ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ТЕХНОЛОГІЯ ОРГАНІЗАЦІЇ ЗАХИЩЕНОГО ОБМІНУ ДАНИМИ ВІДДАЛЕНИХ КОРИСТУВАЧІВ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОРГАНІЗАЦІЇ НА БАЗІ VPN»

на здобуття освітнього ступеня магістра
зі спеціальності ____125 Кібербезпека_____
(код, найменування спеціальності)
освітньо-професійної програми _Інформаційна та кібернетична безпека
(назва)

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

Виконав: здобувач(ка) вищої освіти групи БСДМ-62
ТЕРЕПА Іван
(ПРІЗВИЩЕ, Ім'я)

Керівник: БОРСУКОВСЬКИЙ Юрій
к.т.н., доцент (ПРІЗВИЩЕ, Ім'я)

Рецензент: ТУРОВСЬКИЙ Олександр
д.т.н., професор (ПРІЗВИЩЕ, Ім'я)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

Кафедра Інформаційної та кібернетичної безпеки
Ступінь вищої освіти Магістр
Спеціальність 125 Кібербезпека
Освітньо-професійна програма Інформаційна та кібернетична безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
_____ Галина ГАЙДУР
“___”_____ 2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Терепі Івану Романовичу
(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи:
«Технологія авторизації та автентифікації веб-додатків на базі протоколу OAuth 2.0»
керівник кваліфікаційної роботи: БОРСУКОВСЬКИЙ Юрій, к.т.н., доцент,
(ПРІЗВИЩЕ, Ім’я, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023р. №145.

2. Строк подання студентом кваліфікаційної роботи: 15.12.2023 р.

3. Вихідні дані до кваліфікаційної роботи:
інформаційна система організації;
технологія авторизації та автентифікації OAuth 2.0;
наукова та технічна література, експлуатаційна документація, нормативні документи, міжнародні стандарти.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз необхідності автентифікації користувачів на сайтах та веб додатках.
2. Методи та засоби автентифікації користувачів.
3. Розроблення варіанта технології автентифікації користувачів на базі технології OAuth 2.0.

5. Перелік ілюстративного матеріалу:
Презентація PowerPoint

6. Дата видачі завдання 19.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ зп	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Дослідження актуальності проблеми управління привілеями в інформаційній системі організації	19.10.2023 р.	
2.	Аналіз наукової та технічної літератури за темою кваліфікаційної роботи.	22.10.2023 р.	
3.	Аналіз необхідності автентифікації користувачів на сайтах та веб додатках.	27.10. 2023р.	
4.	Методи та засоби автентифікації користувачів.	03.11.2023 р.	
5.	Розроблення варіанта технології автентифікації користувачів на базі технології OAuth 2.0.	15.11.2023 р.	
6.	Оформлення результатів дослідження. Проходження плагіату	26.11.2023 р.	
7.	Підготовка доповіді до захисту.	15.12.2023 р.	

Здобувач(ка) вищої освіти _____ Іван ТЕРЕПА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Юрій БОРСУКОВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ
ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Терепа І.Р. до захисту кваліфікаційної роботи
(прізвище та ініціали)

За спеціальністю 125 Кібербезпека
освітньо-професійної програми Інформаційна та кібернетична безпека
(шифр і назва спеціальності)

на тему: «Технологія організації захищеного обміну даними віддалених користувачів інформаційної системи організації на базі VPN».

Кваліфікаційна робота і рецензія додаються.

Директор інституту _____ Віталій САВЧЕНКО
(*підпис*) (*Ім'я, ПРІЗВИЩЕ*)

Висновок керівника кваліфікаційної роботи

Здобувач ***ТЕРЕПА Іван*** обрав тему роботи, метою якої було дослідити зміст технології контролю доступу до мережі організацій. Перелік використаних джерел свідчить про вміння здобувача розбиратись в наукових питаннях та застосовувати їх при дослідженнях. Під час виконання кваліфікаційної роботи ***ТЕРЕПА Іван*** показав гарну теоретичну та практичну підготовку, вміння самостійно вирішувати питання і робити висновки. Роботу виконував сумлінно, акуратно та вчасно за планом.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача ***ТЕРЕПА Іван*** на оцінку **«добре»** та присвоїти йому кваліфікацію магістр з кібербезпеки за спеціалізацією інформаційна та кібернетична безпека.

Керівник кваліфікаційної роботи _____ Борсуковський Ю.В.
(підпис) *(Ім'я, ПРІЗВИЩЕ)*
"____" _____ 2023 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач(ка) ***ТЕРЕПА Іван.*** допускається до захисту даної роботи в Державній екзаменаційній комісії.

Завідувач кафедри Інформаційної та кібернетичної безпеки
(назва)

_____ Галина ГАЙДУР
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА

на кваліфікаційну роботу

здобувача Терепи Івана

на тему: «Технологія авторизації та автентифікації веб-додатків на базі протоколу OAuth 2.0».

Актуальність:

За останні десять років практика використання протоколів авторизації у веб-додатках значно поширилася. Більшість онлайн-сервісів потребує ідентифікації користувачів у своїх системах, щоб регулювати доступ до приватних ресурсів та отримувати інформацію про користувачів. Незалежно від того, чи маємо справу з веб-додатком чи веб-сайтом, існують різні підходи для реалізації процедури авторизації. Деякі протоколи призначені для використання на веб-порталах, інші - у мобільних додатках. Головною метою дослідження методів авторизації є оцінка рівня безпеки інформаційної системи організації з урахуванням її подальшого розвитку. Тому тема кваліфікаційної роботи є актуальною та своєчасною.

Позитивні сторони:

На підставі проведеного аналізу роботи виявлено суть проблеми забезпечення автентифікації та авторизації до веб-сайтів та веб-додатків.

Досліджено різноманітні методи та технології авторизації та автентифікації веб-сервісів.

Запропоновано варіант технології авторизації та автентифікації веб-сайтів та веб-додатків за допомогою технології OAuth 2.0.

Текст представлений досить вмотивовано та логічно, використовуючи грамотну мову. Висновки чіткі та важливі. Графічний матеріал виглядає якісно. Список використаної літератури свідчить про вміння дослідника користуватись науковими та технічними ресурсами відповідно до теми магістерської роботи..

Недоліки:

У кваліфікаційній роботі доцільно було б більш детально описати різні технології авторизації.

Запропонований варіант технології автентифікації на конкретному прикладі або підприємстві.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної роботи

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної роботи

Висновок: Враховуючи недоліки, кваліфікаційна робота заслуговує оцінку «**добре**», а здобувач **ТЕРЕПА Іван** - присвоєння кваліфікації магістр з кібербезпеки за спеціалізацією інформаційна та кібернетична безпека.

Рецензент: _____ Олександр ТУРОВСЬКИЙ

підпис *Ім'я, ПРІЗВИЩЕ*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи и на здобуття освітнього ступеня магістра: 79 сторінок, 33 рисунків, 26 джерел.

Об'єкт дослідження – Технології автентифікації.

Предмет дослідження – OAuth 2.0.

Мета роботи – автентифікація користувачів веб сайтів та веб додатків.

Методи дослідження – Дослідження протоколів, функцій та проблеми методів автентифікації;

Дослідження та аналіз сучасних методів автентифікації та авторизації.

В роботі проведено аналіз проблеми сучасних технологій автентифікації та авторизації, а також загроз, пов'язаних з втратою, модифікацією та розкриттям даних, що направляються користувачам.

Досліджено протоколи, методи та технології авторизації та автентифікації.

Запропоновано варіант додатку з використанням технології OAuth 2.0 на базі додатку KeyCloak.

Галузь використання – кібербезпека веб-сайтів та веб-додатків.

АВТОРИЗАЦІЯ, АВТЕНТИФІКАЦІЯ, ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ, КІБЕРАТАКИ, OAUTH 2.0

ABSTRACT

The text part of the qualification work for the master's degree: 79 pages, 33 figures, 26 sources.

The object of research is Authentication technologies.
The subject of research is OAuth 2.0.
The purpose of the work is to authenticate users of websites and web applications.
Research methods – Research of protocols, functions and problems of authentication methods;
Research and analysis of modern authentication and authorization methods.

The paper analyzes the problem of modern authentication and authorization technologies, as well as threats related to the loss, modification and disclosure of data sent to users.

Protocols, methods and technologies of authorization and authentication have been studied.

A version of the application using OAuth 2.0 technology based on the KeyCloak application is proposed.
The field of use is the cyber security of websites and web applications.

AUTHORIZATION, AUTHENTICATION, AUTHENTICATION PROTOCOLS, CYBER ATTACKS, OAUTH 2.0

ЗМІСТ

СПИСОК ВИКОРИСТАНИХ СКОРОЧЕНЬ

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

HTTPS – HyperText Transfer Protocol Secure;

REST – Representational State Transfer;

SOAP – Simple Object Access Protocol;

TLS – Transport Layer Security;

URL – Uniform Resource Locator;

XML – eXtensible Markup Language;

XSS – Cross Site Scripting;

ІБ – інформаційна безпека;

ІС – інформаційна система;

ВСТУП

За останні десять років практика використання протоколів авторизації у веб-додатках значно поширилася. Більшість онлайн-сервісів потребує ідентифікації користувачів у своїх системах, щоб регулювати доступ до приватних ресурсів та отримувати інформацію про користувачів. З цим зростанням використання авторизаційних протоколів, таких як OAuth 1.0, OAuth 2.0, та OpenID, зросло і зацікавлення у віддаленій авторизації. За допомогою цих протоколів авторизації акредитовані сервіси надають доступ до своїх ресурсів і надають ідентифікацію своїх користувачів іншим системам. Наприклад, для реєстрації на форумі вже не потрібно створювати новий обліковий запис та вводити свій логін і пароль, оскільки можна використовувати свої особисті дані від сервісів, таких як Google або Facebook, і авторизуватися на форумі [1].

Незалежно від того, чи маємо справу з веб-додатком чи веб-сайтом, існують різні підходи для реалізації процедури авторизації. Деякі протоколи призначені для використання на веб-порталах, інші - у мобільних додатках. Головною метою дослідження методів авторизації є оцінка рівня безпеки інформаційної системи організації з урахуванням її подальшого розвитку.

Для аналізу алгоритмів авторизації веб-ресурсів важливо дотримуватися певних стандартів і методологій. Особливо корисними за кордоном є RFC 6000, зокрема RFC 6749, RFC 6750 та RFC 6819. Україна також поступово переходить до використання процесного підходу та цінностей, які визначені в RFC 6000 в сфері інформаційної безпеки.

Метою цієї роботи є дослідження механізмів захисту протоколу OAuth 2.0 шляхом підпису даних електронними цифровими підписами та використання асиметричного шифрування для даних користувачів. Розробка сервісу з використанням цього удосконаленого алгоритму авторизації виконується відповідно до міжнародних стандартів для тестування веб-ресурсів.

Додаткові завдання включають в себе наступне:

- Аналіз результатів і оцінка використаних та досліджених підходів з метою визначення переваг і недоліків створеного рішення.

- Розробка рекомендацій для удосконалення алгоритму роботи існуючих рішень.

- Застосування цих рекомендацій до процесу роботи алгоритмів для використання їх у практичній частині роботи.

- Аналіз отриманих результатів і їх порівняння з попередніми даними.

Об'єктом дослідження є процес роботи протоколу авторизації користувачів OAuth 2.0, тоді як предметом дослідження є механізми захисту цього протоколу.

Для здійснення курсової роботи використовувалися такі методи:

- Порівняння - для визначення різниці між різними підходами та рішеннями.

- Аналіз - для ретельного розгляду і вивчення різних аспектів авторизації користувачів.

Загалом, в сучасних умовах важливо постійно вдосконалювати та перевіряти існуючі рішення щодо авторизації користувачів, оскільки це може бути вирішальним фактором для бізнесу або діяльності організації .

РОЗДІЛ 1: ПРОТОКОЛИ АВТОРИЗАЦІЇ ТА АВТЕНТИФІКАЦІЇ, ЇХ ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА.

1.1 Основні поняття

Сучасний світ інформаційних технологій використовує багато способів для ідентифікації, автентифікації та авторизації користувачів у системах. Важливо спершу ознайомитися з основною термінологією:

Ідентифікація - це процедура розпізнавання користувача в системі, яка, зазвичай, використовує наперед визначений ідентифікатор, такий як логін або ім'я, або іншу інформацію, що її система визнає як унікальну для користувача.

Автентифікація - це процедура встановлення належності користувача до системи, на основі пред'явленого ним ідентифікатора. Це включає в себе перевірку того, що суб'єкт дійсно той, за кого він себе видає. Наприклад, вимагається надати правильний пароль, щоб підтвердити ідентифікацію.

Авторизація - це перевірка, чи має користувач дозвіл на доступ до конкретного ресурсу або функціоналу в системі. Це визначає, чи може користувач виконати певну дію в системі [2].

Ці терміни можуть бути аналогічними до ситуації в реальному житті, де, наприклад, вам потрібно бути ідентифікованим (пред'явити ваше ім'я та прізвище), автентифікованим (показати паспорт і підтвердити свою особу), і авторизованим (перевірити, чи ваше ім'я знаходиться у списку гостей), щоб отримати доступ до закритого клубу.

У комп'ютерних системах ці терміни використовуються для забезпечення безпеки, контролю доступу та ідентифікації користувачів.

Існують різні підходи до автентифікації користувачів:

– автентифікація по паролю;

– автентифікація по сертифікатам;

– автентифікація по ключам доступу;

– автентифікація по токенах.

1.2 Види автентифікації

1.2.1 Однофакторна та багатофакторна автентифікація

Однофакторна автентифікація (SFA) - це метод перевірки ідентичності, який вимагає від користувача (це може бути людина, програмне забезпечення або комп'ютер) пред'явити лише один ідентифікатор, який пов'язаний з його ідентичністю. Спрощено кажучи, це вимагає одного способу підтвердження ідентичності користувача.

Найпоширенішим прикладом однофакторної автентифікації є використання паролю. Інші способи однофакторної автентифікації включають в себе отримання SMS-коду на зареєстрований мобільний пристрій, використання одноразового пароля (OTP), який генерується фізичним пристроєм або програмним забезпеченням на мобільному пристрої або комп'ютері.

На відміну від однофакторної автентифікації, багатофакторна автентифікація (MFA) використовує кілька різних факторів для підтвердження особи і надання доступу до різних систем, програм або даних. Зазвичай системи MFA використовують два або більше з наступних інструментів для перевірки ідентичності:

1. Що ви знаєте: Наприклад, це може бути пароль, особистий ідентифікаційний номер або запитання щодо відновлення.

2. Що у вас є: Це може включати смарт-карти, токени FIDO, одноразові паролі (OTP), пристрої Bluetooth, пристрої, які підключаються через Apple Watch або інші автентифікатори.

3. Хто ви: Цей фактор включає в себе біометричну ідентифікацію, таку як відбиток пальця або розпізнавання обличчя.

Багатофакторна автентифікація дозволяє підвищити рівень безпеки, оскільки потрібно успішно пройти кілька етапів перевірки ідентичності, щоб отримати доступ до системи чи даних.

Багатофакторна автентифікація (MFA) дійсно має численні переваги у плані забезпечення безпеки входу користувачів. Поєднання різних факторів ідентифікації може значно підвищити рівень безпеки і зменшити ризик злому даних та програмного забезпечення. Ось деякі з основних переваг MFA:

1. Захист від негативних наслідків втрати одного фактора: Якщо один із факторів ідентифікації (наприклад, пароль) стає відомим зловмисникам, інші фактори (такі як фізичний токен або біометричні дані) все ще залишаються на захисті.

2. Складність злому: Злом MFA вимагає значно більше зусиль і ресурсів, оскільки зловмисник повинен обійти кілька різних методів ідентифікації.

3. Зменшення ризику фішингу та перехоплення паролів: MFA може захистити користувачів від атак фішингу та перехоплення паролів, оскільки пароль сам по собі не дає доступу до системи без інших факторів.

Однак, MFA може бути деякою складністю для користувачів. Налаштування додаткових факторів та їх використання може вимагати додаткових зусиль та збити деяких користувачів. Наприклад, налаштування мобільного телефону для отримання одноразового пароля може бути нав'язуванням для деяких користувачів.

Незважаючи на ці перешкоди, багатофакторна автентифікація залишається дієвим засобом захисту власних ресурсів для багатьох організацій і користувачів, оскільки вона надійно захищає доступ до важливих даних та систем [3].

1.2.2 Апаратні засоби автентифікації

Принцип автентифікації ґрунтується на використанні спеціальних електронних ключів для визначення особистості користувача. Ці ключі перебувають в ексклюзивному користуванні користувача і використовуються для ідентифікації.

Існують два основних типи пристроїв для цього типу ідентифікації:

1. Карти: Карти використовуються для ідентифікації користувача. Їх існує багато видів, і вони працюють за різними принципами. Наприклад, безконтактні карти (проксіміті-карти) дозволяють користувачам проходити ідентифікацію, не

контактуючи з пристроєм, і використовуються як у комп'ютерних системах, так і у системах доступу в приміщення. Смарт-карти є одними з найбільш надійних ідентифікаторів і схожі на звичайні банківські карти. Також існують менш стійкі до злому карти, такі як магнітні карти або карти з штрих-кодом.

2. Токени: Токени - це інша категорія пристроїв для апаратної ідентифікації. Вони мають власну захищену пам'ять і підключаються безпосередньо до портів комп'ютера, таких як USB або LPT. Токени можуть бути фізичними пристроями або програмними рішеннями, що емулюють фізичний пристрій.

Ці методи ідентифікації застосовуються для забезпечення безпеки та обмеження доступу до ресурсів, і вони можуть бути особливо корисними в ситуаціях, де важливо точно визначити особистість користувача.

Використання апаратної ідентифікації має значні переваги і недоліки:

Переваги:

1. Надійність: Апаратна ідентифікація забезпечує високий рівень безпеки, оскільки ключі та інші ідентифікаційні дані можуть бути збережені в безпечній пам'яті пристрою. Це робить їх важкодоступними для несанкціонованого доступу або злому.

2. Захисні механізми: Токени та електронні ключі можуть бути оснащені різними захисними механізмами, такими як шифрування та підписи, що підвищують рівень безпеки.

3. Мікропроцесор: Вбудований мікропроцесор дозволяє електронному ключу виконувати додаткові функції, що підвищують його корисність, наприклад, підписувати документи або взаємодіяти з іншими системами.

Недоліки:

1. Висока ціна: Апаратна ідентифікація може бути дорогим рішенням, особливо при впровадженні для багатьох користувачів. Вартість самого обладнання і програмного забезпечення може бути значною.

2. Споживання ресурсів: Деякі токени вимагають спеціального програмного забезпечення та ресурсів для їх роботи, що може створити додаткове навантаження для організації.

3. Ризик втрати: Токени і ключі можуть бути загублені, вкрадені або пошкоджені, що створює проблеми для користувачів та адміністраторів системи.

З усіма своїми перевагами і недоліками, апаратна ідентифікація залишається ефективним інструментом для забезпечення безпеки та обмеження доступу до ресурсів, і її вибір залежить від конкретних потреб та обставин організації [4].

1.3 Класифікація протоколів авторизації.

1.3.1 OpenId

OpenID - це відкритий стандарт для децентралізованої системи аутентифікації, яка надає користувачам можливість створити єдиний обліковий запис для доступу до різних веб-ресурсів. Користувачам не потрібно запам'ятовувати окремі дані для авторизації на різних сайтах, адже їхні облікові записи об'єднуються під ідентифікатором OpenID, який вони зареєстровані на сайті "провайдера ідентифікації" OpenID. Оскільки OpenID - це децентралізована технологія, будь-який сайт може використовувати OpenID для авторизації користувачів, і не потрібно залежити від централізованого ресурсу для підтвердження ідентичності користувача.

Особливі риси системи OpenID включають:

- Децентралізованість: система не потребує центрального органу або організації для реєстрації та використання облікових записів OpenID. Користувачі можуть вибирати провайдера OpenID і змінювати його, якщо потрібно.
- Простота використання: OpenID не вимагає від користувачів встановлювати додаткове програмне забезпечення або використовувати специфічні браузери. Всі комунікації здійснюються через стандартні HTTP (S) запити і відповіді, і не потребує використання cookies або інших механізмів керування сесією.

- Гнучкість: OpenID може бути використаний без вимог до JavaScript або сучасних браузерів, але також може бути інтегрованим з AJAX-застосунками. Користувачі можуть пройти аутентифікацію на сайтах, не залишаючи поточну сторінку, і комунікація з провайдером OpenID може відбуватися фоново. OpenID дозволяє використовувати різноманітні розширення, але вони не є обов'язковими для використання стандарту.

Алгоритм роботи OpenID включає такі кроки:

1. Користувач ініціює процес аутентифікації на веб-сервісі та вводить попередній ідентифікатор.

2. З попереднього ідентифікатора визначається URL OpenID-провайдера, який використовує користувач. Іноді ідентифікатор може містити лише ідентифікатор провайдера, і тоді користувач вказує свій ідентифікатор під час взаємодії з провайдером.

