Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерної інженерії та управління _____
(повна назва)

Кафедра _____Безпеки інформаційних технологій _____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____

__Метод захисту веб-добатків на основі протоколів Oauth 2.0 і OpenID_____
(тема)

Виконав:

студент __2_ курсу, групи __БІКСм-20-1_____

____Холоша О.С._____

(прізвище, ініціали)

Спеціальність _____125 Кібербезпека _____

(код і повна назва спеціальності)

Освітня програма ___«Безпека інформаційних і комунікаційних систем»
___ (повна назва освітньої програми)

Керівник __доцент Северінов О.В._____

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____Халімов Г.З.__
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Безпеки інформаційних технологій_____

Рівень вищої освіти_____ другий (магістерський)_____

Спеціальність_____125 Кібербезпека _____
(код і повна назва)

Освітня програма_____«Безпека інформаційних і комунікаційних систем»___
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Холоші Олегу Сергійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи ____ Метод захисту веб-добатків на основі протоколів Oauth 2.0 і OpenID
_____
затверджена наказом по університету від _08_ ____11___2021 р. № __1685ст__ ___
2. Термін подання студентом роботи до екзаменаційної комісії _14_ грудня_____2021 р.
3. Вихідні дані до роботи _ документація протоколів та стандартів, статистичні дані щодо кіберінцидентів пов'язаних з автентифікацією і авторизацією у веб- додатках
_____
_____
_____
_____
_____
_____
_____
_____
_____

4. Перелік питань, що потрібно опрацювати в роботі _____
Види автентифікації.
Методи автентифікації у веб-додатках.
Аналіз вразливостей під час автентифікації.
Розробка розробка ПЗ сервера авторизації.
Створення документації щодо практичного використання ПЗ.
_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _презентаційний матеріал у вигляді слайдів _____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача завдання	1.09.2021	Виконано
2	Аналіз літературних джерел за отриманим завданням	1.09.2021 – 15.09.2021	Виконано
4	Аналіз протоколів авторизації та автентифікації	26.09.2021- 15.10.2021	Виконано
5	Розробка ПЗ сервера автентифікації	1.11.2021 – 20.11.2021	Виконано
6	Створення документації щодо описання принципу роботи сервера автентифікації	21.11.2021-29.11.2021	Виконано
7	Оформлення пояснювальної записки	30.11.2021-4.12.2021	Виконано
8	Здача на перевірку та підпис кваліфікаційної роботи керівнику	13.12.2021	Виконано
9	Проходження перевірки на плагіат та нормоконтроль кваліфікаційної роботи	15.12.2021	Виконано
10	Допуск завідувачем кафедри до захисту кваліфікаційної роботи	15.12.2021	Виконано
11	Захист кваліфікаційної роботи	17.12.2021	Виконано

Дата видачі завдання __1___ __вересня_____2021 р.

Студент _____
(підпис)

Керівник роботи _____ _____ доцент Северінов О.В._
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка включає в себе 76 сторінок, 54 рисунка, 14 джерел, 3 таблиці, 1 додаток.

ВЕБ-ДОДАТОК, МЕХАНІЗМ ЗАХИСТУ, АВТЕНТИФІКАЦІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, OAUTH, OPENID CONNECT

Об’єктом дослідження є протоколи авторизації та автентифікації Oauth2.0 і OpenID Connect.

Предметом дослідження є процес розробки програмного забезпечення сервера автентифікації на основі протоколу OpenID Connect.

Метою роботи є розробка програмного забезпечення сервера авторизації на основі протоколу OpenID Connect.

У роботі проведений аналіз сучасних механізмів авторизації та автентифікації у веб-додатках. Було проведено порівняння популярних механізмів авторизації та автентифікації, які зараз використовуються. Також було розроблено програмне забезпечення сервера авторизації на основі протоколу OpenID Connect.

ABSTRACT

The explanatory note includes 76 pages, 54 figures, 14 sources, 3 tables, 1 appendix.

