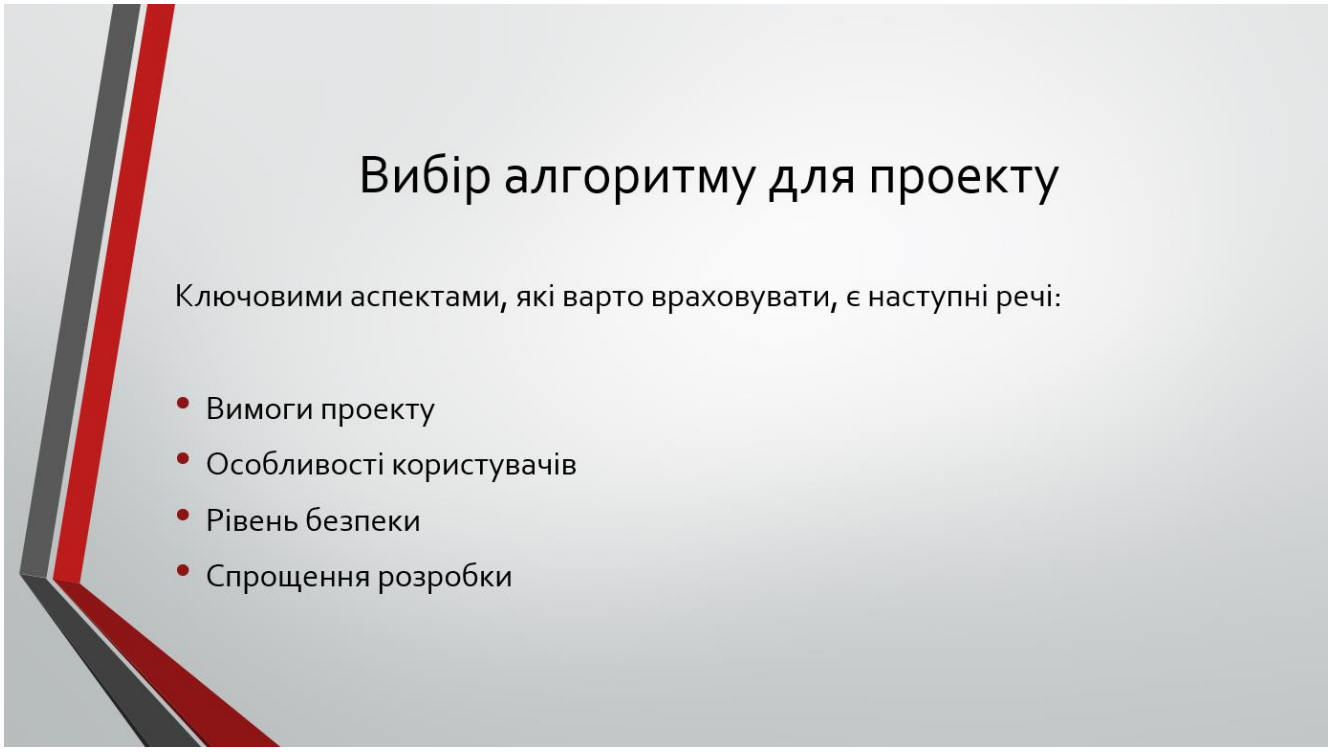


Слайд 7

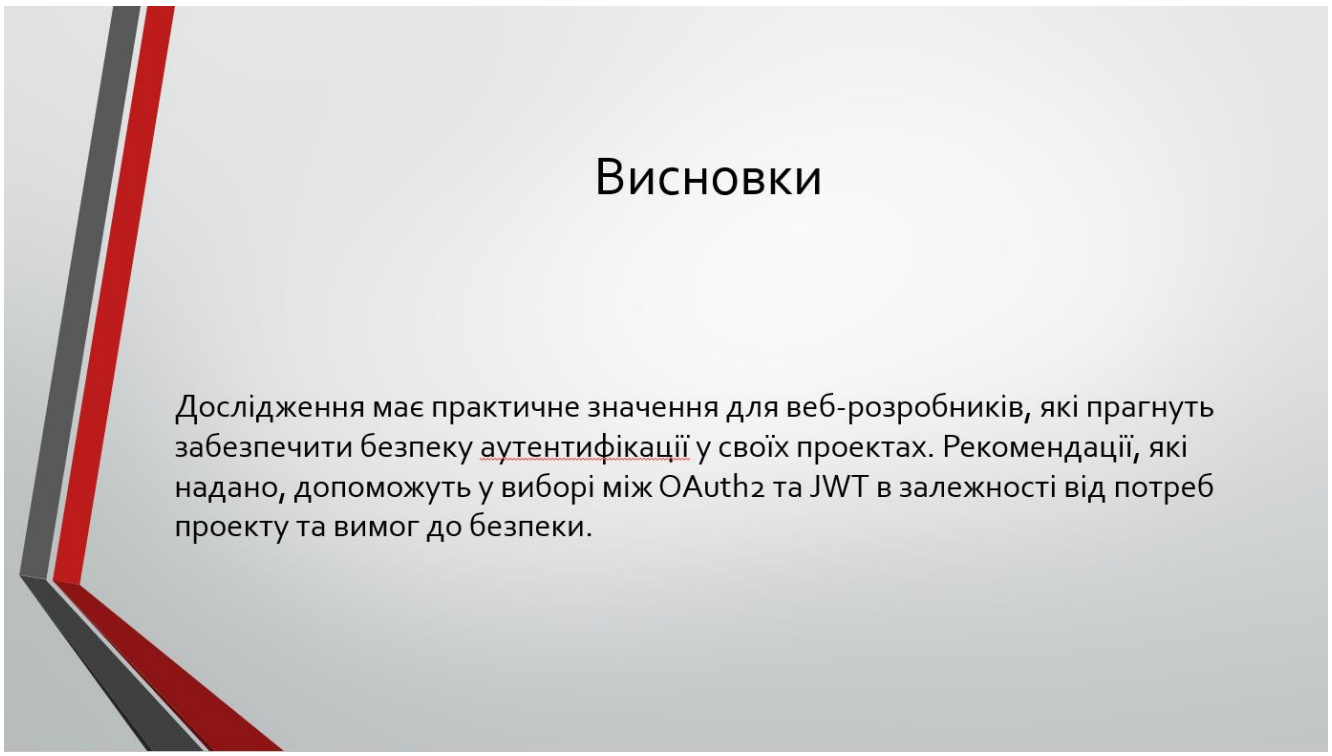