3. Опціонально, веб-сервіс і OpenID-провайдер можуть створити спільний секретний ключ для коду аутентифікаційних повідомлень, використовуючи протокол Діффі-Хеллмана. Цей ключ дозволяє веб-сервісу підтверджувати повідомлення від провайдера без додаткових запитів для перевірки їх дійсності.

4. У режимі "checkid_setup," веб-сервіс перенаправляє користувача на сайт OpenID-провайдера для подальшої аутентифікації. У режимі "checkid_immediate," комунікація між браузером і провайдером відбувається непомітно для користувача.

5. OpenID-провайдер перевіряє, чи користувач автентифікований на його стороні та чи бажає він авторизуватися на веб-сервісі. Специфікація OpenID не визначає процес аутентифікації користувача на стороні провайдера.

6. Провайдер перенаправляє браузер користувача назад на веб-сервіс, передаючи результат аутентифікації.

7. Веб-сервіс перевіряє інформацію, отриману від провайдера, враховуючи повернений URL, інформацію про користувача, одноразову мітку (nonce) та підпис повідомлення. Якщо був створений спільний секретний ключ на кроці 3, то

перевірка проводиться з його допомогою. У протилежному випадку, веб-сервіс відправляє провайдеру додатковий запит (check_authentication) для перевірки дійсності. У першому випадку, веб-сервіс вважається "stateless" (без пам'яті), у другому - "dumb" (німим) [5].

1.3.2 BrowserId

BrowserID - це відкритий стандарт для децентралізованої системи авторизації користувачів на веб-ресурсах та мобільних додатках. Він дозволяє користувачам входити на сайти, використовуючи свою адресу електронної пошти як унікальний ідентифікатор. Для користувачів BrowserID вимагає, щоб їх імена користувачів були адресами електронної пошти.

Основні переваги BrowserID включають:

1. Позбавлення користувачів веб-ресурсів від необхідності створення та пам'ятання паролів.
2. Можливість зареєструватися один раз у системі BrowserID і використовувати цей ідентифікатор для входу на різні сайти без потреби у реєстрації на кожному з них.

Процес авторизації BrowserID включає наступні елементи:

1. Користувач - особа, яка використовує протокол BrowserID для входу на сайт.
2. Перевіряюча сторона - веб-сайт, який надає можливість входу через протокол авторизації BrowserID.
3. Постачальник ідентифікації - домен, який видає ідентифікаційні сертифікати сумісні з Persona для своїх користувачів. Зазвичай, це поштові сервіси, які надають адреси електронної пошти як ідентифікатори.

Алгоритм роботи BrowserID включає наступні етапи:

1. Надання сертифіката користувачу.

2. Генерація ствердження.

3. Верифікація ствердження.

Для того, щоб адреса електронної пошти могла однозначно ідентифікувати користувача в системі BrowserID, необхідно встановити верифікований зв'язок між браузером і цією адресою електронної пошти. Цей зв'язок досягається за допомогою криптографічно підписаного сертифіката, отриманого від постачальника ідентифікації. Сертифікат підписується закритим ключем постачальника ідентифікації і містить наступну інформацію:

1. Адреса електронної пошти користувача.

2. Відкритий ключ.

3. Мітку часу.

4. Доменне ім’я.

5. Термін дії сертифіката.

Для кожної адреси електронної пошти браузер користувача генерує свою пару відкритого і закритого ключів, і ці пари не передаються між різними браузерами користувача. Тому користувач повинен отримувати новий сертифікат кожен раз, коли закінчується час його дії або коли користувач використовує новий браузер або комп'ютер.

Після ідентифікації за допомогою поштової адреси, браузер користувача фоново перевіряє наявність сертифіката для цієї адреси. У випадку відсутності відповідного сертифіката, браузер запитує новий сертифікат у "постачальника ідентифікації" [5].

1.3.3 OAuth 1.0

OAuth - це відкритий стандарт авторизації, який дозволяє користувачам відкривати доступ до своїх приватних даних, які зберігаються на одному сайті, іншому сайті без необхідності вводу імені користувача та паролю. OAuth дозволяє користувачам надавати сайтам маркери доступу до їх даних, що розміщуються на

інших сайтах, таким чином, не передаючи повною мірою самі дані та без використання імені та пароля. Це дозволяє користувачам контролювати доступ третіх сторін до їх даних та надавати доступ на певний термін.

OAuth відрізняється за своїм призначенням від OpenID та BrowserID. OpenID використовується для ідентифікації користувача на веб-сайтах, в той час як OAuth дозволяє надавати доступ до даних на сайтах. BrowserID, зі свого боку, використовується для авторизації користувачів за допомогою адреси електронної пошти.

OAuth з'явився як результат роботи Блейна Кука та Кріса Мессіно на створення протоколу для доступу до Twitter API, і він став важливим стандартом для авторизації на багатьох інших веб-сервісах. Розробка стандарту почалася у 2006 році, і перша версія OAuth 1.0 була представлена в 2007 році. Впродовж року 2008 велися роботи зі стандартизації протоколу в Інженерній раді Інтернету.

15 квітня 2009 року Twitter представив новий функціонал під назвою "Увійти через Твіттер," який дозволив користувачам делегувати доступ до своїх акаунтів стороннім сайтам і сервісам. Цей функціонал базувався на протоколі OAuth. Але ця подія також стала приводом для виявлення потенційних вразливостей, що стосуються всіх існуючих реалізацій протоколу OAuth. Невдовзі спільнота розробників виявила потенційну вразливість, яка впливає на всі наявні реалізації OAuth. В результаті, 23 квітня було опубліковано перший додаток безпеки до протоколу. Цей додаток було включено до оновленої специфікації OAuth Core 1.0 Revision A, яка була опублікована 24 червня. Нещодавно, у квітні 2010 року, було видано інформаційний документ RFC 5849, який стосується стандарту OAuth.

OAuth 2.0 - це наступне покоління протоколу OAuth і відрізняється від OAuth 1.0, не сумісного з ним. Основний акцент у розробці OAuth 2.0 було зроблено на спрощенні процесу створення клієнтських додатків. Він надає спеціальні потоки дозволу для різних типів додатків, таких як веб-застосунки, настільні застосунки та мобільні додатки. Специфікація OAuth 2.0 була розроблена в рамках робочої групи IETF OAuth.

Facebook з впровадженням свого нового Graph API повністю перейшов на використання OAuth 2.0 і є однією з найбільших реалізацій цього стандарту. Починаючи з 2011 року, Google додав експериментальну підтримку OAuth 2.0 в своєму API [5].

1.3.4 OAuth 2.0

OAuth 2.0 - це відкритий протокол авторизації, який дозволяє користувачам надавати обмежений доступ до своїх приватних ресурсів на веб-додатках та мобільних застосунках третім сторонам без необхідності передавати свій логін та пароль. OAuth 2.0 дозволяє отримувати доступ до ресурсів власника цих ресурсів шляхом використання клієнтських додатків на HTTP-сервісах, таких як Facebook або GitHub, тощо. Замість передачі облікових даних використовуються спеціальні маркери, що робить процес більш безпечним.

Основною перевагою OAuth 2.0 є відсутність необхідності обміну паролями між користувачем і третьою стороною. Він використовує потоки користувацьких агентів для запуску клієнтських додатків, які використовують мову сценаріїв, таку як JavaScript. Браузери часто виступають як користувацькі агенти, які отримують доступ до даних, використовуючи спеціальні токени замість передачі облікових даних, і зберігають ці дані онлайн у файловій системі користувача, такі як Google Docs або Dropbox [7].

1.4 Нормативно-правове забезпечення

1.4.1 RFC 6749. OAuth authorization framework

Стандарт RFC 6749, який належить до серії RFC 6000, представляє собою документ, що надає важливі вказівки та рекомендації щодо використання авторизаційних протоколів в веб-додатках. Цей стандарт включає опис відкритого протоколу авторизації OAuth, який дозволяє третій стороні отримати обмежений доступ до захищених ресурсів користувача, не потрібно надавати їй (третій стороні) логін та пароль. Згідно з цим стандартом, OAuth є фреймворком для авторизації, що

дозволяє додаткам здійснювати обмежений доступ до облікових записів користувачів на HTTP-сервісах, таких як Facebook, GitHub та DigitalOcean. Він працює на основі делегування аутентифікації користувача сервісу, на якому розташований обліковий запис користувача, надаючи сторонньому додатку можливість отримувати доступ до цього облікового запису користувача. OAuth застосовується в Інтернеті, а також в десктопних і мобільних додатках [5].

OAuth визначає наступні ролі:

- Власник ресурсу: це користувач, який має обліковий запис на сервісі та контролює доступ до своїх ресурсів.
- Клієнт: це додаток, який звертається до сервера ресурсів для отримання доступу до ресурсів користувача.
- Сервер ресурсів: це сервер, який зберігає ресурси користука та обробляє запити на доступ до цих ресурсів від клієнтів.
- Авторизаційний сервер: це сервер, який надає авторизацію та видання маркерів доступу для клієнтів на запит власників ресурсів [6].

1.4.2 RFC 6750. Bearer tokens

Згідно зі специфікацією стандарту RFC 6750, протокол авторизації OAuth 2.0 використовує два основних типи токенів:

1. Токени (маркери) доступу (Access Tokens):
   - Токени доступу використовуються додатками для створення запитів API від імені користувача.
   - Маркер доступу представляє авторизацію конкретної програми для доступу до певних частин даних користувача.
   - Токени доступу повинні бути збережені конфіденційно під час транспортування та зберігання.
   - Їх можуть бачити лише програма, сервер авторизації та сервер ресурсів.

- Програма повинна забезпечити, щоб зберігання маркера доступу було недоступним для інших програм на тому ж пристрої.

- Токен доступу може використовуватися тільки через безпечне з'єднання HTTPS для запобігання його перехопленню третіми сторонами.

- Кінцевою точкою для видачі токена є місце, де програми подають запит на отримання маркера доступу для користувача.

2. Токени оновлення (Refresh Tokens):

- Токени оновлення містять інформацію, необхідну для отримання нового маркера доступу.

- Використовуються, коли потрібно оновити маркер доступу, наприклад, після закінчення дії старого маркера або отримання доступу до нового ресурсу.

- Токени оновлення можуть бути тривалими, але їхній термін обмежений.

- Вони дозволяють отримати новий маркер доступу без необхідності повторно входити або надавати логін та пароль.

- Токени оновлення також підлягають вимогам до зберігання, щоб забезпечити їхню безпеку та запобігти несанкціонованому доступу.

- Іноді сервер авторизації може включити токени оновлення до чорного списку та заборонити їхнє подальше використання.

Обидва типи токенів грають важливу роль у забезпеченні безпеки та зручності авторизації в протоколі OAuth 2.0 [8].

1.4.3 RFC 6819. Threat model and security considerations

Стандарти RFC серії 6000 надають комплексну модель загроз для протоколу OAuth 2.0. Згідно з цими стандартами, загрози групуються за наступними спрямуваннями:

1. Атаки на клієнта (Client Threats): Це загрози, спрямовані на безпеку клієнтських додатків, які використовують OAuth. Вони можуть включати в себе атаки, спрямовані на клієнтські маркери та ідентифікатори.

2. Атаки на сервери авторизації (Authorization Server Threats): Ці загрози оцінюють безпеку серверів авторизації, які видавали бар'єри доступу та обробляли запити на авторизацію.

3. Атаки на сервери ресурсів (Resource Server Threats): Тут оцінюється безпека серверів ресурсів, які зберігають захищені ресурси та обробляють запити на доступ до них.

Загрози також можуть бути груповані в залежності від цільового потоку, наприклад:

- Атаки на отримання маркера доступу (Access Token Request Threats): Ці загрози спрямовані на отримання маркерів доступу в процесі авторизації.
- Атаки на отримання маркера оновлення (Refresh Token Request Threats): Це загрози, спрямовані на отримання маркерів оновлення для оновлення маркерів доступу.

Попередження та захист від цих загроз є важливими аспектами реалізації та використання протоколу OAuth 2.0. Ці стандарти надають важливий орієнтир для розробників та операторів систем, що використовують OAuth для забезпечення безпеки і захисту від потенційних загроз [9].

1.5 Порівняльна характеристика протоколів авторизації

На основі міжнародного стандарту RFC 5748 вимоги до протоколів авторизації визначаються за кількома критеріями, які включають в себе:

1. Підтримка мобільних додатків: Протоколи авторизації повинні бути здатні підтримувати інтеграцію з мобільними застосунками, оскільки це важливо для багатьох сучасних додатків і сервісів.

2. Рівень захищеності: Забезпечення високого рівня захищеності є обов'язковим. Протоколи мають забезпечувати надійну і безпечну авторизацію користувачів.

3. Підтримка серед сервісів: Протоколи повинні бути загальноприйнятими та підтримуватися серед багатьма різними сервісами та додатками. Це важливо для інтероперабельності.

4. Зручність: Протоколи повинні бути легкими для реалізації в сервісах та додатках, тобто забезпечувати зручність у використанні.

Порівняння найпоширеніших протоколів авторизації				
	OpenId	Persona	OAuth 1.0	OAuth 2.0
Підтримка мобільних додатків	5	4	1	5
Зручність	3	4	3	5
Рівень захищеності	4	4	4	4
Підтримка серед сервісів	2	3	4	5

Таблиця 1.1 Порівняння найпоширеніших протоколів авторизації

1. Підтримка мобільних додатків: OAuth 2.0 підтримує мобільні додатки, що є важливим для сучасного інтернет-середовища, де користувачі використовують різноманітні пристрої.

2. Широка підтримка серед сервісів: OAuth 2.0 є популярним і підтримується багатьма веб-сервісами та додатками. Це робить його інтероперабельним та використовується у багатьох різних сценаріях.

3. Зручність інтеграції: OAuth 2.0 показує хорошу зручність для інтеграції в різні фреймворки та мови програмування, включаючи C# ASP.NET для серверної частини та JavaScript AngularJS для клієнтської частини.

4. Високий рівень захищеності: OAuth 2.0 забезпечує високий рівень захищеності, що важливо для захисту даних користувачів та уникнення небажаних загроз.

Хоча протоколи OpenID та OAuth 1.0 також мають свої переваги, такі як безпека та інтероперабельність, ваш аналіз свідчить про те, що OAuth 2.0 є найбільш підходящим варіантом для багатьох випадків використання в сучасних додатках та сервісах [7].

РОЗДІЛ 2: АНАЛІЗ ПРОТОКОЛУ АВТОРИЗАЦІЇ OAUTH 2.0. ТА ПРИНЦИП ЙОГО РОБОТИ

2.1 Протокол Oauth 2.0

2.1.1 Специфікація Oauth 2.0

OAuth 2.0 - це протокол делегування доступу, який дозволяє одному сервісу або додатку отримувати доступ до ресурсів іншого сервісу від імені користувача без необхідності використовувати користувачів і паролі. Він використовується для вирішення ситуацій, коли потрібно надавати доступ до даних або функцій іншим сервісам чи додаткам, не розкриваючи власних облікових даних користувача. В основному, OAuth дозволяє одній стороні отримати "токен" від іншої сторони, який використовується для доступу до ресурсів на стороні, яка видала цей токен.

Наприклад, ви можете мати фотографії у хмарному сховищі та хотіти надати іншому сервісу дозвіл на друк цих фотографій. OAuth дозволить цьому сервісу отримати токен для доступу до ваших фотографій без передачі власного пароля. Тобто, OAuth робить можливим делегування обмеженого доступу до ваших ресурсів іншим сервісам.

OAuth 2.0 часто використовується для взаємодії з RESTful веб-сервісами і може бути впроваджений в різних типах додатків, від веб-додатків до настільних програм. Він має гнучку архітектуру, яку можна масштабувати для великої кількості користувачів та ресурсів.

Важливо зазначити, що OAuth не є протоколом автентифікації, хоча його можна використовувати для створення системи автентифікації. Він надає доступ до ресурсів, але сам по собі не визначає, хто є користувачем або чи існує користувач взагалі. Наприклад, фотопринтеру не потрібно знати, хто є користувачем, важливо лише, що дозвіл на друк фотографій був наданий.

OAuth - це протокол, який дозволяє одному компоненту системи, такому як клієнтська програма, отримати доступ до захищеного ресурсу від імені власника

ресурсу, який зазвичай є кінцевим користувачем. У специфікації OAuth існують чотири основні ролі:

2.1.2 Ролі в Oauth 2.0

1) Власник ресурсу: Це особа, яка має доступ до API і може делегувати доступ до цього API. Власник ресурсу зазвичай має доступ до веб-браузера.

2) Захищений ресурс: Це компонент системи, до якого має доступ власник ресурсу. Захищений ресурс може бути в різних формах, але це часто веб-сервіс або API, яке надає певні функції або дані.

3) Клієнт: Це програмне забезпечення, яке отримує доступ до захищеного ресурсу від імені власника ресурсу. Термін "клієнт" в OAuth не обов'язково вказує на веб-браузер; це може бути будь-яке програмне забезпечення, яке використовує API захищеного ресурсу.

4) Сервер авторизації: Цей сервер надає клієнту маркери доступу після успішної автентифікації власника ресурсу та отримання авторизації.

OAuth не обов'язково вирішує питання автентифікації користувача (хто він) чи навіть чи існує він. Він призначений для делегування доступу одного компонента системи до іншого, забезпечуючи при цьому безпеку та авторизацію [10].

2.2 Принцип роботи

OAuth - це протокол, спрямований на авторизацію, в якому кінцевий користувач дозволяє клієнтській програмі використовувати обмежені повноваження для доступу до захищеного ресурсу від їхнього імені. Основною метою OAuth є забезпечення безпеки авторизації та делегування доступу. Для досягнення цієї мети вводиться додатковий компонент - сервер авторизації (AS).

Основні кроки в процесі отримання токена доступу OAuth такі:

1. Клієнт запитує авторизацію власника ресурсу.

- У цьому кроці клієнтська програма запитує власника ресурсу про доступ до ресурсу.

2. Коли власник ресурсу надає авторизацію клієнту, клієнт отримує грант авторизації (спеціальний код).

- Власник ресурсу надає клієнту дозвіл на доступ до ресурсу, і цей дозвіл виражається у вигляді гранту авторизації.

3. Клієнт запитує токен шляхом автентифікації за допомогою сервера авторизації та представлення гранту авторизації.

- Клієнтська програма подає запит на отримання токена доступу до сервера авторизації і надає грант авторизації.

4. Сервер авторизації автентифікує клієнта, перевіряє грант авторизації та, якщо він дійсний, видає маркер доступу (токен доступу) і маркер оновлення (токен оновлення).

- Сервер авторизації перевіряє легітимність клієнта, перевіряє грант авторизації та, якщо усе в порядку, видає маркер доступу та маркер оновлення.

5. Клієнт запитує захищений ресурс у постачальника та автентифікується, представляючи маркер доступу.

- Клієнтська програма подає запит до захищеного ресурсу та представляє маркер доступу.

6. Постачальник перевіряє доступ до маркерів та, якщо вони дійсні, обслуговує запит.

- Постачальник перевіряє легітимність маркерів доступу та, якщо вони дійсні, надає доступ до захищеного ресурсу.

Цей процес дозволяє клієнту отримувати доступ до ресурсу від імені власника ресурсу, не викриваючи при цьому облікових даних власника ресурсу.

Однією з ключових переваг протоколу OAuth є те, що власник ресурсу (зазвичай кінцевий користувач) не повинен відкривати свої облікові дані (логін і пароль) клієнту. Всі облікові дані авторизації та делегування доступу керуються на

сервері авторизації, і власник ресурсу має повну контроль над дозволами та доступом до свого ресурсу.

Зазвичай клієнти взаємодіють із серверами авторизації через стандартний веб-браузер або інші зручні інтерфейси. Власник ресурсу може авторизувати клієнта, надаючи дозвіл на доступ під час авторизаційного процесу. Отримавши дозвіл, клієнт отримує маркер доступу, який використовується для отримання доступу до захищеного ресурсу від імені власника ресурсу.

Однією з переваг такого підходу є те, що власник ресурсу може контролювати та відкликати доступ до свого ресурсу для конкретного клієнта без необхідності змінювати свій пароль або облікові дані. Це робить процес керування доступом більш безпечним і зручним для користувачів.

Загалом, OAuth дійсно полегшує процес авторизації та керування доступом, і він широко використовується в інтернет-додатках для забезпечення безпеки та зручності для користувачів [11].

2.2.1 Схема роботи протоколу авторизації OAuth 2.0

Розглядаючи ролі авторизації в протоколі OAuth 2.0 та їх взаємодію, абстрактна схема роботи цього протоколу ілюструється на рисунку 1.1.