WEB APP, SECURITY MECHANISM, AUTHENTICATION, SOFTWARE, OAUTH, OPENID CONNECT

The object of research is the authorization and authentication protocols Oauth2.0 and OpenID Connect.
The subject of the study is the process of developing authentication server software based on the OpenID Connect protocol.
The aim of the work is to develop software for the authorization server based on the OpenID Connect protocol.
The paper analyzes modern authorization and authentication mechanisms in web applications. A comparison was made of the currently used popular authorization and authentication mechanisms. Also, the authorization server software was developed based on the OpenID Connect protocol.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AOP - Aspect-oriented programming

API - Application Programming Interface

AS – Authorization Server

AWS – Amazon Web Services

EJB – Enterprise JavaBeans

FIDO - Fast IDentity Online

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IDE – Integrated development environment

IOC – Inversion of control

IP – Identity Provider

JAR – Java Archive

JDK – Java Development Kit

JEE - Java Enterprise Edition

JSON - JavaScript Object Notation

JTA – Java Transaction API

JVM - Java virtual machine

JWS - JSON Web Signature

JWT – JSON Web Token

LPT - Life Pro Tip

MFA - Multi-Factor Authentication

OIDC – OpenID Connect

OP – OpenID Provider

ORM - Object–relational mapping

OTP - One-time password

POJO – Plain Old Java Object

REST – Representational state transfer

RP – Resource Provider

SAML - Security Assertion Markup Language

SFA - Single-Factor Authentication

SMS - Short Message Service

SP –Service Provider

SQL - Structured Query Language

SSL - Secure Sockets Layer

SSO – Single sign-on

SWT - Simple Web Token

TLS - Transport Layer Security

URL - Uniform Resource Locator

USB – Universal Serial Bus

XML – Extensible Markup Language

ВСТУП

За останні десять років використання протоколів авторизації у веб-додатках значно поширилося. Більшість сервісів вимагають ідентифікацію користувачів у своїх системах для розділення прав доступу між приватними ресурсами, а також для отримання даних про користувача.

Збільшення використання функції авторизації та автентифікації в різних сервісах, порталах спричинило пожвавлення інтересу до таких протоколів авторизації та автентифікації, як OAuth 2.0 та OpenID, які дозволяють використовувати, так звану, віддалену авторизацію. При цьому акредитовані сервіси, які мають реалізацію цих протоколів надають доступ до своїх ресурсів та надають ідентифікацію своїх користувачів для інших систем. Наприклад, для реєстрації на форумі вже не виникає необхідності проходити процедуру автентифікації саме на цьому форумі й надавати сервісу для зберігання в базі даних свої логін та пароль. Використовуючи один з наведених вище протоколів можна вказати персональні дані, наприклад, від сервісу Google або Facebook й користуватися форумом [1].

Ці протоколи дуже гнучкі і дозволяють використовувати різні методи авторизації та автентифікації. Тобто ці протоколи можуть бути використані не тільки для веб-додатків, а також для мобільних та настільних додатків, що дозволяє використовувати їх в будь-яких додатках.

Мета роботи - аналіз та порівняння цих протоколів реалізація захисту веб додатка з використанням протоколу OpenID Connect. Мають бути розглянуті можливі атаки, та дані рекомендації, що допоможуть розробникам зменшити вплив загроз, та збитки при атаках.

Об’єктом дослідження є протоколи авторизаціі та автентифікації Oauth2.0 і OpenID Connect.

Предметом дослідження є процес розробки програмного забезпечення сервера автентифікації на основі протоколу OpenID Connect.

Завдання, які було поставлено у рамках цієї атесттитаційної роботи:

1) Дослідоити види там методи авторизації та автентифікації.

2) Дослідити та порівняти сучасні протоколи автентифікації та авторизації.

3) Розробити програмне забезпечення сервера автентифікації на основі протоколу OpenID Connect.

4) Створити документацію щодо використання розробленого програмного забезпечення.

1 СУЧАСНІ МЕТОДИ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ

1.1 Основні поняття

На сьогоднішній день існує багато способів авторизації та автентифікації користувачів у системах, проте спочатку необхідно ознайомитися з термінологією.