Слайд 8

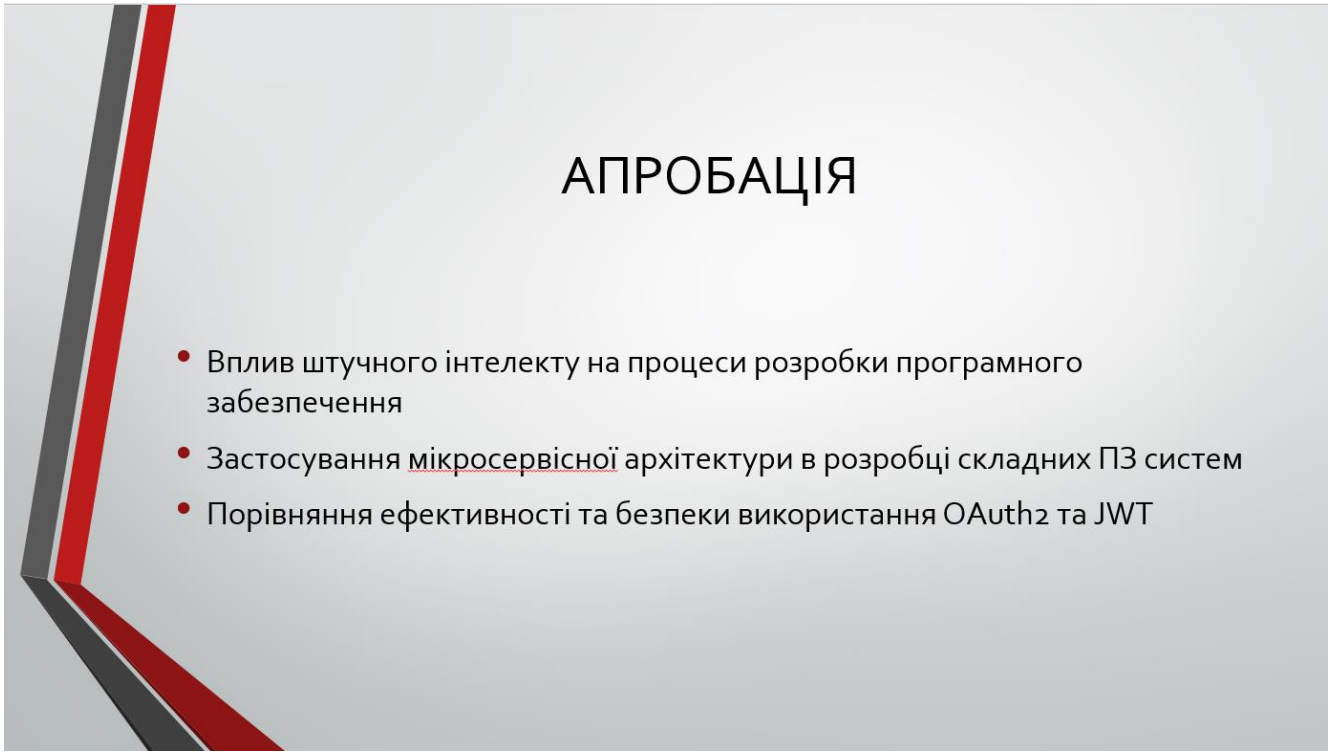
Слайд 9