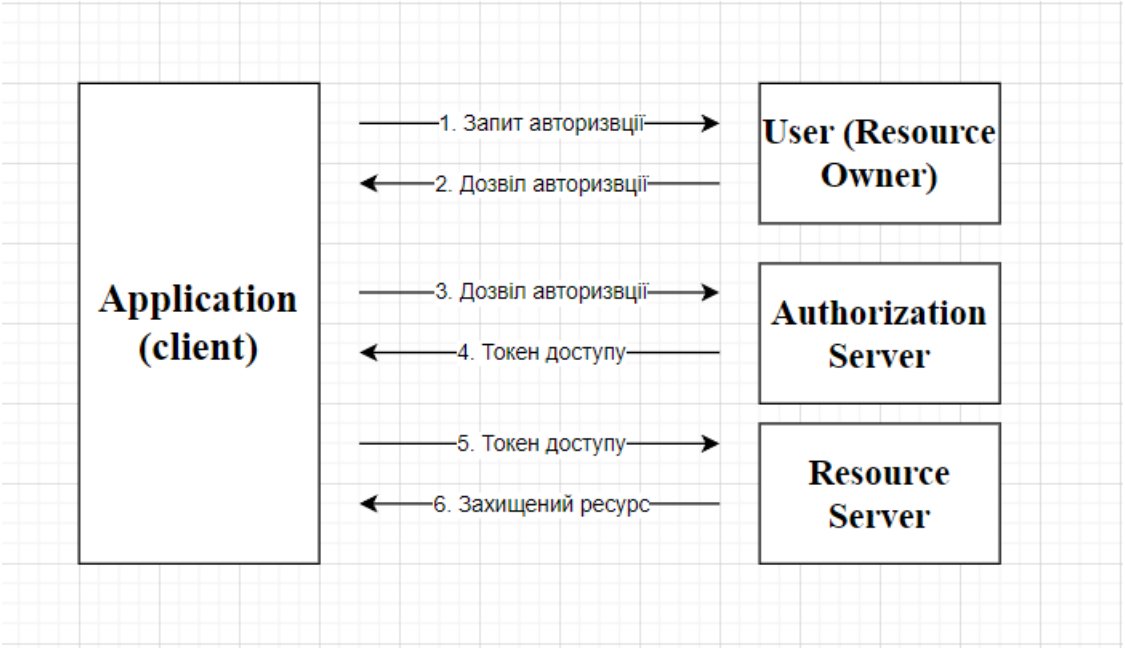

Рисунок 1.1Схема роботи протоколу OAuth 2.0

Під час проходження алгоритму авторизації в веб-додатках за допомогою протоколу авторизації OAuth 2.0 діє наступна послідовність кроків:

1. Додаток запитує у користувача дозвіл на доступ до сервера ресурсів.
2. Якщо користувач надає дозвіл, додаток отримує дозвіл на авторизацію (authorization grant).
3. Додаток запитує авторизаційний токен у сервера авторизації (API), надаючи інформацію про себе та дозвіл на авторизацію від користувача.
4. Якщо справжність додатка підтверджена, і дозвіл на авторизацію є дійсним, сервер авторизації (API) створює токен доступу для додатка, завершуючи процес авторизації.
5. Додаток запитує ресурс у сервера ресурсів (API), надаючи при цьому токен доступу для аутентифікації.
6. Якщо токен є дійсним, сервер ресурсів (API) надає запитуваний ресурс додатку.

Ця послідовність кроків описує загальний процес авторизації в рамках протоколу OAuth 2.0. Фактичний порядок кроків може різнитися в залежності від типу дозволу на авторизацію, який використовується, але загальна суть процесу лишається незмінною. Далі будуть розглянуті різні типи дозволів на авторизацію, які можуть використовуватися в залежності від конфігурації додатку. [12],

2.3 Атаки на протокол OAuth 2.0

2.3.1 Недостатня перевірка URI перенаправлення

Деякі сервери авторизації дозволяють клієнтам реєструвати шаблони URI переадресації замість повних URI перенаправлень. Це означає, що сервер авторизації використовує ці шаблони для порівняння з фактичними значеннями параметра URI переадресації в кінцевій точці авторизації.

Цей підхід має свої переваги і недоліки:

Переваги:

1. Зручність для клієнтів: Шаблони URI переадресації дозволяють клієнтам кодувати стан транзакції або реєструвати один шаблон для декількох URI перенаправлень. Це полегшує роботу клієнтів та спрощує їхню реалізацію.

2. **Швидкість перевірки:** Порівняння зі шаблонами може бути швидше, ніж порівняння з повними URI перенаправленнями, що дозволяє прискорити процес авторизації.

Недоліки:

1. Складність та помилки: Використання шаблонів може бути складніше для реалізації, і це може призвести до збільшення кількості помилок. Неправильно налаштовані шаблони можуть призвести до некоректного доступу або авторизації.

2. Потенційні атаки: Деякі успішні атаки використовують недоліки в реалізації шаблонів URI переадресації. Це може вразити ідентифікацію або автентифікацію клієнта і дозволити зловмиснику отримати код авторизації або маркер доступу. Наприклад, атаки можуть включати відправку агента користувача до URI, контроль над яким перейшов до зловмисників, або розкриття облікових даних OAuth через відкрите перенаправлення та обробку фрагментів URL-адреси агентами користувача.

З цими перевагами і недоліками важливо ретельно розглянути, які параметри і функції безпеки важливі для конкретного випадку використання та вибрати відповідний підхід до авторизації [15].

2.3.1.1 Атака на надання коду авторизації

Атака, яка використовує недолік в шаблоні URI перенаправлення для клієнта, який використовує код типу гранту. Ось кроки цієї атаки:

1. Створення підробленого URL: Зловмисник створює підроблений URL, наприклад, "https://www.eviler.example", і намагається обдурити користувача для відкриття цього URL у своєму браузері. Зловмисник може виглядати, як легітимний веб-сайт, щоб спростити цю атаку.

2. Запит авторизації: Після того як користувач відкриває підроблений URL, запит авторизації надсилається на сервер авторизації. У запиті є параметри, такі як "response_type", "client_id", "state" і "redirect_uri". Цей запит намагається отримати авторизацію від легітимного клієнта, використовуючи підроблений URL в параметрі "redirect_uri".

3. Перевірка URI перенаправлення: Сервер авторизації перевіряє URI перенаправлення для ідентифікації клієнта. Оскільки шаблон URI дозволяє довільні імена доменів в "website.example", сервер авторизації вважає, що запит на авторизацію є легітимним. Це може призвести до обробки запиту як від легітимного клієнта.

4. Отримання "коду" авторизації: Якщо користувач не розпізнає атаку, сервер авторизації видає "код" авторизації та надсилає його безпосередньо зловмиснику.

5. Обмін на маркери доступу: Оскільки зловмисник представляє собою публічного клієнта, він може обміняти "код" авторизації на маркери доступу на відповідній кінцевій точці маркера. Зловмисник може отримати доступ до ресурсів від імені законного користувача.

Ця атака використовує недолік в перевірці URI перенаправлення та дозволяє зловмиснику отримати доступ до ресурсів від імені законного користувача. Для запобігання таким атакам важливо належним чином налаштувати перевірку URI перенаправлення та використовувати надійні методи авторизації та аутентифікації.

Обмін кодами передбачає автентифікацію клієнта з легітимним секретом клієнта. Це значно ускладнює можливість зловмиснику отримати доступ до коду авторизації без знання секрету клієнта.

Зловмисник, який хотів би використовувати конфіденційного клієнта для отримання коду, повинен бути в змозі отримати секрет клієнта або виконати інші атаки, щоб отримати доступ до секрету. Це значно складніше і потребує додаткових зусиль і навичок зловмисника.

Таким чином, конфіденційні клієнти забезпечують додатковий рівень безпеки у порівнянні з публічними клієнтами, оскільки вони вимагають валідного секрету клієнта для успішного обміну кодами [15].

2.3.1.2 Атака на неявний грант

Атака, описана вище, працює також у випадку неявного надання гранту. Якщо зловмисник може відправити відповідь серверу авторизації до URI, який перебуває під його контролем, він може отримати доступ до фрагмента з маркером доступу.

Зловмисники також можуть використовувати агентів користувачів для обходу строгих шаблонів URI переадресації, надсилаючи фрагменти до цільової URL переадресації, якщо заголовок розташування не включає їх. Ця атака поєднує цю поведінку з тим, що клієнт виступає в ролі відкритого переадресувача. Це дозволяє обходити навіть строгі шаблони URI переадресації (але не строгу перевірку URL).

Припустимо, що шаблон для клієнта "d3GfVHdmt7" виглядає так: "https://client.website.example/cb?*," що дозволяє будь-які параметри для перенаправлення на "https://client.website.example/cb." Однак, цей клієнт, який виступає в ролі відкритого переадресувача, вразливий до атаки. В цьому випадку, зловмисник може скористатися цією слабкістю.

Інша атака використовує відкритого переадресувача та включає параметр "redirect_to," який приймає цільовий URL та відправляє браузер до цієї URL за допомогою HTTP заголовка перенаправлення Location 302. Такий метод дозволяє зловмиснику отримати маркер доступу через використання агента користувача.

Ці атаки підкреслюють важливість належного налаштування та застосування механізмів безпеки для запобігання подібним ризикам.

Подібно до попередньої атаки, зловмиснику необхідно переконати користувача відкрити підроблений URL в браузері, який запускає сторінку, контроль над якою знаходиться у зловмисника, наприклад, "https://www.eviler.example."

1. Коли користувач відкриває цей підроблений URL, він ініціює запит на авторизацію, схожий на атаку з використанням потоку коду. Проте в даному випадку використовується відкритий переадресувач, і параметр "redirect_to" кодується у URI перенаправлення, а також вказується тип відповіді як "маркер."

2. Приклад запиту на авторизацію виглядає наступним чином:

GET

/authorize?response_type=token&client_id=d3GfVHdmt7&state=xyz&redirect_uri=https%3A%2F%2Fclient.website.example%2Fcb%26redirect_to%253Dhttps%253A%252F%252Fclient.eviler.example%252Fcb HTTP/1.1

Host: server.website.example

3. Оскільки URI перенаправлення відповідає зареєстрованому шаблону, сервер авторизації дозволяє цей запит і надсилає отриманий маркер доступу через перенаправлення HTTP 302.

4. Отримавши відповідь, переадресація на сайт example.com виконується через відкритого переадресувача, який читає параметр перенаправлення та виконує перенаправлення HTTP 302 на URL "https://eviler.example.com/cb."

5. Оскільки у заголовку розташування переадресації відсутній фрагмент, агент користувача повторно додає оригінальний фрагмент до URL і переходить за ним за наступним посиланням:

https://client.eviler.example/cb#access_token=3FsYndGDVjr4dXgfdCGkdFF&...

6. На цій сторінці, що належить зловмиснику (client.evil.example), може бути отримано доступ до фрагмента і, в результаті, маркера доступу.

Ця атака наголошує на важливості належної захисту та безпеки при використанні авторизаційних механізмів для уникнення подібних ризиків [15].

2.3.2 Витік «кодів» або станів через сторони клієнта або сервера авторизації через Referrer заголовки

Коди авторизації або значення стану можуть ненавмисно витікати до зловмисників через Referrer заголовок в ситуаціях, коли існує можливість витоку

інформації з веб-сайту клієнта або з веб-сайту сервера авторизації. Це може статися через два основних сценарії:

1. Витік з клієнта OAuth: Цей сценарій передбачає, що клієнт, після успішного запиту на авторизацію, відображає сторінку, яка містить або посилається на сторінки, що перебувають під контролем зловмисника (наприклад, реклама, часто задані питання тощо), або містить вміст з третіх сторін (наприклад, зображення), особливо, якщо ця сторінка містить користувацький контент (наприклад, блог). Якщо користувач переходить на такі сторінки або завантажує вміст з третіх сторін, зловмисник отримує інформацію про URL-адресу відповіді на авторизацію, і це дозволяє йому видобувати код авторизації або значення стану.
2. Витік з серверу авторизації: Аналогічно, зловмисник може дізнатися значення стану, якщо кінцева точка авторизації на сервері авторизації містить посилання або вміст з третіх сторін, як зазначено вище.

Наслідки цих витоків даних полягають у тому, що зловмисники, які мають доступ до дійсних кодів авторизації через Referrer заголовок, можуть використовувати їх для виконання атак, які були описані в попередніх розділах. Якщо зловмисник отримує значення стану, це може позбавити захист від атак CSRF, що призводить до потенційних атак на перехоплення сесії. Тому важливо вживати заходів для захисту від таких можливих витоків даних і ретельно розробляти та впроваджувати заходи безпеки в OAuth-протоколі 16].

2.3.3 Атаки через історію браузера

Коди авторизації та маркери доступу можуть бути збережені в історії відвідуваних URL-адрес веб-переглядача, що створює потенційну загрозу для наступних атак, які розглядаються нижче.

2.3.3.1 Код в історії браузера

Коли браузер переходить до "client.com/redirection_endpoint?code=asdf" після перенаправлення з кінцевої точки авторизації провайдера, URL-адреса, що включає код авторизації, може зберегтися в історії браузера. Якщо зловмисник має доступ до пристрою, він може отримати цей код та спробувати його використати для незаконного доступу.

2.3.3.2 Маркер доступу в історії браузера

Маркер доступу може потрапити в історію веб-переглядача, якщо клієнт або навіть веб-сайт, який вже має маркер, свідомо переходить на сторінку, наприклад, «provider.com/get_user_profile?access_token=qwerty». Фактично, такий спосіб передачі маркерів не є рекомендованим, і бажано передавати маркери через HTTP-заголовок. Однак на практиці, веб-сайти часто просто включають маркер доступу до параметрів запиту.

У випадку неявного гранту, URL-адреса, схожа на "client.com/redirection_endpoint#access_token=qwerty", також може зберігатися в історії веб-переглядача після перенаправлення з кінцевої точки авторизації провайдера [17].

2.3.4 Підміна

Атака підміни - це сценарій атаки, в якому клієнт OAuth взаємодіє з декількома серверами авторизації, часто з використанням динамічної реєстрації. Мета атаки полягає в тому, щоб отримати код авторизації або маркер доступу, примусивши клієнта надсилати ці дані зловмиснику замість використання їх на відповідному сервері авторизації або ресурсу.

Передумови для успішної атаки включають наступне:

1. Використання декількох серверів авторизації, один з яких є "чесним" (H-AS), а інший - керується зловмисником (A-AS).

2. Клієнт зберігає в сеансі, пов'язаному з браузером користувача, вибраний користувачем сервер авторизації і використовує той же URI кінцевої точки перенаправлення для кожного сервера авторизації.

3. Зловмисник має можливість маніпулювати першою парою запит/відповідь від браузера користувача до клієнта (де користувач вибирає певний сервер авторизації і потім перенаправляється клієнтом до цього серверу).

Нижче наведені різні варіанти атаки, які можуть вимагати різних передумов. У наступних прикладах ми припускаємо, що клієнт зареєстрований як з H-AS (URI: «https://honest.as.example», id клієнта: 2FZSJNLieQ) і з A-AS (URI: «https://attacker.example», id клієнта: EV666DIL).

Атака на надання коду авторизації може бути виконана наступним чином:

1. Користувач починає процес надання гранту, вибираючи H-AS (чесний сервер авторизації), наприклад, натискаючи кнопку на веб-сайті клієнта.

2. Зловмисник перехоплює цей запит та змінює вибір користувача на "A-AS" (сервер авторизації зловмисника).

3. Клієнт зберігає інформацію в сеансі користувача про те, що користувач обрав "A-AS" та перенаправляє користувача на кінцеву точку авторизації A-AS, надіславши наступну відповідь:

HTTP/1.1 302 Found

Location:

https://attacker.example/authorize?response_type=code&client_id=666RVZJTA

4. Зловмисник перехоплює цю відповідь та змінює перенаправлення таким чином, що користувач перенаправляється на H-AS (чесний сервер авторизації). Зловмисник також замінює ідентифікатор клієнта A-AS ідентифікатором клієнта H-AS, в результаті чого веб-переглядачу надсилається наступна відповідь:

HTTP/1.1 302 Found

Location:

https://honest.as.example/authorize?response_type=code&client_id=2FZSJNLieQ

5. Тепер користувач надає клієнту доступ до своїх ресурсів на H-AS. H-AS видає код і надсилає його (через веб-переглядач) назад до клієнта.

6. Оскільки клієнт все ще припускає, що код був виданий A-AS, він спробує отримати код на кінцевій точці авторизації A-AS.

7. Таким чином, зловмисник отримує код і може або обміняти його на маркер доступу (для публічних клієнтів), або виконати ін'єкцію коду.

Варіанти цієї атаки включають непрямий грант, при якому зловмисник отримує маркер доступу замість коду, і підміну без перехоплення, коли зловмисник не перехоплює першу відповідь, а просто використовує іншу атаку.

Вищезазначений варіант атаки може працювати навіть у тих випадках, коли першу пару запит/відповідь не можна перехопити через використання TLS або інших захисних заходів. В цьому випадку, якщо користувач хоче використовувати A-AS для надання гранту, а зловмисник хоче перенаправити його на H-AS, атака може виконуватися так:

1. Користувач починає процес надання гранту, вибираючи A-AS.

2. Зловмисник негайно перенаправляє користувача на H-AS, змінюючи ідентифікатор клієнта на "2FZSJNLieQ". Пильний користувач може помітити цю спробу перенаправлення на інший сервер авторизації (H-AS замість A-AS).

3. Після цього атака протікає так само, як у першому описаному варіанті.

Другий варіант атаки передбачає, що клієнти використовують різні URI перенаправлення для різних серверів авторизації, і вони не зберігають вибрані сервери авторизації в сесії користувача. Якщо сервери авторизації не виконують ретельну перевірку URI перенаправлення, то зловмисники можуть використовувати атаку "Міжсайтова підробка запиту" для перенаправлення користувачів на інші сервери авторизації. У цьому випадку зловмисники можуть спробувати перехопити авторизацію або отримати доступ до даних користувача.

Ці атаки підкреслюють важливість належної конфігурації серверів авторизації та вибору відповідних заходів безпеки для захисту від можливих атак [18].

2.3.5 Ін'єкція коду

Атака ін'єкції коду авторизації передбачає спробу зловмисника використовувати вкрадений "код" авторизації на своєму клієнті або в інших контекстах, де він може бути використаний. Проте, в деяких ситуаціях зловмисник може зіткнутися з обмеженнями, які роблять це складним або неможливим:

1. Зловмисник хоче видати себе за жертву в певному застосунку або на веб-сайті: У деяких випадках зловмисники можуть намагатися видати себе за іншого користувача і отримати доступ до функцій конкретного клієнта. Проте це може виявитися складним завданням, оскільки вимагає імітації поведінки жертви та доступу до її облікових даних

2. Код прив'язаний до конфіденційного клієнта: Якщо "код" авторизації прив'язаний до певного конфіденційного клієнта, зловмисник може стикнутися з проблемою отримання необхідних облікових даних цього клієнта для використання "коду". В цьому випадку важко або навіть неможливо використовувати "код" на сторонньому клієнті.

3. Обмеженість мереж і доступу: Деякі сервери авторизації або ресурси можуть бути обмежені певними мережами або статусом довіри. Зловмисники можуть не мати можливості отримати доступ до цих мереж і, відповідно, використовувати "код" авторизації.

Зловмисники можуть намагатися подолати ці обмеження, але це може бути складним завданням і не завжди вдається. Розробники повинні бути уважними та вживати відповідні заходи безпеки, щоб уникнути атак ін'єкції коду авторизації та захистити облікові дані користувачів і клієнтів.

Описана атака на ін'єкцію коду авторизації є досить складною та зазнає декількох перевірок безпеки на шляху, що робить її менш ймовірною. Ось ключові кроки та відповідні важливі перевірки на кожному з них:

1. Зловмисник отримує код авторизації: Цей код може бути вкрадений через різні атаки, включаючи підміну, перехоплення інших атак, описаних в попередніх повідомленнях.

2. Зловмисник вводить вкрадений код авторизації: На цьому етапі зловмисник вводить вкрадений код авторизації від імені користувача.

3. Клієнт надсилає код до кінцевої точки маркера сервера авторизації: Зловмисник відправляє вкрадений код і інші параметри, включаючи ідентифікатор клієнта, секрет клієнта і фактичний URI перенаправлення, які коректно налаштовані в зловмисника.

4. Сервер авторизації перевіряє секрет клієнта і інші параметри: Сервер авторизації перевіряє, чи секрет клієнта відповідає ідентифікатору клієнта, чи URI перенаправлення відповідає налаштованому для цього клієнта. Це важлива перевірка на підтвердження правомірності запиту.

5. Сервер авторизації видає маркери доступу і т. д.: Якщо всі перевірки (у тому числі перевірка ідентифікатора клієнта і URI перенаправлення) проходять успішно, сервер авторизації видаватиме клієнту доступ і інші маркери.

6. Спроби атаки ін'єкції коду: Зловмисник намагатиметься використовувати код на своєму клієнті або в інших контекстах. Проте виявлення атаки може відбутися на цьому етапі, якщо існують перевірки на сценарій підміни коду авторизації. Наприклад, якщо сервер авторизації зберігає повний URI перенаправлення та порівнює його з параметром redirect_uri.

Загальна ідея цієї атаки полягає в тому, що зловмисник намагається використовувати код авторизації, який був виданий одному клієнту, на іншому клієнті. Однак ретельні перевірки сервера авторизації можуть допомогти виявити атаку і відмовити у використанні недійсного коду.

Зазначені рекомендації стосовно прив'язування "коду" авторизації до певного екземпляра клієнта на певному пристрої, для певного агента користувача, в контексті певної транзакції є важливими для забезпечення безпеки авторизації OAuth. Ці рекомендації допомагають запобігти ін'єкції коду авторизації в іншому контексті, ніж під час авторизації користувача.