Ідентифікація — процедура розпізнавання користувача в системі як правило за допомогою наперед визначеного імені (ідентифікатора) або іншої апріорної інформації про нього(наприклад, адреса електронної пошти), яка сприймається системою.

Автентифікація — процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатором. За допомогою автентифікації друга сторона переконується, що суб'єкт дійсно той, за кого він себе видає.

Авторизація — перевірка, що вам дозволено доступ до запитуваного ресурсу.

Наприклад, при спробі потрапити в закритий клуб вас ідентифікують (запитають ваше ім'я і прізвище), автентифікують (попросять показати паспорт і звірять фотографію) і авторизують (перевірять, що прізвище знаходиться в списку гостей), перш ніж пустять усередину [2].

Аналогічно ці терміни вживаються в комп'ютерних системах, де традиційно під ідентифікацією розуміють отримання вашого профілю по логіну; під автентифікацією — перевірку, що ви знаєте пароль від цього облікового запису, а під авторизацією - перевірку вашої ролі в системі і рішення про надання доступу до запитаної сторінці або ресурсу.

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 процес перевірки ідентичності, який вимагає від запитувача доступу (може бути людиною, програмним забезпеченням або комп'ютером), щоб надати стороні, що автентифікує, єдиний ідентифікатор — один фактор — який пов'язаний з її ідентичністю. SFA використовується за замовчуванням у багатьох системах, оскільки його легко та дешево реалізувати.

Найпомітнішим однофакторним ідентифікатором є пароль. Інші зазвичай використовувані ідентифікатори включають SMS-код на зареєстрований мобільний пристрій, одноразовий пароль (OTP), згенерований фізичним пристроєм або програмним забезпеченням, запущеним на мобільному пристрої чи комп'ютері. MFA використовує кілька різних факторів, щоб підтвердити особу та надати доступ до різного програмного забезпечення, систем і даних. Зазвичай системи MFA використовують два або більше з наступних інструментів для автентифікації осіб:

1) Що ви знаєте: пароль, особистий ідентифікаційний номер або запитання щодо відновлення.

2) Що у вас є: смарт-карта, токен FIDO, одноразовий пароль (OTP), пристрій Bluetooth, Apple Watch або інший автентифікатор.

3) Хто ви: біометричний автентифікатор, наприклад, відбиток пальця або розпізнавання обличчя.

Перевага багатофакторної автентифікації полягає в тому, що в більшості випадків вона дуже безпечна. Поєднання пароля, фізичного токена та

біометричних даних може значно знизити ризик злому даних та програмного забезпечення.

Однак, якщо MFA має певні переваги у забезпеченні входу користувачів, він також має репутацію (іноді добре заслужену) як дещо складну в управлінні. Користувачі повинні бути забезпечені другим фактором (перший вони запам’ятовують). Для деяких кінцевих користувачів навіть налаштувати мобільний телефон на отримання одноразового пароля через текстове повідомлення може бути нав’язуванням. Тим не менш, MFA безпечно використовується у багатьох організаціях, для захисту власних ресурсів [3].

1.2.2 Апаратні засоби автентифікації

Цей принцип ідентифікації ґрунтується на визначенні особистості користувача за певним предметом, ключем, що перебуває в його ексклюзивному користуванні. Мова йде про спеціальні електронні ключі. На даний момент найбільше поширення одержали два типи пристроїв. До першого ставляться всілякі карти. Їх досить багато, і працюють вони за різними принципами. Так, наприклад, досить зручні у використанні безконтактні карти (їх ще називають проксіміті-карти), які дозволяють користувачам проходити ідентифікацію як у комп’ютерних системах, так й у системах доступу в приміщення. Найбільш надійними вважаються смарт-карти - аналоги звичних багатьом людям банківських карт. Крім того, є й більш дешеві, але менш стійкі до злому карти: магнітні, зі штрих-кодом і т.д.