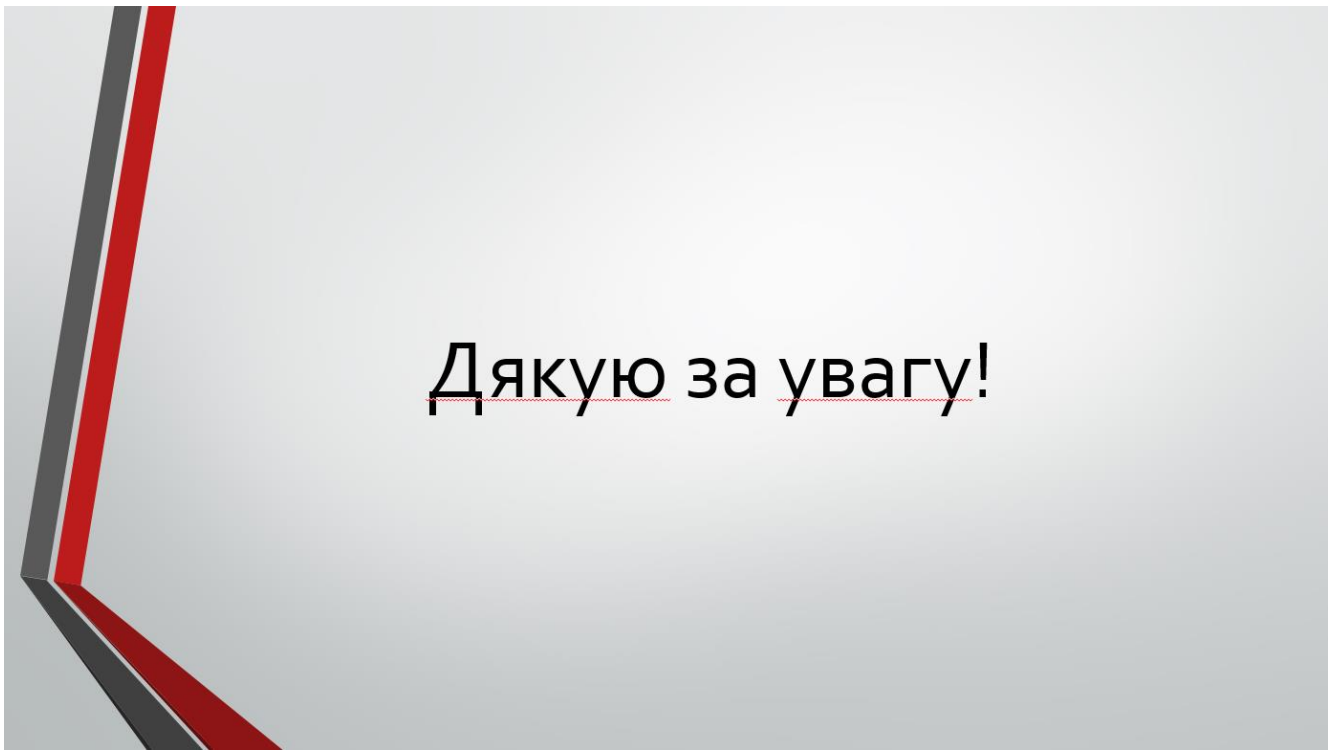
Слайд 10



Слайд 11



Слайд 12



Слайд 13