Основні переваги цього підходу:

1. Прив'язка до конкретного клієнта: Кожен "код" авторизації пов'язаний з конкретним клієнтом, що зменшує можливість використання коду на іншому клієнті.

2. Прив'язка до конкретного пристрою: "Код" авторизації може бути пов'язаний з конкретним пристроєм або агентом користувача, що допомагає уникнути використання коду на інших пристроях.

3. Прив'язка до конкретної транзакції: Кожна транзакція отримання "коду" авторизації повинна бути унікальною, що запобігає повторному використанню коду.

Однак важливо пам'ятати, що реалізація цих рекомендацій може бути складнішою для клієнтів, оскільки вони повинні правильно зберігати та прив'язувати код авторизації до конкретного клієнта, пристрою та транзакції. Проте ця додаткова складність в реалізації є необхідною для підвищення безпеки авторизації OAuth і запобігання потенційним атакам [19].

2.3.6 Міжсайтова підробка запиту

Атака міжсайтового підроблення запиту на URI перенаправлення клієнта дозволяє зловмиснику вставити власний "код" авторизації або маркер доступу. Ця атака може призвести до того, що клієнт використовує маркер доступу, пов'язаний з атакуючим, замість жертви, і отримує доступ до захищених ресурсів атакуючого. Ось як вона виглядає:

1. Зловмисник обирає клієнта, наприклад, example.website.com, і розпочинає процес автентифікації, отримуючи перенаправлення від сервера автентифікації, але не переходить за цим посиланням.

2. Замість цього зловмисник зберігає отримане посилання, яке містить "код" авторизації, і додає його до зображення або вбудованого веб-фрейму, такого як iframe.

3. Зловмиснику потрібно змусити користувача виконати HTTP-запит до URL перенаправлення. Це може бути досягнуто, наприклад, шляхом відправлення

користувачу електронного листа або твіту, або змушення його відвідати певний веб-сайт, який містить вбудований iframe зі спеціальним посиланням.

4. Тепер OAuth-аккаунт зловмисника прив'язаний до облікового запису справжнього користувача на example.website.com. Таким чином, зловмисник може авторизуватися в обліковому записі цього користувача і отримувати доступ до його ресурсів.

Ця атака дозволяє зловмиснику використовувати доступ до ресурсів жертви, навіть якщо клієнт намагається авторизуватися на власному обліковому записі.

2.3.7 Виток маркерів доступу на сервері ресурсів

За деяких обставин може відбуватись виток маркерів доступу з серверу ресурсів.

2.3.7.1 Фішинг маркерів доступу шляхом підроблення серверу ресурсів

Зловмисник може налаштувати власний сервер ресурсів і заманити клієнта відправити маркери доступу до цього сервера, які фактично призначені для інших серверів ресурсів. Якщо клієнт випадково відправить цей маркер доступу на підроблений сервер ресурсів, зловмисник матиме змогу використовувати його для отримання доступу до інших служб від імені власника ресурсу.

Ця атака передбачає, що під час розробки клієнт не пов'язаний з конкретним сервером ресурсів (і відповідним URL), але під час виконання клієнти об'єднуються з URL сервера ресурсів. Це стається у випадках, коли клієнт використовує стандартний API, наприклад, для електронної пошти, календаря або банківської діяльності, і налаштований користувачем або адміністратором на основі стандартів, які використовуються в організації або визначаються конкретним користувачем [19].

2.3.7.1.1 Маркери обмеженого доступу відправника

Маркер обмеженого доступу зв'язується з конкретним відправником та обмежує область використання маркера на ресурсному сервері. Цей відправник зобов'язаний довести свою здатність до знання певної таємниці як необхідну умову для прийняття маркера доступу на ресурсному сервері. Стандартний потік подій для цього процесу виглядає наступним чином:

1. Сервер авторизації створює зв'язок між маркером доступу та конкретним клієнтом. Ця прив'язка може використовувати ідентифікацію клієнта, але в більшості випадків сервер авторизації використовує матеріал ключа (або дані, отримані з матеріалу ключа), який відомий клієнту.

2. Матеріал ключа повинен бути переданий між сторонами. Цей матеріал ключа може існувати перед створенням прив'язки, або ж сервер авторизації може створити ефемерні ключі. Спосіб передачі існуючого матеріалу ключа залежить від конкретних методів і може включати в себе використання сертифікатів X.509 або здійснювати автоматичний обмін на рівні TLS.

3. На сервері ресурсів проводиться перевірка доказу володіння. Зазвичай ця перевірка здійснюється на рівні додаткового програмного забезпечення, але може також використовувати можливості та функції на рівні транспортного рівня, такі як TLS.

2.3.7.1.2 Маркери обмеженого доступу для аудиторії

Обмеження аудиторії передбачає, що кожен маркер доступу пов'язується з певним сервером ресурсів. Сервер авторизації асоціює маркер доступу із конкретним сервером ресурсів, і кожен сервер ресурсів повинен перевірити, чи дозволено використовувати цей маркер на своєму сервері. Якщо маркер доступу не відповідає серверу ресурсів, то сервер ресурсів відхилить запит на обробку.

Ця практика в основному використовує логічні або фізичні адреси (наприклад, URL) для визначення аудиторії, на яку розповсюджується маркер доступу. Щоб запобігти фішингу, важливо використовувати фактичну URL-адресу,

на яку клієнт надсилатиме запити. У випадку фішингу, зловмисник намагатиметься використовувати маркер доступу на справжньому сервері ресурсів (який має іншу URL-адресу), але сервер ресурсів виявить невідповідність (неправильна аудиторія) і відхилить запит.

У розгорнутих проектах, де сервер авторизації знає URL-адреси всіх серверів ресурсів, сервер авторизації може просто відмовити видавати маркери доступу для URL-адрес невідомого сервера ресурсів.

Клієнт повинен повідомити серверу авторизації, на якій URL-адресі він буде використовувати маркер доступу, на який він запитує. Цю інформацію можна кодувати в області видимості або інших частинах запиту.

Замість URL-адреси, також можна використовувати відбиток сертифіката X.509 сервера ресурсів як значення для аудиторії. Це рішення допомагає захистити від спроби крадіжки URL-адреси справжнього серверу ресурсів, особливо за умови використання дійсного сертифіката TLS, отриманого від іншого центру сертифікації. Крім того, це сприяє підвищенню конфіденційності, оскільки приховує URL-адресу серверу ресурсів від сервера авторизації.

Обмеження аудиторії виглядає досить просто, оскільки не потребує виконання жодних криптографічних операцій на стороні клієнта. Однак, оскільки кожен маркер доступу пов'язаний з певним сервером ресурсів, клієнт також повинен отримати різні маркери доступу, специфічні для кожного сервера ресурсів, якщо він планує отримати доступ до декількох серверів ресурсів.

Важливо відзначити, що обмеження аудиторії, або вказівка від клієнта до сервера авторизації, де він планує використовувати маркер доступу, має додаткові переваги, які виходять за рамки запобігання витоку маркера. Вона дозволяє серверу авторизації створювати різні маркери доступу, які мають формат і вміст, специфічний для конкретного сервера ресурсів. Це забезпечує великі переваги в ефективності та збереженні конфіденційності при використанні структурованих маркерів доступу [20].

2.3.7.2 Скомпрометований сервер ресурсів

Зловмисник може спробувати компрометувати сервер ресурсів для отримання доступу до його ресурсів та інших ресурсів у відповідному розгортанні. Рівень компрометації може варіюватися від обмеженого доступу до системи, такого як доступ до журнальних файлів сервера, до повного контролю над відповідним сервером. Якщо зловмисник отримає повний контроль, включаючи доступ до оболонки, він зможе обійти всі наявні контролі та отримати доступ до ресурсів без обмежень. Зловмисник також може отримати доступ до маркерів доступу, які надсилаються на компрометовану систему і можуть бути використані для доступу до інших серверів ресурсів. Навіть у випадку, коли зловмисник має доступ лише до лог-файлів або баз даних серверної системи, він все одно може отримати доступ до справжніх маркерів доступу.

Для запобігання порушенням сервера шляхом зміцнення та моніторингу серверних систем використовуються стандартні операційні процедури, які виходять за рамки цього документа. Однак цей розділ фокусується на впливі таких порушень на частини екосистеми, пов'язані з OAuth, зокрема на відтворення викрадених маркерів доступу на компрометованому сервері ресурсів та інших серверах ресурсів у відповідному розгортанні.

Наступні заходи повинні бути розглянуті виконавцями для боротьби з відтворенням маркера доступу:

- Сервер ресурсів повинен розглядати маркери доступу, як конфіденційні дані, і вважається хорошою практикою не зберігати їх у звичайному текстовому форматі.
- Використання обмеження відправника, як описано в розділі 3.7.1.2, допомагає запобігти відтворенню маркерів доступу на інших серверах ресурсів. В залежності від рівня інфільтрації це також запобігає відтворенню компрометованої системи.
- Використання обмеження аудиторії, як описано в розділі 3.7.1.3, може бути використано для запобігання відтворенню викрадених маркерів доступу на інших серверах ресурсів[20].

2.3.8 Відкриті перенаправлення

Можливі наступні загрози виникають тоді, коли сервер авторизації або клієнт дозволяють відкриті перенаправлення, що означає, що URL може ініціювати HTTP-перенаправлення на веб-сайт, який контролюється зловмисником.

2.3.8.1 Сервер авторизації як відкрите перенаправлення

Зловмисники можуть намагатися скористатися довірою користувачів до сервера авторизації, зокрема до його URL, для виконання фішингових атак. Незважаючи на рекомендацію сервера авторизації не автоматично перенаправляти агента користувача при недійсних поєднаннях client_id і redirect_uri, зловмисники також можуть використовувати коректно зареєстрований URI перенаправлення для фішингових атак. Наприклад, вони можуть зареєструвати клієнта через динамічну реєстрацію і намагатися спровокувати автоматичне перенаправлення агента користувача на фішинговий сайт зловмисника, відправляючи недійсний запит на авторизацію, такий як помилкове значення області видимості.

Серверу авторизації слід вжити відповідних заходів для запобігання цій загрозі. Основуючись на оцінці ризику, сервер авторизації повинен вирішити, чи можна довіряти URI перенаправленням, і чи слід автоматично перенаправляти агента користувача, якщо сервер довіряє URI перенаправлення. У випадку недовіри, сервер авторизації повинен інформувати користувача про свою намір перенаправити його на інший сайт і розраховувати на користувача для прийняття рішення. Або ж просто повідомити користувача про помилку [21].

2.3.8.2 Клієнти як відкрите перенаправлення

Клієнтам не слід розголошувати, які URL-адреси можна використовувати як відкриті перенаправлення. Це важливо для того, щоб запобігти можливості зловмисників створювати URL-адреси, які видаватимуться за дозволені і змушуватимуть користувачів довіряти їм та переходити за такими посиланнями у своєму веб-переглядачі. Інший вид зловживання полягає в створенні URL-адрес, що

вказують на клієнта, і використання їх для того, щоб видати себе за клієнта при взаємодії з сервером авторизації.

Для запобігання відкритим перенаправленням клієнти повинні обмежувати цю можливість лише до "білого списку" цільових URL-адрес, які є дозволеними, або ж визначати джерело запиту, яке може бути автентифіковано[23].

РОЗДІЛ 3: РЕАЛІЗАЦІЯ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ У ВЕБ-ДОДАТКУ НА ОСНОВІ ПРОТОКОЛУ OAUTH 2.0

3.1 Технології, які використовуються в проекті

3.1.1 Мова програмування Java

Java - це широко використовувана об'єктно-орієнтована мова програмування та платформа, яка працює на багатьох різних пристроях, включаючи ноутбуки, мобільні пристрої, ігрові консолі, медичні пристрої та багато інших. Правила та синтаксис Java базуються на мовах C і C++.

Однією з головних переваг розробки програм на Java є її портативність. Після написання коду для Java-програми на комп'ютері його легко перенести на мобільний пристрій. Коли мову створено у 1991 році Джеймсом Гослінгом у компанії Sun Microsystems (пізніше придбаній Oracle), основною метою була можливість "писати один раз і виконувати де завгодно".

Java - це технологія, яка включає в себе мову програмування і програмну платформу. Для створення програми на Java вам потрібно завантажити Java Development Kit (JDK), який доступний для операційних систем Windows, macOS та Linux. Ви програмуєте на мові Java, і потім компілятор перетворює програму в байт-код Java - набір інструкцій для віртуальної машини Java (JVM), яка є частиною середовища виконання Java (JRE). Байт-код Java може виконуватися на будь-якій системі, яка підтримує JVM, що робить ваш код на Java переносним і можливим до виконання на будь-якому пристрої.

Програмна платформа Java складається з декількох ключових компонентів: Java Virtual Machine (JVM), Java API та інтегрованого середовища розробки (IDE). JVM відповідає за аналіз і виконання (інтерпретацію) байт-коду Java. Java API включає обширну бібліотеку, яка охоплює різноманітні функції, такі як робота з об'єктами, мережеві операції, забезпечення безпеки, генерація розширених мов розмітки XML та розробка веб-сервісів. В сукупності, Java мова програмування та

програмна платформа Java утворюють потужну та перевірену технологію для розробки корпоративних програм.

Технологія Java ідеально підходить для створення веб-додатків та служить основою для цифрового бізнесу у різних галузях. Сервери додатків Java діють як веб-контейнери для компонентів Java, XML та веб-сервісів, які взаємодіють з базами даних і генерують динамічний веб-контент. Ці сервери надають стійке середовище для розгортання корпоративних програм з такими можливостями, як керування транзакціями, забезпечення безпеки, кластеризація, продуктивність, доступність, підключення та масштабованість.

3.1.2 Фреймворк Spring Boot

Spring є найпопулярнішим фреймворком для розробки корпоративних додатків на Java. Цей відкритий фреймворк, створений Родом Джонсоном, був випущений у червні 2003 року під ліцензією Apache 2.0. Початково Spring був розроблений як фреймворк для інверсії залежностей, але з часом до нього було додано численні модулі, які значно розширили його функціонал.

Spring Framework надає розробникам можливість створювати високопродуктивні, легко тестовані та повторно використовувані додатки на платформі Java. Цей фреймворк дозволяє використовувати свої основні можливості для розробки будь-яких Java-додатків і має розширення для створення веб-додатків поверх платформи Java EE. Один з основних принципів Spring - це спрощення розробки додатків Java EE і використання моделі програмування на основі POJO, що сприяє впровадженню передових практик програмування.

Spring Framework має декілька переваг, які роблять його популярним фреймворком для розробки корпоративних додатків на Java:

1) Використання POJO: Spring дозволяє розробникам створювати корпоративні програми за допомогою простих об'єктів Java, відомих як Plain Old Java Objects (POJO). Ви не повинні перейматися внутрішніми залежностями, бо Spring вирішує цю проблему.

2) Модульність: Spring організований у вигляді модулів. Вам не потрібно включати всі пакети і класи, лише ті, які вам потрібні, що робить розробку більш структурованою.

3) Використання існуючих технологій: Spring не винаходить щось нове, але інтегрує різні існуючі технології, такі як фреймворки ORM, журнали, таймери JEE, Quartz і інші. Це дозволяє використовувати краще вже наявне.

4) Легкість тестування: Spring спрощує тестування програм, оскільки він дозволяє вам внедрювати залежності для введення тестових даних. Використання POJO сприяє простоті тестування.

5) Spring Web Framework: Фреймворк для веб-розробки Spring є добре структурованим і надає альтернативу іншим веб-фреймворкам, таким як Struts або іншим менш популярним варіантам.

6) Легкість контейнера IoC: Контейнери IoC в Spring зазвичай легкі, особливо в порівнянні з контейнерами EJB. Це особливо корисно на комп'ютерах з обмеженими ресурсами пам'яті та обчислювальної потужності.

7) Узгоджений інтерфейс керування транзакціями: Spring надає єдиний інтерфейс для керування транзакціями, який може бути налаштований для локальних або глобальних транзакцій, зменшуючи складність управління транзакціями.

Spring Framework включає приблизно 20 модулів, які організовані у наступні категорії: основний контейнер, доступ до даних та інтеграція, веб, AOP (аспектно-орієнтоване програмування), інструментарій та тест. Вони представлені на наступній схемі.

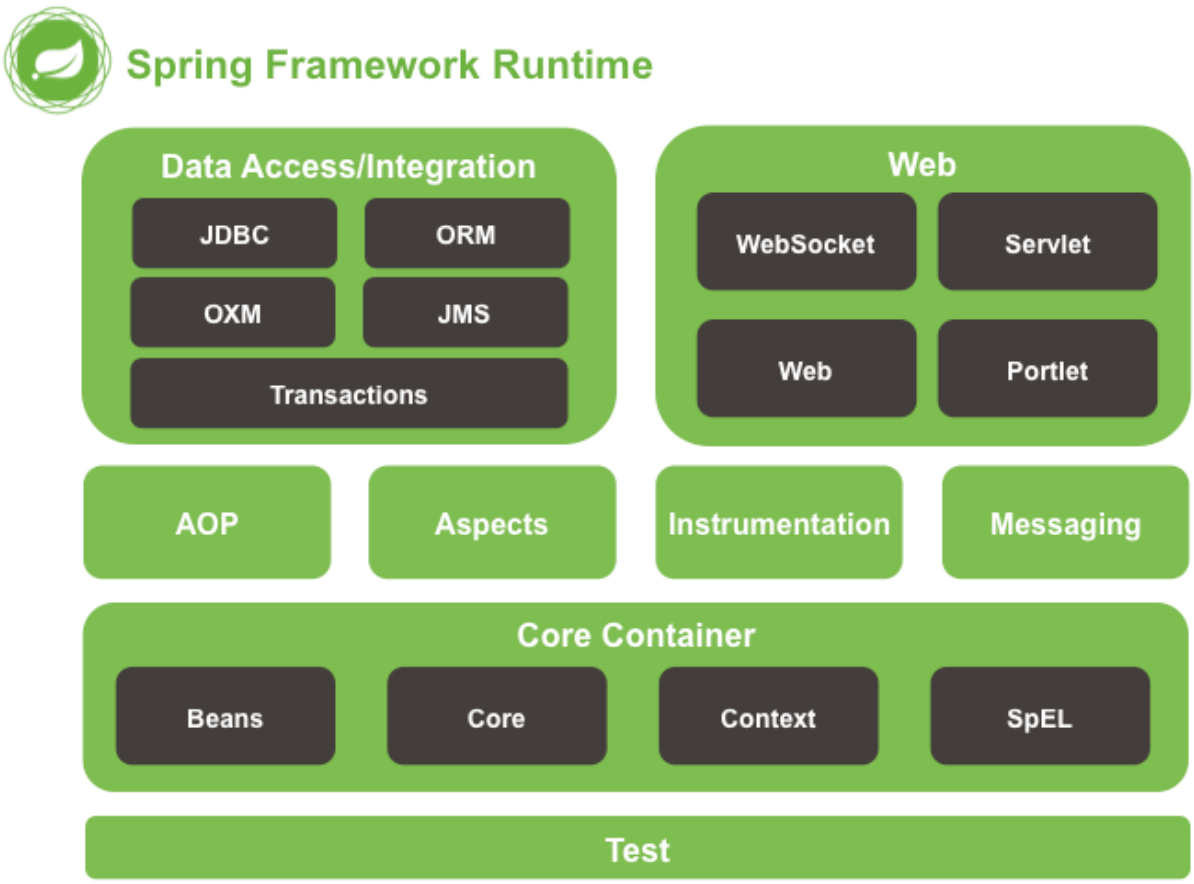

Рисунок 3.1 - Модулі в Spring Framework

Spring Boot побудований на основі фреймворку Spring і має у своєму складі різні залежності, які можна включити у вашу програму Spring. Прикладами є Spring Kafka, Spring LDAP, Spring Web Services і Spring Security. Проте, розробники повинні вручну налаштовувати кожен з цих залежних модулів, що вимагає великої кількості конфігураційних файлів XML або використання анотацій.

З іншого боку, Spring Framework акцентується на забезпеченні гнучкості шляхом ін'єкції залежностей, що спрощує додавання необхідних залежностей і розробку програми з використанням слабкої залежності між компонентами.

Spring Boot, натомість, спрямований на скорочення обсягу коду та надає простий спосіб запуску Spring-додатків. Ви можете розпочати роботу з мінімальними конфігураціями без необхідності ретельного налаштування Spring.

Деякі переваги використання Spring Boot включають:

- Гнучку настройку Java Beans, конфігурацію XML та транзакції бази даних.

- Можливість потужної обробки пакетів та управління точками входу REST.

- Автоматичне налаштування всіх аспектів додатку без необхідності ручних конфігурацій.

- Можливість розробки додатків на основі анотацій.

- Спрощене управління залежностями завдяки стартерам.

- Вбудований контейнер сервлетів у складі Spring Boot.

Spring Boot надає автоматичне налаштування вашої програми на основі залежностей, які ви додаєте до проекту за допомогою анотації @EnableAutoConfiguration. Наприклад, якщо у вашому проекті є база даних MySQL, але ви не вказали жодного підключення до неї, Spring Boot автоматично налаштовує базу даних у режимі в пам'яті. Точкою входу в програму для завантаження Spring Boot є клас, позначений анотацією @SpringBootApplication та основним методом. Spring Boot автоматично сканує всі компоненти, включені в проект, за допомогою анотації @ComponentScan.

Управління залежностями є важливим аспектом будь-якого складного проекту, і вручну це робити не завжди ефективно. Розробники Spring витрачали багато часу на цей процес до виникнення Spring Boot. Стартери Spring Boot були створені, щоб вирішити цю проблему. Стартери - це зручні набори дескрипторів залежностей, які можна включити у вашу програму. Вони автоматично надають всі необхідні залежності для Spring та пов'язаних з ним технологій, звільняючи вас від необхідності пошуку і вставки великої кількості дескрипторів залежностей.

3.1.3 Інструмент автоматизації збірки проекту Maven

Apache Maven - це інструмент автоматизації для управління та побудови програмних проєктів. Спочатку розроблявся для Java-проєктів, і його створив Джейсон ван Зіл у 2002 році. Він відрізняється від Apache Ant та має більш спрощений підхід до конфігурації засобу збірки, використовуючи формат XML. Файл XML описує проєкт, його залежності від зовнішніх модулів і компонентів, а також послідовність збірки (build), теки та необхідні плагіни. Сервер із

додатковими модулями та бібліотеками розміщується на серверах. Maven використовує концепцію Project Object Model (POM) для опису програмного проєкту, його залежностей та порядку збірки.

Основною особливістю Maven є його здатність взаємодіяти з мережею. Ядро може динамічно завантажувати плагіни з репозиторію, що забезпечує доступ до різних версій Java-проєктів з відкритим кодом від Apache та інших джерел. Maven підтримує побудову, завантажуючи артефакти у кінці збірки та зберігаючи їх у локальному кеші.

Maven базується на плагін-архітектурі, що дозволяє стандартизувати вхідні дані для будь-якого інструменту для збірки, тестування та інших завдань. Хоча теоретично Maven міг би підтримувати будь-яку мову, на практиці його підтримка для мов, окрім Java, була обмеженою. Однак тепер існують плагіни для .NET та C/C++.

Головна функція Maven - це управління залежностями в програмних проєктах. Maven використовує систему координат для ідентифікації окремих артефактів, таких як програмні бібліотеки або модулі. Наприклад, POM-файл проекту може вказати залежність від JUnit. Інший проект, якому потрібна бібліотека Hibernate, просто має вказати координати проекту Hibernate у своєму POM. Maven автоматично завантажує ці залежності, а також транзитивні залежності, які потрібні Hibernate, і зберігає їх у локальному сховищі користувача. Maven 2 використовує центральний репозиторій за замовчуванням для пошуку бібліотек, але можна налаштувати репозиторії для використання (наприклад, приватні репозиторії компанії) у POM.

Основна відмінність між Maven і Ant полягає в тому, що дизайн Maven розглядає всі проекти як структуровані та мають певний набір робочих процесів (наприклад, отримання ресурсів із джерелами, компіляція проекту, модульне тестування тощо). В основі Maven лежать конвенції щодо структури проекту та робочих процесів, які зазвичай підтримуються в усіх проектах.

Для пошуку координат для різних бібліотек і фреймворків з відкритим кодом можна використовувати пошукові системи, такі як The Central Repository Search Engine.

Проекти, створені на одній машині, можуть використовувати один одного через локальне сховище. Локальний репозиторій діє як кеш для завантажених залежностей і як централізоване місце зберігання для локально створених артефактів. Команда "mvn install" створює проект і розміщує його двійкові файли в локальному сховищі. Тоді інші проекти можуть використовувати цей проект, вказавши його координати у своїх POM.

3.1.4 База даних PostgreSQL

PostgreSQL - це потужна система управління об'єктно-реляційними базами даних з відкритим вихідним кодом, яка розширює мову SQL та забезпечує безпечне зберігання та масштабування навіть найскладніших обсягів даних. Історія PostgreSQL налічує понад 30 років розвитку, вона бере свій початок з проекту POSTGRES, розпочатого в Каліфорнійському університеті в Берклі в 1986 році.

PostgreSQL завдяки своїй перевіреній архітектурі, надійності, цілісності даних та розширюваності здобув собі міцну репутацію. Він поставляється з надійним набором функцій і користувацькими можливостями, що допомагають розробникам створювати програми і адміністраторам забезпечувати цілісність даних та надійність середовища. PostgreSQL підтримує стандарт ACID з 2001 року.

Окрім того, PostgreSQL є дуже розширюваним. Ви можете визначати власні типи даних, створювати власні функції та навіть використовувати різні мови програмування для розробки без перекомпіляції бази даних. При цьому PostgreSQL є безкоштовним та відкритим вихідним кодом, і він підтримується активною спільнотою, що надає постійні інновації та продуктивні рішення. PostgreSQL сумісний з усіма основними операційними системами і є відмінним вибором для управління даними будь-якого обсягу.

PostgreSQL старається дотримуватися стандартів SQL, якщо це не суперечить традиційним функціям або не може призвести до неправильних архітектурних

рішень. Багато функцій, які вимагаються стандартом SQL, підтримуються, хоча іноді у них може бути трохи відмінний синтаксис чи функціонал. З часом можна очікувати подальших кроків у напрямку більшої відповідності до стандартів.

Починаючи з версії 14, яка вийшла у вересні 2021 року, PostgreSQL відповідає щонайменше 170 з 179 обов'язкових функцій для відповідності до SQL:2016 Core.

3.1.5 Docker

Docker - це програмна платформа, що дозволяє швидко створювати, тестувати та розгортати програми. Docker упаковує програмне забезпечення у стандартизовані блоки, які називаються контейнерами. Кожен контейнер містить усе необхідне для функціонування програми, включаючи бібліотеки, системні інструменти, код та середовище виконання. Використовуючи Docker, ви можете швидко розгортати програми в будь-якому середовищі та бути впевненими, що ваш код буде працювати.

Контейнери - це невеликі та легкі середовища виконання, які спільно використовують ядро операційної системи, але працюють ізольовано один від одного. Хоча існували системи використання контейнерів у системах Linux і Unix протягом певного часу, Docker, відкритий проект, запущений у 2013 році, допоміг поширити цю технологію та спростив розробникам завдяки можливості "побудови один раз і запуску де завгодно".

3.1.6 Утиліта для створення та виконання запитів Postman

Postman - це комп'ютерний додаток, призначений для тестування API. Використовуючи Postman, користувач може надсилати запити до веб-серверів і отримувати відповіді для подальшого аналізу. Postman спрощує процес тестування API, оскільки не вимагає додаткового програмування чи налаштування фреймворку. Цей інструмент широко використовується тестувальниками та розробниками для забезпечення якості програмного забезпечення.

Postman має багато корисних функцій і інтуїтивний інтерфейс, який дозволяє легко створювати та надсилати запити. Користувачу достатньо заповнити необхідні дані, вибрати метод HTTP і натиснути кнопку "Відправити". Крім того, Postman підтримує автоматизацію тестування, що дозволяє налаштовувати тести та створювати набори тестів для автоматичного виконання. Цей інструмент користується популярністю серед мільйонів тестувальників завдяки своїй зручності та функціональності.

3.1.7 Swagger

Swagger — це набір інструментів для розробників API від SmartBear Software і колишня специфікація, на якій базується специфікація OpenAPI.

Використання інструментів Swagger з відкритим вихідним кодом можна розділити на різні випадки використання: розробка, взаємодія з API та документація.

Розробка API

Під час створення API можна використовувати інструменти Swagger для автоматичного створення документа Open API на основі самого коду. Це вбудовує опис API у вихідний код проекту та неофіційно називається розробкою API за принципом коду або знизу вгору.

Крім того, використовуючи Swagger Codegen, розробники можуть відокремити вихідний код від документа Open API і створити код клієнта та сервера безпосередньо з дизайну. Це дає змогу відкласти аспект кодування.

Взаємодія з API

Використовуючи проект Swagger Codegen, кінцеві користувачі генерують клієнтські SDK безпосередньо з документа OpenAPI, зменшуючи потребу в створеному людиною клієнтському коді. Станом на серпень 2017 року проект Swagger Codegen підтримував понад 50 різних мов і форматів для створення клієнтського SDK.

Документування API

Якщо описано в документі OpenAPI, інструменти Swagger з відкритим вихідним кодом можуть використовуватися для безпосередньої взаємодії з API через інтерфейс користувача Swagger. Цей проект дозволяє підключатися безпосередньо до живих API через інтерактивний інтерфейс користувача на основі HTML. Запити можна робити безпосередньо з інтерфейсу користувача та параметрів, досліджених користувачем інтерфейсу.

3.1.8 KeyCloak

Keycloak - це продукт з відкритим вихідним кодом, який призначений для впровадження механізму одноразового входу з можливістю управління ідентичністю та керування доступом, спрямований на потреби сучасних застосунків і сервісів. За даними на 2018 рік, цей проект спільноти JBoss перебуває під управлінням компанії Red Hat, яка використовує його як джерело для свого продукту RH-SSO. Мета Keycloak - забезпечити можливість створення безпечних застосунків та сервісів з мінімальним написанням коду для реалізації процесів аутентифікації та авторизації.

Серед ключових функцій Keycloak можна визначити:

- Реєстрацію користувачів
- Вхід за допомогою облікових записів у соціальних мережах
- Одноразовий вхід та вихід для всіх застосунків одного реалму (Single Sign-On/Sign-Off)
- Видачу JSON Web Token для автентифікованих облікових записів
- Двофакторну аутентифікацію
- Інтеграцію з LDAP
- Брокер Kerberos
- Підтримку багатоокремих реалмів з можливістю налаштування зовнішнього вигляду сторінки входу для кожного реалму [26].

3.2 Розробка програмного забезпечення для сервісу автентифікації

Перш за все, треба створити проект на веб-сайті https://start.spring.io/. Для цього оберіть мову програмування Java, виберіть проект Maven, оберіть актуальну версію Spring Boot та Java. Потім заповніть всю мета-інформацію про проект, як показано на рисунку (див. рисунок 3.2).

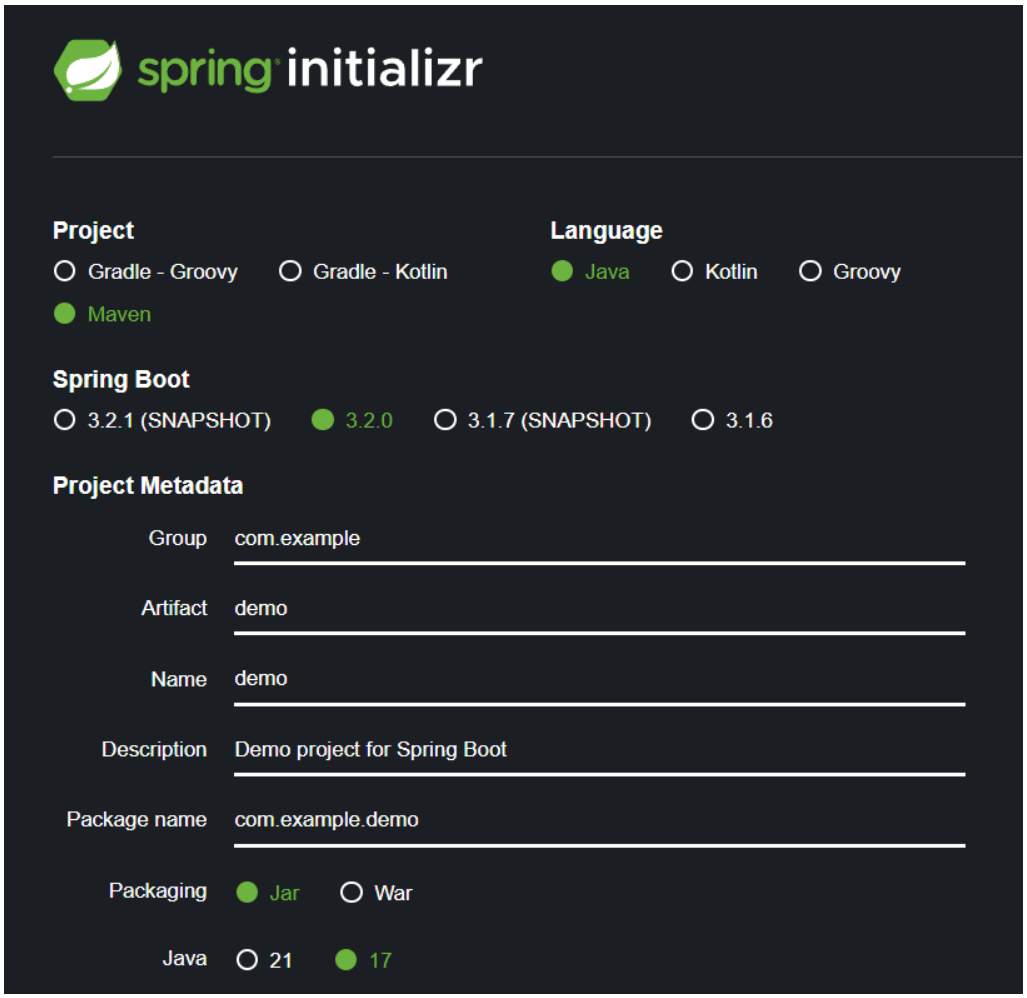

Рисунок 3.2 Мета інформація

Після цього виберіть залежності, які вам потрібні для проекту (див. рисунок 3.3).

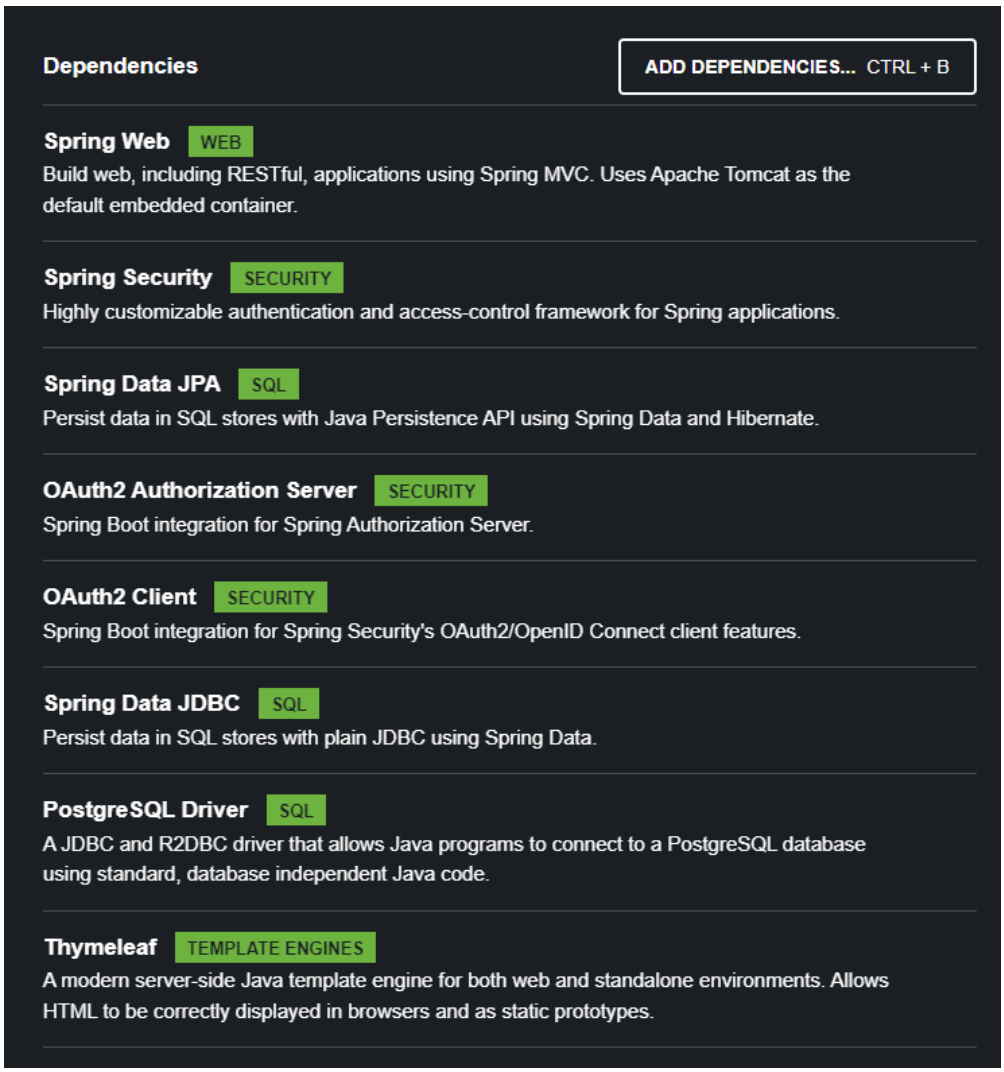

Рисунок 3.3 Залежності у проекті

Після вибору залежностей натискайте кнопку «Generate» та зберігайте отриманий архів з проектом, який потрібно розпакувати та відкрити в IntelliJ IDEA.

Далі вам потрібно додати сторонні залежності до файлу pom.xml (див. рисунок 3.4):

```xml
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>4.23.0</version>
</dependency>
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.1.0</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>21.0.0</version>
</dependency>
```

Рисунок 3.4 pom.xml

Також важливо додати конфігурацію сервера до файлу конфігурації проекту. Тут ви можете налаштувати доступ зчитувача JSON повідомлень, вказати порт сервера, на якому він буде працювати, і встановити параметри доступу до бази даних, зокрема, PostgreSQL у нашому випадку. Крім того, у цьому файлі можна визначити властивості для конфігурації keycloak (див. рисунок 3.5).

```
spring.profiles.active=local

spring.datasource.url=jdbc:postgresql://localhost:5432/testMyIt
spring.datasource.username=postgres
spring.datasource.password=root

spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
spring.liquibase.change-log=classpath:db/liq/changelog.xml

keycloak.authServerUrl = ${keycloak_url}
keycloak.realm = ${keycloak_realm}
keycloak.clientId = ${keycloak_id}
keycloak.clientSecret = ${keycloak_secret}
keycloak.authType = client_credentials
```

Рисунок 3.5 Файл конфігурації проекту

Згідно до завдання створимо структуру проекту, як показано на рисунку 3.6:

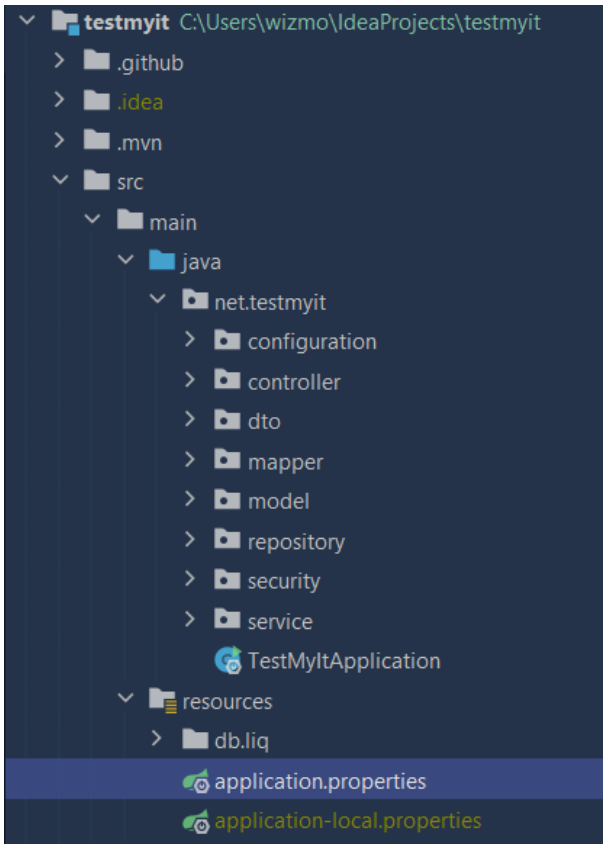

Рисунок 3.6 Структура проекту

Зібрати проект у виконуючий файл можна за допомогою команди «maven build». Ця команда компілює код у виконуючий код та створює jar-архів, який можна запустити на будь-якому комп'ютері (див. рисунок 3.7).

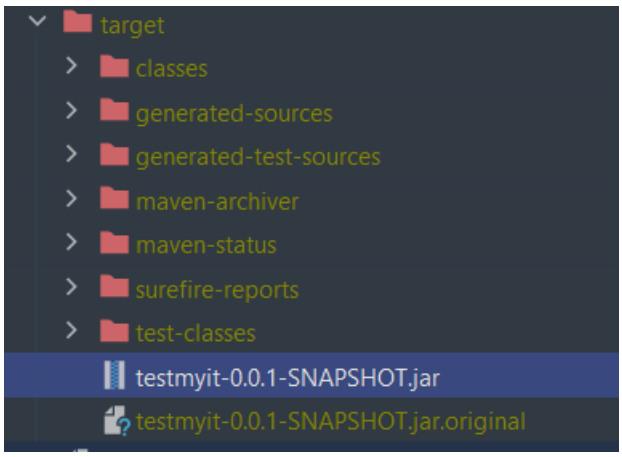

Рисунок 3.7 jar-архів з компільованим кодом

3.2.1 Конфігурація додатка

У проекті знаходиться декілька класів, які конфігурують наш додаток. Перший з них відповідає за конфігурацію keycloak . Другий відповідає за створення серверу автентифікації (рисунки 3.8 та 3.9).

```java
@Getter
@Setter
@ConfigurationProperties(prefix = "keycloak")
@Configuration
public class KeycloakConfiguration {
    private String authServerUri;
    private String realm;
    private String clientId;
    private String clientSecret;
    private String tokenUri;

    @Bean
    public Keycloak keycloak(){
        return KeycloakBuilder.builder()
                .serverUrl(authServerUri)
                .realm(realm)
                .clientId(clientId)
                .clientSecret(clientSecret)
                .grantType(CLIENT_CREDENTIALS)
                .build();
    }
}
```

Рисунок 3.8 Конфігурацію Keycloak

```java
package net.testmyit.configuration;

import ...

@Configuration
public class WebClientConfiguration {

    @Value("${KEYCLOAK_URI}")
    private String authServerUri;

    @Bean
    public WebClient keycloakWebClient() {
        return WebClient.builder()
                .baseUrl(authServerUri)
                .build();
    }

}
```

Рисунок 3.9 Серверу автентифікації

3.3 Налаштування KeyCloak

Перш ніж почати переконайтеся, що у вас встановлено Docker. Запустіть Keycloak З терміналу введіть таку команду, щоб запустити Keycloak:

docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:23.0.0 start-dev

Ця команда запускає Keycloak, відкритий на локальному порту 8080, і створює початкового користувача адміністратора з іменем користувача admin і паролем admin. Якщо у вас немає image KeyCloak то Docker завантажить його самостійно як показано на рисунку 3.10

```
PS C:\Users\wizmo> docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/key
cloak:23.0.0 start-dev
Unable to find image 'quay.io/keycloak/keycloak:23.0.0' locally
23.0.0: Pulling from keycloak/keycloak
133efa1a6421: Pull complete
021b45597fb2: Downloading [==============================================>    ]  71.85MB/77.15MB
8feeb8866cf2: Downloading [==================>                                ]  67.57MB/177.8MB
5e70055f9013: Download complete
```

Рисунок 3.10 Автоматичне завантаження KeyCloak

```
2023-11-28 09:37:12,184 INFO  [org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (main) No
de name: node_515646, Site name: null
2023-11-28 09:37:12,308 INFO  [org.keycloak.broker.provider.AbstractIdentityProviderMapper] (main) Registering class org
.keycloak.broker.provider.mappersync.ConfigSyncEventListener
2023-11-28 09:37:12,348 INFO  [org.keycloak.services] (main) KC-SERVICES0050: Initializing master realm
2023-11-28 09:37:14,014 INFO  [io.quarkus] (main) Keycloak 23.0.0 on JVM (powered by Quarkus 3.2.9.Final) started in 19.
142s. Listening on: http://0.0.0.0:8080
2023-11-28 09:37:14,015 INFO  [io.quarkus] (main) Profile dev activated.
2023-11-28 09:37:14,015 INFO  [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-h2, jdbc-mariadb
, jdbc-mssql, jdbc-mysql, jdbc-oracle, jdbc-postgresql, keycloak, logging-gelf, micrometer, narayana-jta, reactive-route
s, resteasy-reactive, resteasy-reactive-jackson, smallrye-context-propagation, smallrye-health, vertx]
2023-11-28 09:37:14,425 INFO  [org.keycloak.services] (main) KC-SERVICES0009: Added user 'admin' to realm 'master'
2023-11-28 09:37:14,430 WARN  [org.keycloak.quarkus.runtime.KeycloakMain] (main) Running the server in development mode.
 DO NOT use this configuration in production.
```

Рисунок 3.11 Запуск KeyCloak

1. Увійдіть до консолі адміністратора
2. Перейдіть до консолі адміністратора Keycloak.
3. Увійдіть за допомогою імені користувача та пароля, які ви створили раніше.

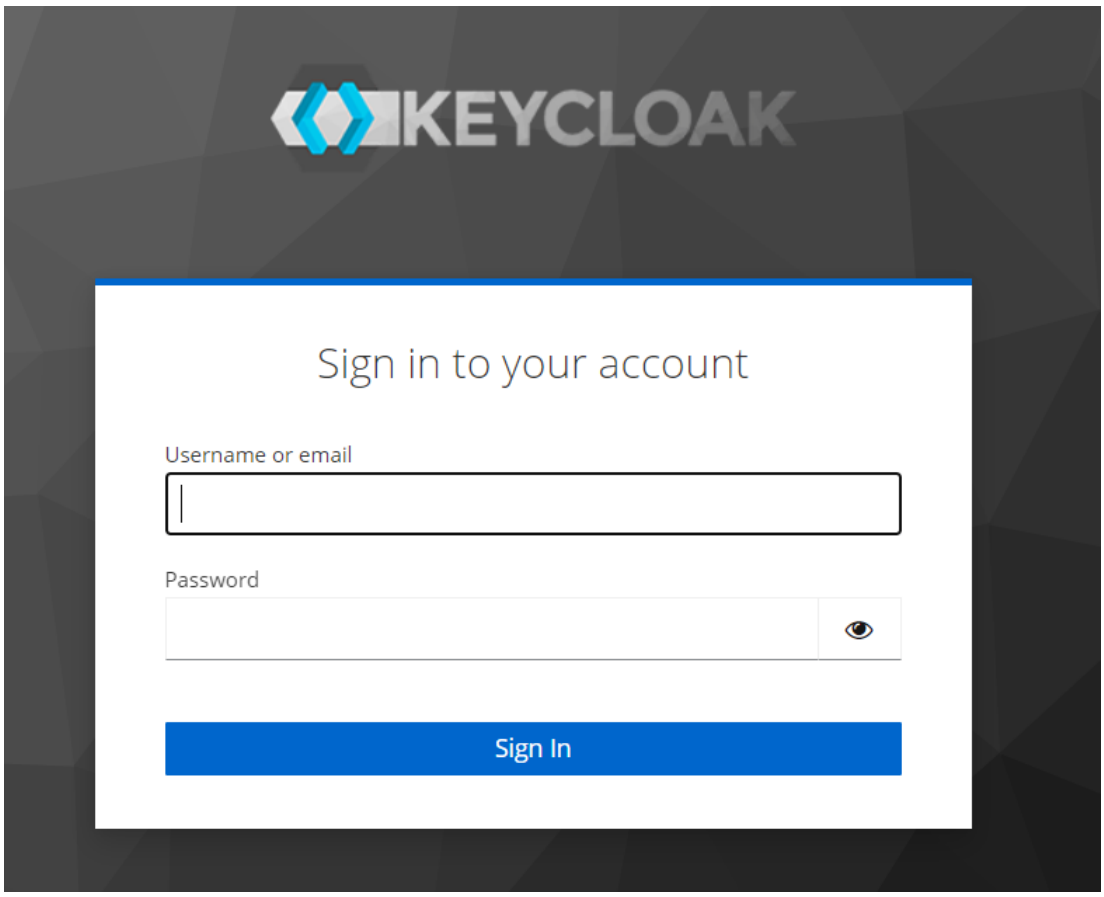

Рисунок 3.12 Вхід до консолі адміністратора KeyCloak

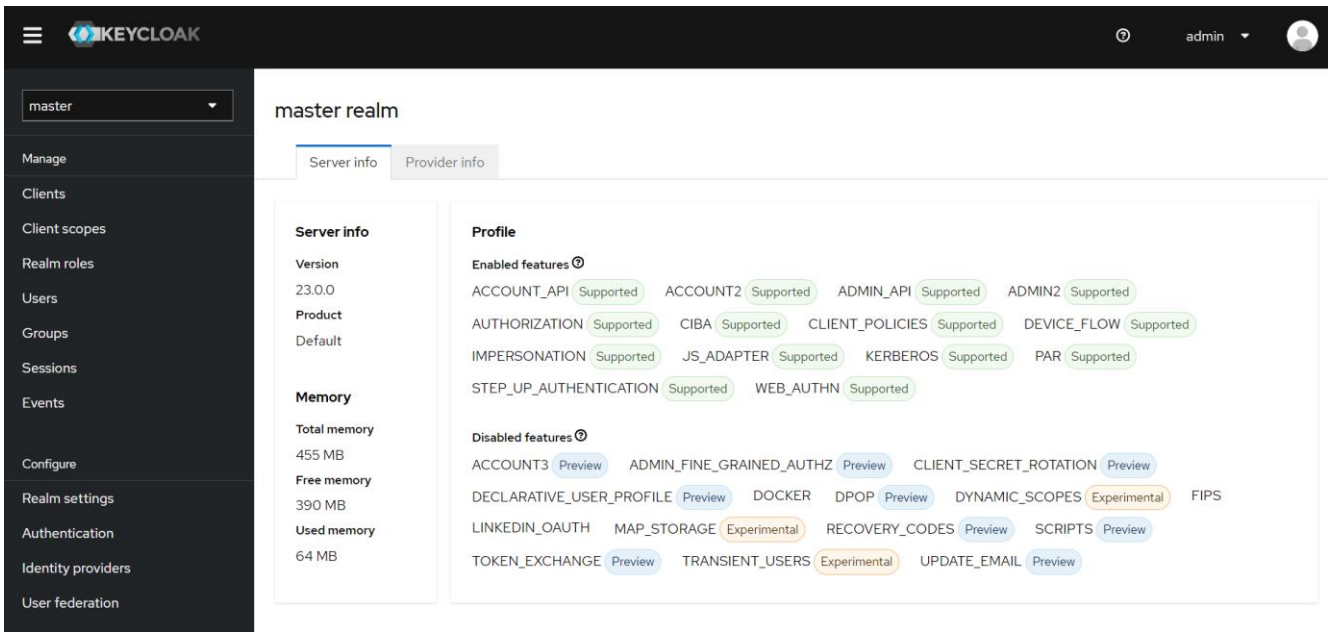

Рисунок 3.13 KeyCloak main page

3.3.2 Створення Realm у Keycloak.

Realm у Keycloak еквівалентний орендарю. Кожна область дозволяє адміністратору створювати ізольовані групи програм і користувачів. Спочатку Keycloak містить єдину сферу, яка називається master. Використовуйте цю область лише для керування Keycloak, а не для керування програмами. Використовуйте ці кроки, щоб створити перший realm.

1. Відкрийте консоль адміністратора Keycloak.
2. Клацніть master слова у верхньому лівому куті, а потім натисніть «Create realm».
3. Введіть myrealm у поле Realm name.
4. Натисніть Create.

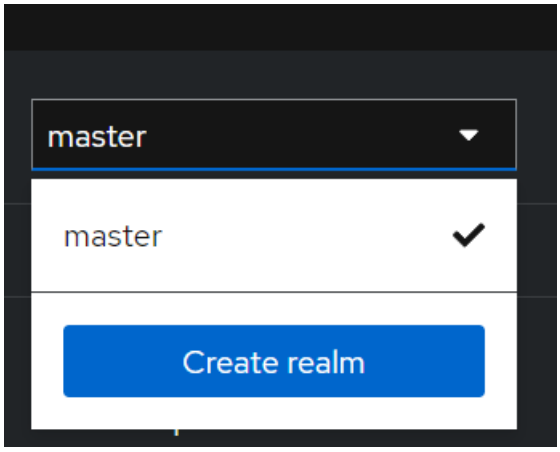

Рисунок 3.14 Крок 2

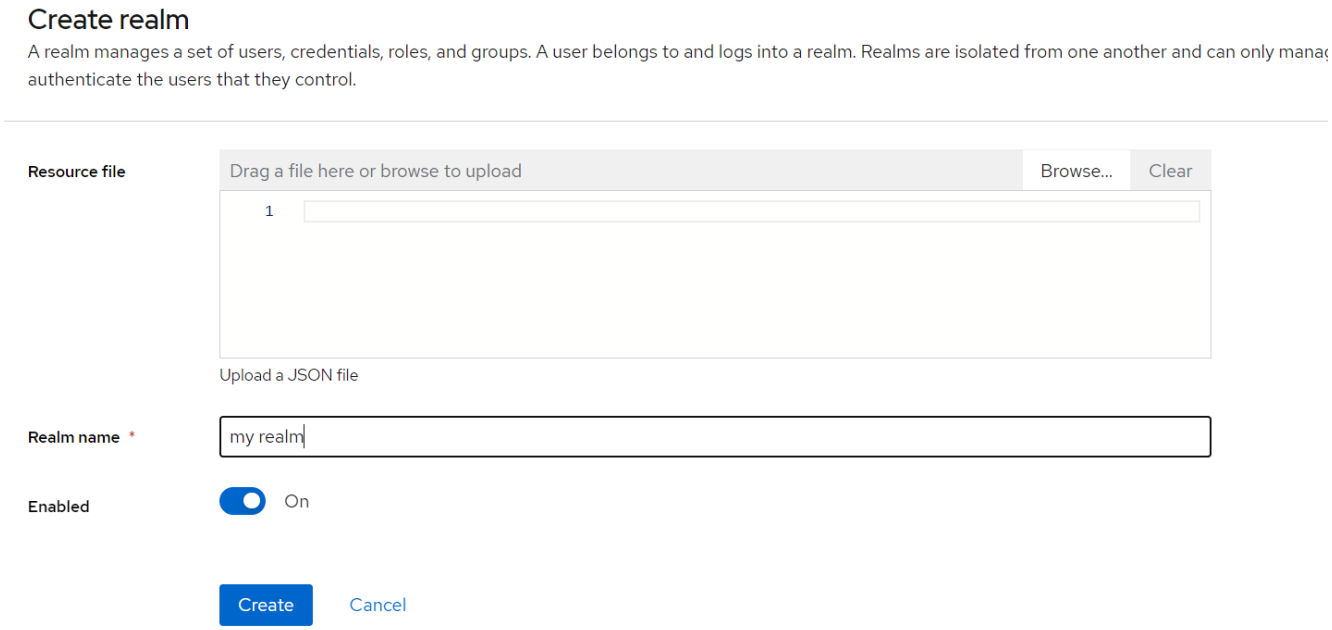

Рисунок 3.15 Крок 3

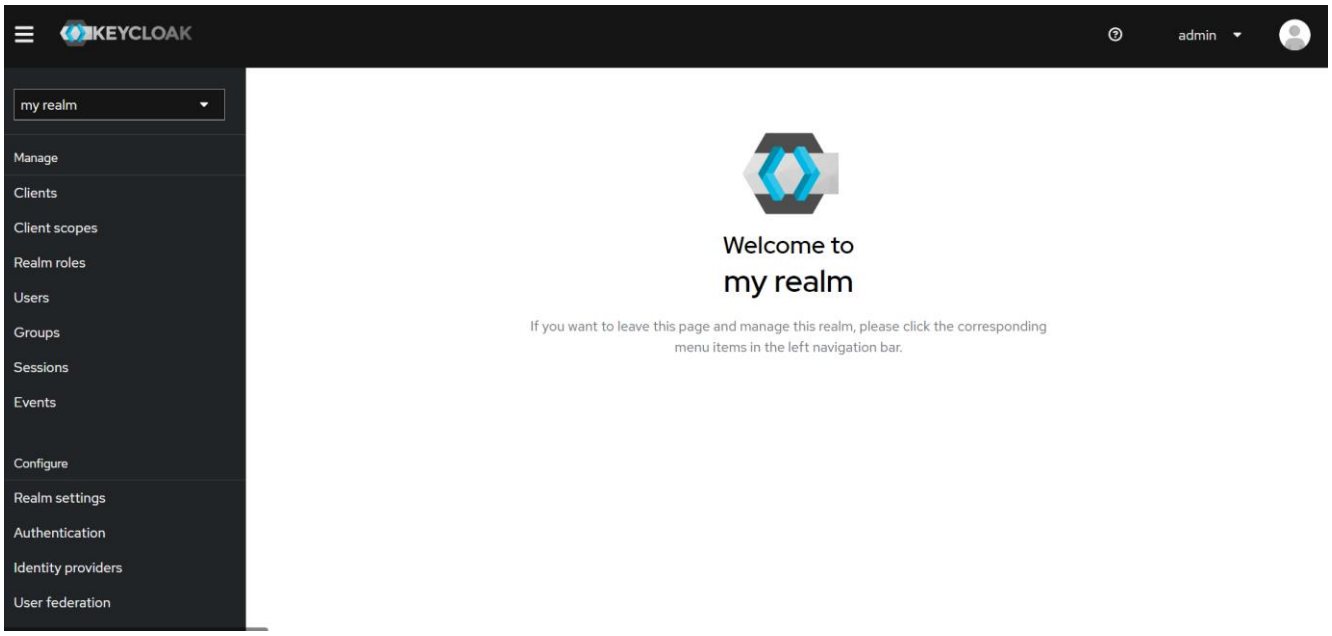

Рисунок 3.16 Крок 4

3.3.3Створити користувача

1. Спочатку realm не має користувачів. Щоб створити користувача, виконайте такі дії:

2. Відкрийте консоль адміністратора Keycloak.
3. Клацніть слово master у верхньому лівому куті, а потім клацніть myrealm. Натисніть Користувачі в меню ліворуч.
4. Натисніть Додати користувача.
5. Заповніть форму такими значеннями:
   - Ім’я користувача: myuser
   - Ім’я: будь-яке ім’я
   - Прізвище: будь-яке прізвище
6. Натисніть «Create».

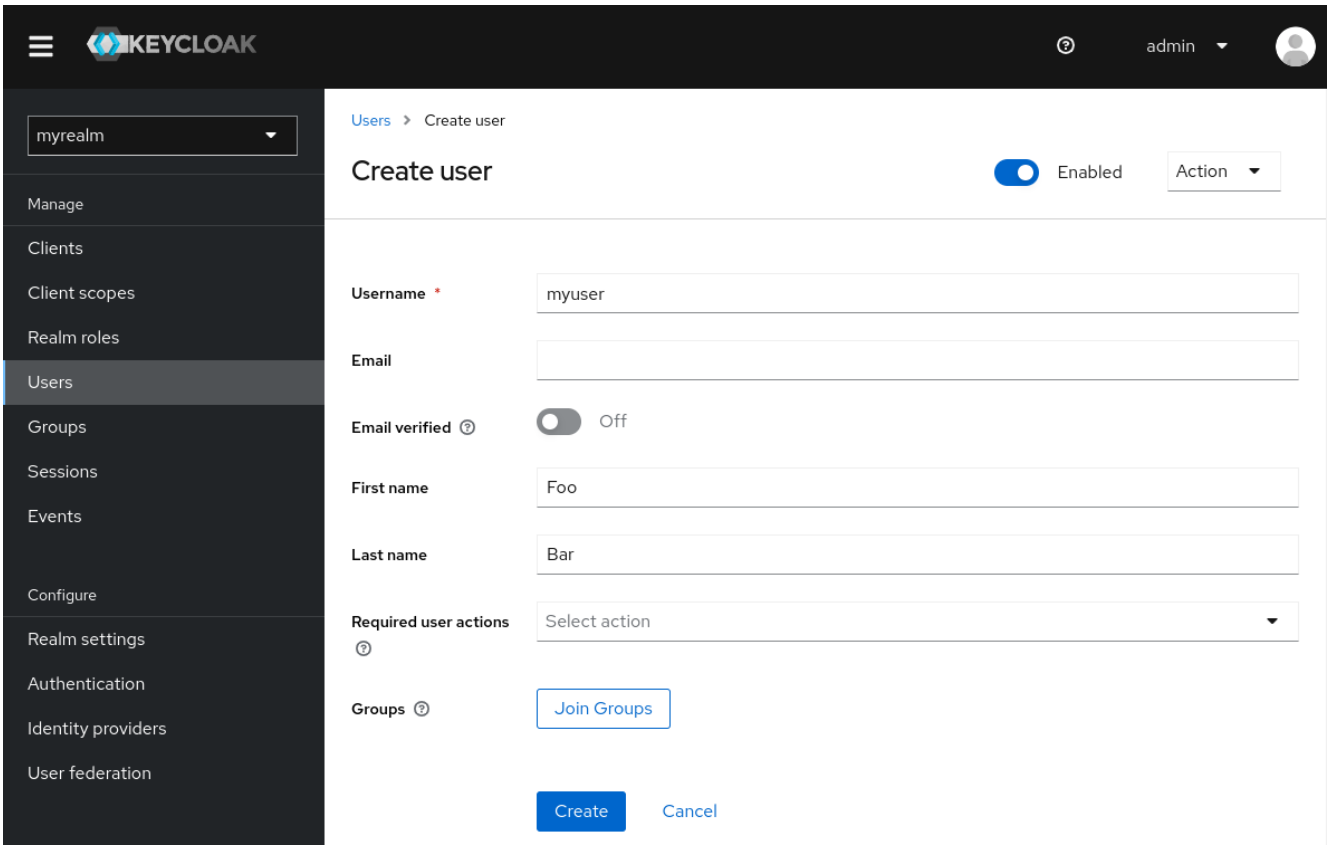

Рисунок 3.17 Створення користувача у KeyCloak

Цьому користувачеві потрібен пароль для входу.

1. Щоб установити початковий пароль: натисніть «Credentials» у верхній частині сторінки.
2. Заповніть форму встановлення пароля з паролем.
3. Щоб користувачеві не потрібно було оновлювати цей пароль під час першого входу, установіть для параметра «Temporary» значення «off.».

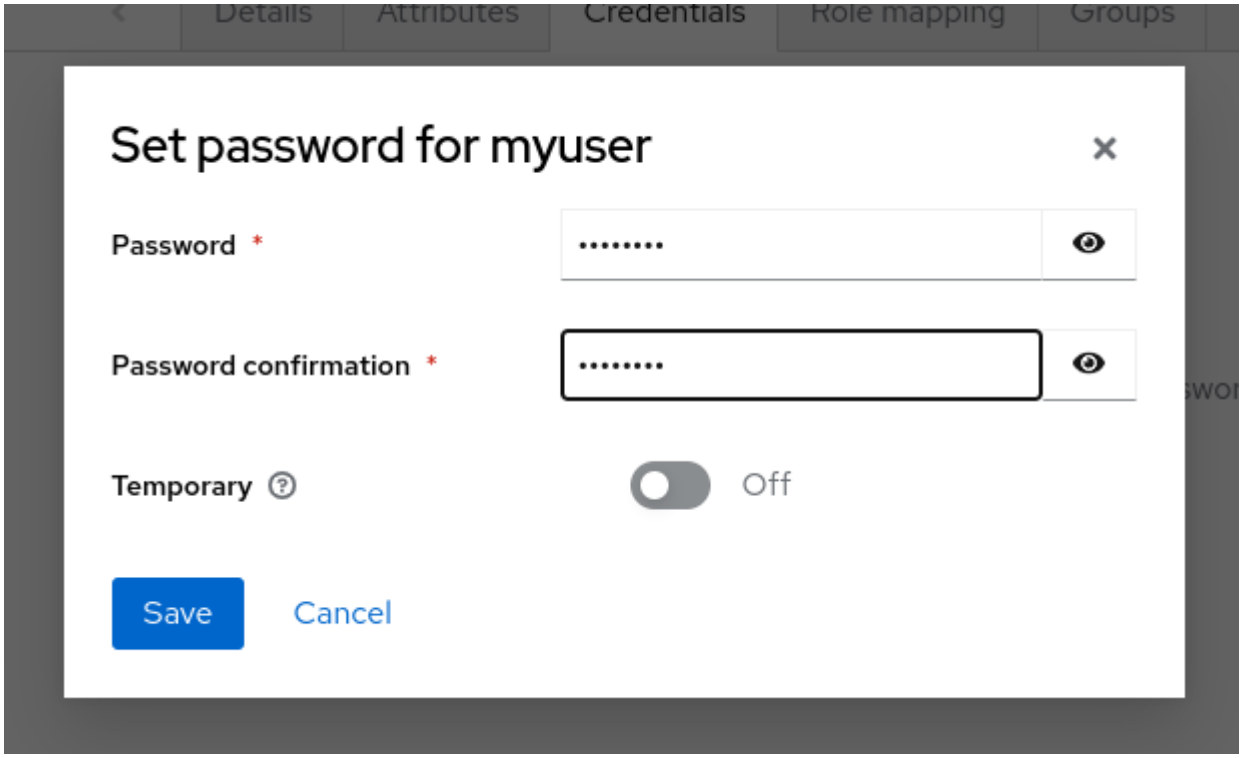

Рисунок 3.18 Задавання паролю для користувача

Увійдіть до консолі облікового запису

Тепер ви можете увійти до консолі облікового запису, щоб перевірити, чи правильно налаштовано цього користувача.

1. Відкрийте консоль облікового запису Keycloak.
2. Увійдіть за допомогою myuser і пароля, який ви створили раніше.

Як користувач Консолі облікового запису ви можете керувати своїм обліковим записом, зокрема змінювати свій профіль, додавати двофакторну автентифікацію та включати облікові записи постачальників ідентифікаційних даних.

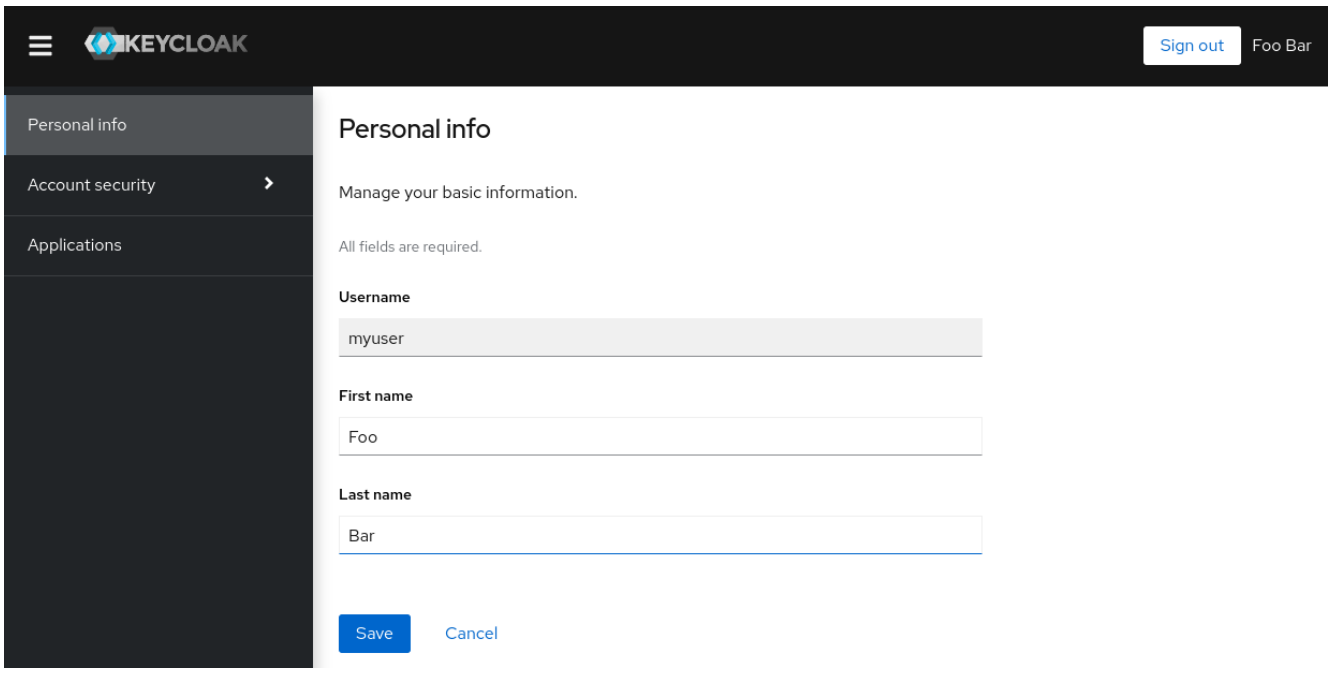

Рисунок 3.19 Базова інформація про користувача

Захистіть першу програму

Щоб захистити першу програму, ви починаєте з реєстрації програми у своєму realm Keycloak:

1. Відкрийте консоль адміністратора Keycloak.
2. Клацніть слово master у верхньому лівому куті, а потім клацніть myrealm.
3. Натисніть Клієнти.
4. Клацніть «Створити клієнт».
5. Заповніть форму такими значеннями:
6. Тип клієнта:
    - OpenID Connect
    - Client ID: my client

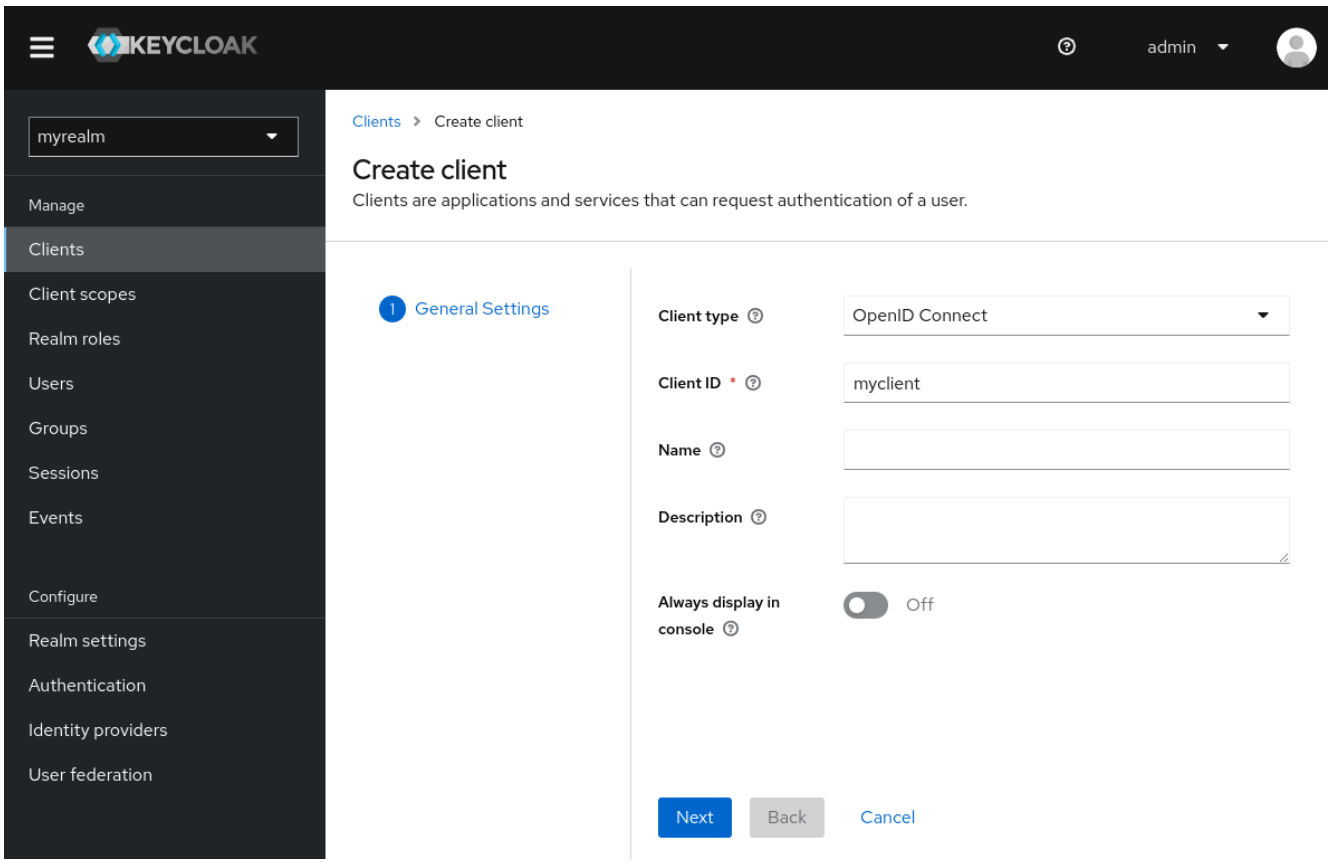

Рисунок 3.20 Створення клієнту

7. Натисніть «Далі».
8. Підтвердьте, що стандартний потік увімкнено.
9. Натисніть Далі.
    - Valid redirect URIs to https://www.keycloak.org/app/*
    - Set Web origins to https://www.keycloak.org
10. Натисніть зберегти

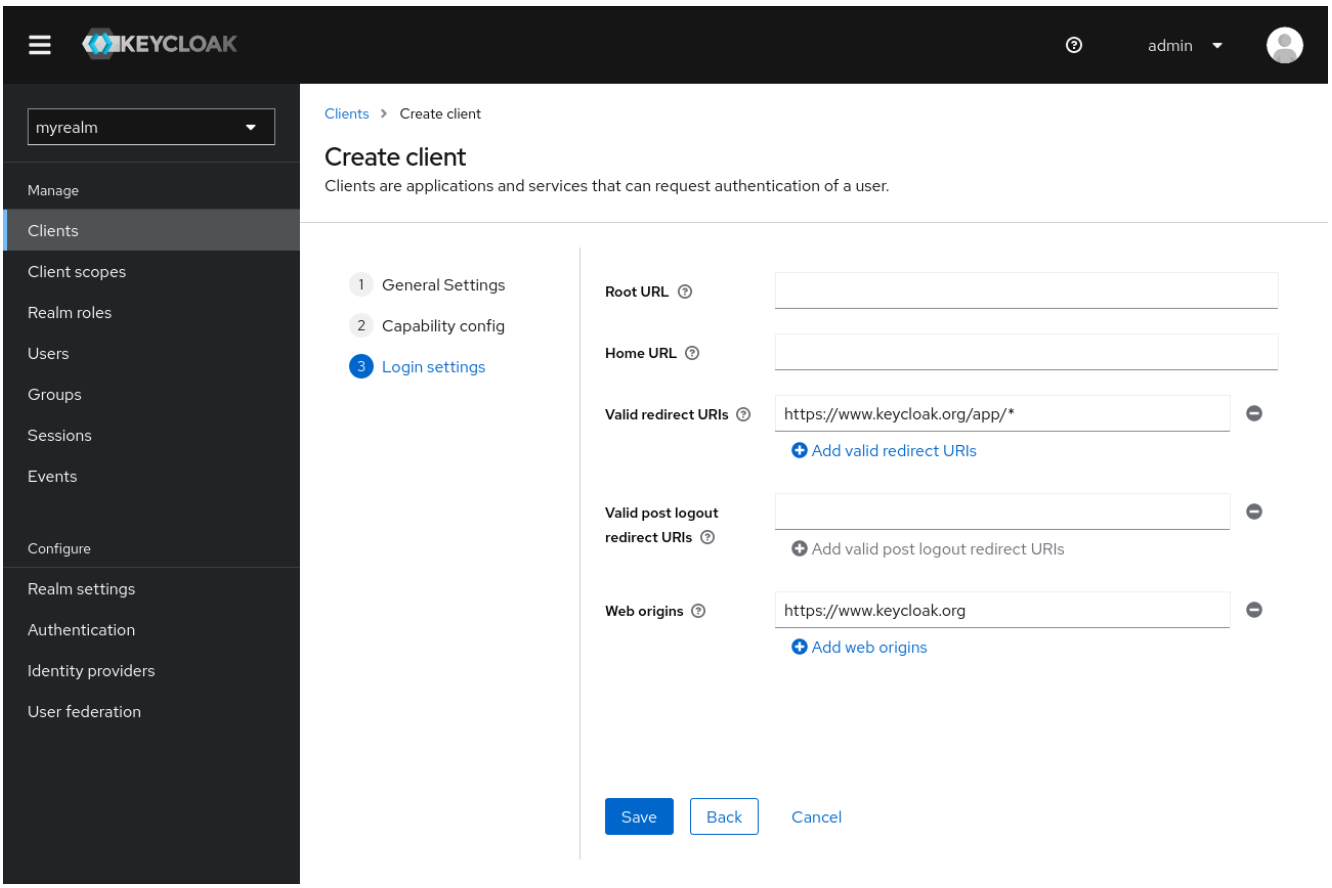

Рисунок 3.21 Валідний URL

Щоб підтвердити успішне створення клієнта, ви можете скористатися програмою тестування SPA на веб-сайті Keycloak.

1. Відкрийте https://www.keycloak.org/app/.
2. Натисніть «Зберегти», щоб використовувати конфігурацію за замовчуванням.
3. Натисніть «Увійти», щоб автентифікуватися в цій програмі за допомогою сервера Keycloak, який ви запустили раніше.

3.2.4 Модуль користувачів

Для того, щоб створити користувача, потрібно відправити POST-запит на ендпоінт /api/users. Це може бути використано для реєстрації нових користувачів (див. рисунок 3.22).

```
15
16      @PostMapping
17      public UserResponseDto createUser(@RequestBody @Valid UserRequestDto userRequest) {
18          return userService.createUser(userRequest);
19      }
```

Рисунок 3.22 Create User endpoint

Також наявні ендпоінти для отримання всіх користувачів та конкретного користувача за допомогою ідентифікатора (рисунок 3.23).

```
20
21      @GetMapping("/{id}")
22      public UserResponseDto getUser(@PathVariable Long id) { return userService.getUser(id); }
25
```

Рисунок 3.23 Get User endpoint

Також іноді потрібно вносити зміни до інформації про користувача, тому для цього спеціально створено ендпоінт /api/users, який обробляє PUT-запити (див. рисунок 3.24).

```
26      @PutMapping("/{id}")
27      public UserResponseDto updateUser(@PathVariable Long id, @RequestBody UserRequestDto userRequestDto) {
28          return userService.updateUser(id, userRequestDto);
29      }
30
```

Рисунок 3.24 Update user endpoint

Також створено ендпоінт /api/users, призначений для обробки DELETE-запитів. Цей ендпоінт використовується для видалення користувача з бази даних, щоб він більше не міг автентифікуватися за допомогою своїх ідентифікаційних даних (див. рисунок 3.25).

```
30
31      @DeleteMapping("/{id}")
32      public void deleteUser(@PathVariable Long id) { userService.deleteUser(id); }
35  }
```

Рисунок 3.25 Delete user endpoint

Всі ці ендпоінти захищені і тільки адміністратор має до них доступ.

3.3.2 Інструкція користувача

Для успішного запуску програми необхідно мати встановлену версію Java 14 на комп'ютері. Перед запуском також важливо переконатися, що база даних запущена, оскільки саме там будуть зберігатися дані про користувачів, клієнтів та токени. Для запуску бази даних PostgreSQL використовується утиліта Docker, і вона може бути запущена за допомогою команди "docker run -p 5432:5432 -e POSTGRES_PASSWORD=postgres --name test -d postgres".

Сервер можна запустити за допомогою команди "java -jar build/libs/*snapshot_name*". Також можна вказати POSTGRES_USERNAME та

POSTGRES_PASSWORD як змінні середовища, щоб змінити логін та пароль для підключення до бази даних. Після запуску сервера, додаток буде доступний за адресою http://<ip_adress>:9090, де ip_adress – це IP-адреса комп'ютера (див. рисунок 3.26).

Рисунок 3.26 Запуск проекту

Далі переходимо у swagger і робимо пост запит для перевірки створення користувача (рисунок 3.27)

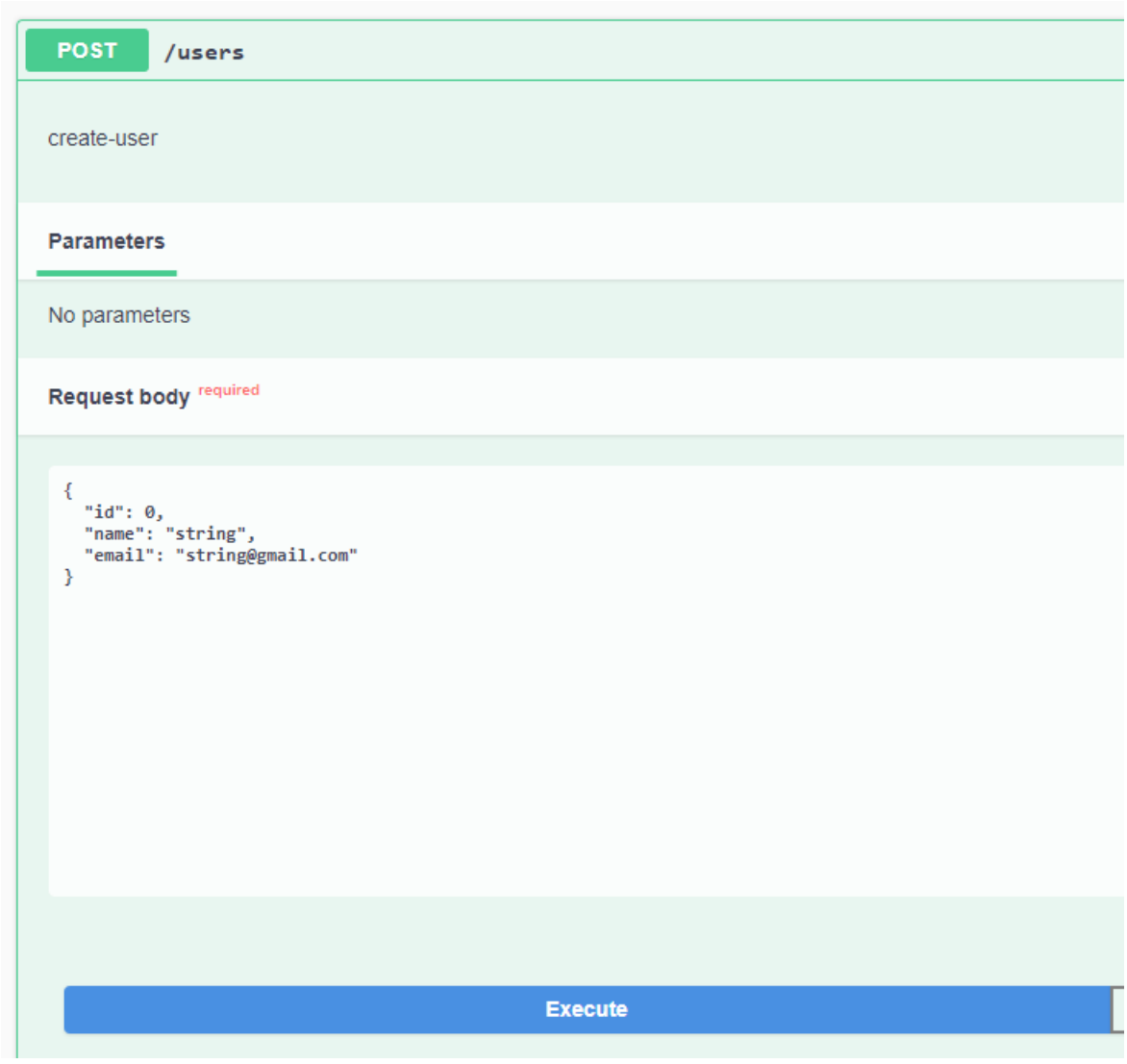

Рисунок 3.27 Пост запит для перевірки створення користувача

Отримуємо таку відповідь (рисунок 3.28)

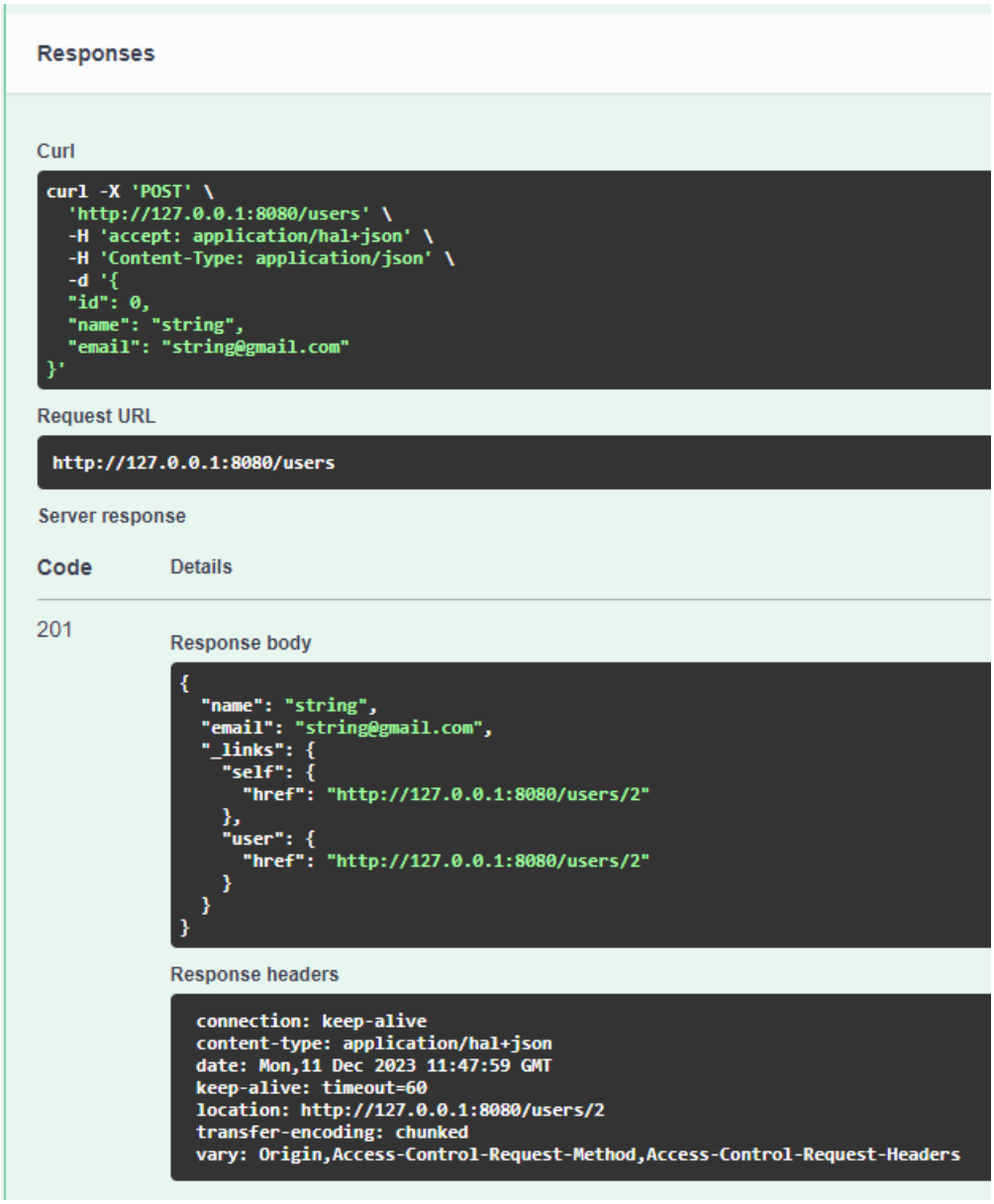

Рисунок 3.29 Вдповідь на post запит

Перевіряємо у KeyCloak створення користувача (рисунок 3.30)

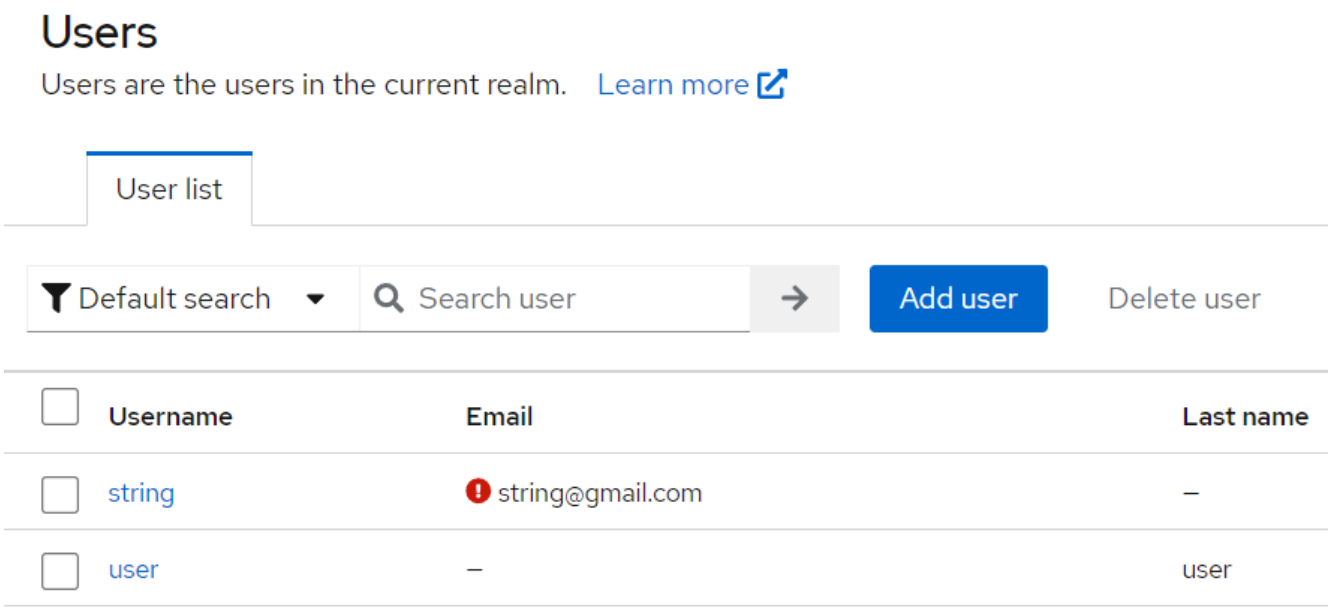

Рисунок 3.30 Створений користувач у KeyCloak

Також перевіряємо чи зможемо ми отримати цього користувача через Get запит (рисунок 3.31)

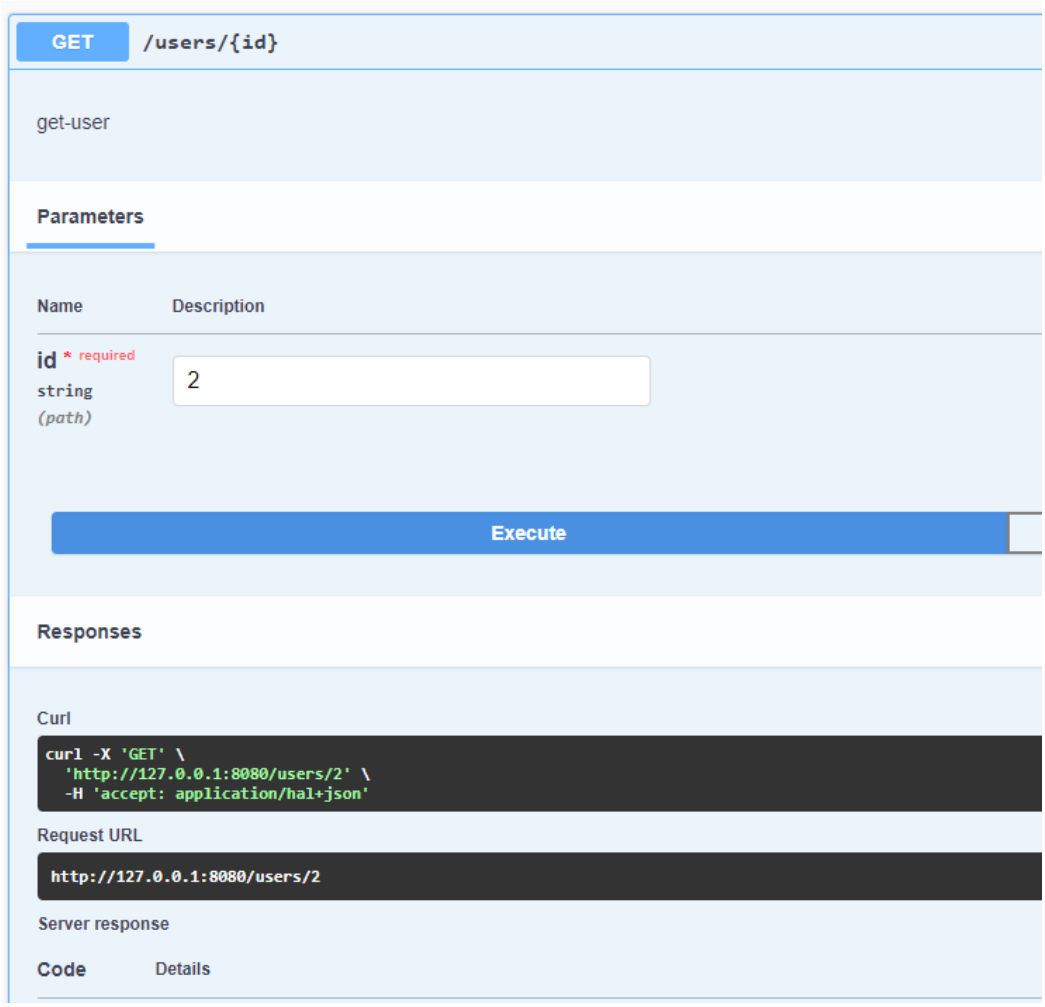

Рисунок 3.31 Get запит у Swagger

Отримуємо таку відповідь від сервера, де бачимо нашого користувача у відповідь на Get запит по id (рисунок 3.32)

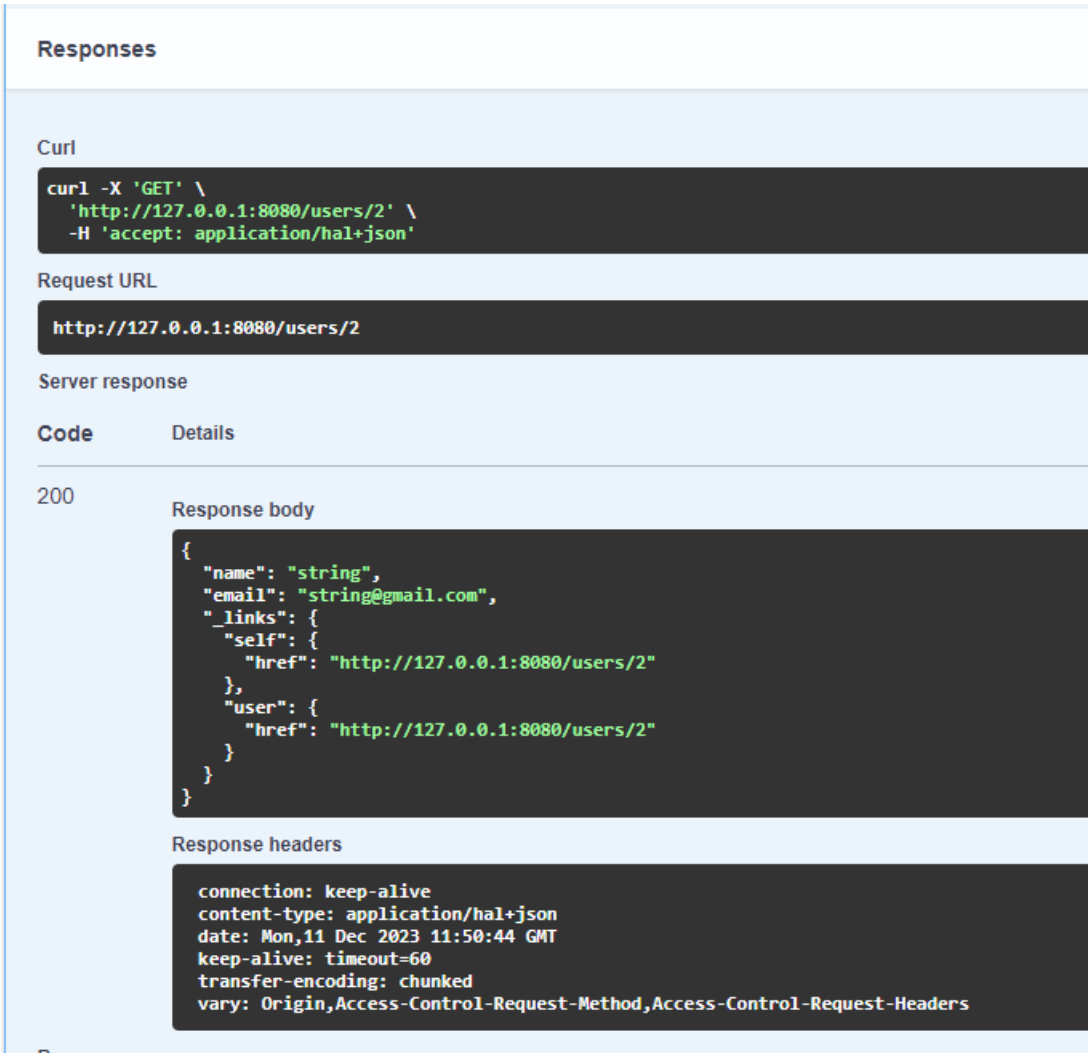

Рисунок 3.32 Відповідь на Get запит

ВИСНОВКИ

У першому розділі дослідження було проведено аналіз існуючих протоколів авторизації та їх порівняльну характеристику. Теоретичні відомості щодо концепції роботи протоколів авторизації для користувачів в веб-додатках та інших програмах створили основу для формування основних принципів та методів проведення даної курсової роботи. Головними цілями аналізу існуючих рішень в галузі авторизації користувачів та надання рекомендацій і розробки нових підходів та алгоритмів є оцінка поточного рівня захищеності інформаційної системи, перевірка відповідності інформаційної системи існуючим стандартам в галузі інформаційної безпеки, а також аналіз ризиків, пов'язаних із можливими загрозами безпеці ресурсів та розробка рекомендацій щодо впровадження нових механізмів безпеки та підвищення ефективності існуючих. Для цього дослідження використовується комбінований метод, який найкращим чином відображає рівень захищеності веб-ресурсу.

У розділі 2 було проведено аналіз роботи протоколів авторизації користувачів у веб-додатках та інших програмних застосунках, що послужило основою для розробки рекомендацій щодо покращення існуючих рішень. Основні цілі цього аналізу включають оцінку рівня безпеки інформаційної системи, визначення відповідності системи існуючим стандартам і оцінку ризиків, пов'язаних з можливими загрозами безпеці ресурсів.

Результати аналізу стали підґрунтям для розробки створено програмного забезпечення для сервісу автентифікації, базованого на протоколі OAuth 2.0. Розроблена програма здійснює автентифікацію користувачів за допомогою REST-запитів.

Робота включала в себе наступні етапи:

- Аналіз протоколів OAuth та OpenID Connect.
- Розгляд та аналіз видів автентифікації.
- Аналіз і розгляд різних методів автентифікації.

- Вивчення технологій та середовища виконання.

- Розробка програмного забезпечення для сервера автентифікації, базованого на протоколі OAuth 2.0.

- Практичне тестування розробленого додатка.

Отримані результати сприятимуть покращенню процесу автентифікації та забезпечать безпеку користувачів у системі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Single sign-on: Beginner's Guide, 2017. – 124 с. – (CreateSpace Independent Publishing Platform). – (ISBN 978-1978211865).
2. Martin T. Identification vs Authentication Authorization [Електронний ресурс] / Thoma Martin. – 2020. – Режим доступу до ресурсу: https://medium.com/plain-and-simple/identification-vs-authentication-vsauthorization-e1f03a0ca885.
3. Password Authentication for Web and Mobile Apps: The Developer's Guide To Building Secure User Authentication, 2020. – 142 с. – (Independently published). – (ISBN 979-8649303095).
4. Authentication: From Passwords to Public Keys – Addison-Wesley Professional, 2001. – 549 с. – (Addison-Wesley Professional). – (ISBN 978- 0201615999).
5. LeBlanc J. Identity and Data Security for Web Development: Best Practices / Jonathan LeBlanc., 2016. – 204 с. – ("O'Reilly Media, Inc."). – (ISBN- 10 : 1491937017).
6. Digital Certificates: Applied Internet Security, 1998. – 480 с. – (Addison-Wesley Professional). – (ISBN 978-0201309805).
7. Protocols for Authentication and Key Establishment 2 Edition, 2019. – 549 с. – (Springer) – (B081653251).
8. API Security in Action, 2020. – 576 с. – (Manning). – (ISBN 9781617296024).
9. Richer J. OAuth 2 in Action / J. Richer, A. Sanso., 2017, Manning – 400 с. – (ISBN 9781617293276).
10. OAuth 2.0 Simplified, 2018. – 240 с. – (Lulu.com). – (ISBN 1387751514).
11. Matthias B. OAuth 2.0: Getting Started in Web-API Security / Biehl 76 Matthias., 2015. – 86 с. – (CreateSpace Independent Publishing Platform). – (ISBN 1507800916).
12. Parecki A. OAuth 2.0 Servers / Aaron Parecki., 2016. – 350 с. – (O'Reilly Media). – (ISBN 149192098X).

13. Biehl M. OpenID Connect: End-user Identity for Apps and APIs / Matthias Biehl., 2019. – 137 c. – (CreateSpace Independent Publishing Platform). – (ISBN 1979718474).
14. Siriwardena P. OpenID Connect in Action / Prabath Siriwardena., 2020. – 400 c. – (Manning). – (SBN 9781617298974).
15. The OAuth 2.0 Authorization Framework [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6749#section-2.1.
16. M. Jones. The OAuth 2.0 Authorization Framework: Bearer Token Usage [Електронний ресурс] / M. Jones, D.Hardt. – 2012. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6750.
17. R. Fielding. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content [Електронний ресурс] / R. Fielding, J. Reschke. – 2014. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7231.
18. Discovering concrete attacks on websiteauthorization by formal analysis [Електронний ресурс] / Chetan Bansal, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Sergio Maffeis // IOS Press. – 2014. – Режим доступу до ресурсу: https://www.doc.ic.ac.uk/~maffeis/papers/jcs14.pdf.
19. Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations [Електронний ресурс] / Tim Polk, Kerry McKay, Santosh Chokhani, Santosh Chokhani // NIST Special Publication. – 2014. – Режим доступу до ресурсу: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf.
20. M. McGloin. OAuth 2.0 Threat Model and Security Considerations [Електронний ресурс] / M. McGloin, T. Lodderstedt, P. Hunt. – 2013. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6819.
21. B. Campbell. OAuth 2.0 Form Post Response Mode [Електронний ресурс] / B. Campbell, M. Jones. – 2015. Режим доступу до ресурсу: https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html. \

22. Douglas Purdy. Platform Updates: Operation Developer Love [Електронний ресурс] / Douglas Purdy // 2018 – Режим доступу до ресурсу: https://developers.facebook.com/blog/post/552/.
23. N. Sakimura. The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request [Електронний ресурс] / N. Sakimura, J. Bradley. – 2018. – Режим доступу до ресурсу: https://tools.ietf.org/html/draft-ietf-oauth-jwsreq-16.
24. OAuth 2.0 Token Binding [Електронний ресурс] / M. Jones, B. Campbell, J. Bradley, W. Denniss. – 2018. – Режим доступу до ресурсу: https://tools.ietf.org/html/draft-ietf-oauthtoken-binding-06.
25. Developer Survey Results 2019 [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2019.
26. Cloud Datastore [Електронний ресурс] // Google – Режим доступу до ресурсу: https://cloud.google.com/datastore/.