Іншим типом ключів, які можуть використатися для апаратної ідентифікації, є так звані токени. Ці пристрої мають власну захищену пам’ять і підключаються безпосередньо до одного з портів комп’ютера (USB, LPT).

Головною перевагою застосування апаратної ідентифікації є досить висока надійність. У пам’яті токенів можуть зберігатися ключі, підібрати які хакерам не вдасться. Крім того, у них реалізовано чимало різних захисних механізмів. А вбудований мікропроцесор дозволяє електронному ключу не

тільки брати участь у процесі ідентифікації користувача, але й виконувати деякі інші корисні функції.

Недоліком апаратної ідентифікації є висока ціна. Взагалі ж останнім часом вартість як самих електронних ключів, так і програмного забезпечення, що може працювати з ними, помітно знизилася. Проте для введення в експлуатацію системи майнової ідентифікації однаково будуть потрібні деякі вкладення. Все-таки кожного зареєстрованого користувача потрібно забезпечити персональними токенами. Крім того, згодом деякі типи ключів можуть зношуватися або можуть бути загублені користувачами [4].

1.3 Методи автентифікації користувачів

1.3.1 Автентифікація по паролю

Цей метод ґрунтується на тому, що користувач повинен надати логіні пароль для успішної ідентифікації та автентифікації в системі. Пара логін/пароль задається користувачем при його реєстрації в системі, при цьому в якості логіна може виступати адреса електронної пошти користувача. Існує кілька стандартних протоколів для автентифікації по паролю, а саме: HTTP, Forms, URL query, Request body, HTTP header.

Описаний в стандартах HTTP 1.0 / 1.1, HTTP автентифікація використовується вже багато років і за цей час стала корпоративним стандартом. Розглянемо схему взаємодії з веб-сайтами:

1) Неавторизований клієнт робить запит до захищеного ресурсу на сервері, той у свою чергу відповідає HTTP статусом "401 Unauthorized", також у заголовок додається "WWW-Authenticate", у якому міститься схема і параметри автентифікації.

2) Отримавши таку відповідь, браузер автоматично виводить на екран користувача діалог введення логіна і пароля, куди користувач вводитьдані свого облікового запису.

3) У подальших викликах на сервер даного захищеного ресурсу, браузер додає у заголовок запиту «Authorization», в якому знаходяться всі дані користувача, які необхідні для автентифікації сервером.

4) Сервер проводить автентифікацію користувачів спираючись надані отримані з цього заголовку та робить рішення допускати до захищеногоресурсу користувача, в залежності від ролі або інших даних.

Весь процес стандартизований і добре підтримується всіма браузерами і веб-серверами. Існує кілька схем автентифікації, що відрізняються за рівнем безпеки:

— Basic - найбільш проста схема, при якій логін і пароль користувача передаються в заголовку Authorization в незашифрованому вигляді або зашифрованому за допомогою base64encoded. Однак при використанні HTTPS протоколу, є відносно безпечною (рисунок 1.1);

— Digest - challenge-response-схема, при якій сервер посилає унікальне значення nonce, а браузер передає MD5 хеш пароля користувача, обчислений з використанням зазначеного nonce. Більш безпечна альтернатива Basic схеми при незахищених з'єднаннях, але схильна до man- in-the-middle attacks (з заміною схеми на basic). Крім того, використання цієїсхеми не дозволяє застосувати сучасні хеш-функції для зберігання паролів користувачів на сервері;

— Forms authentification протокол немає певного стандарту, тому всі його реалізації специфічні для конкретних систем, а точніше, для модулів автентифікації фреймворків розробки. Працює за наступним принципом: в веб-додаток включається HTML-форма, в яку користувач повинен ввести свої облікові дані і відправити їх на сервер через HTTP POSTдля автентифікації. У разі успіху веб-додаток створює ідентифікатор сесії, який зазвичай поміщається в куки браузера. При наступних веб-запитах ідентифікатор сесії автоматично передається на сервер і дозволяє додаткамотримати інформацію про поточного користувача для авторизації запиту (рисунок 1.2).

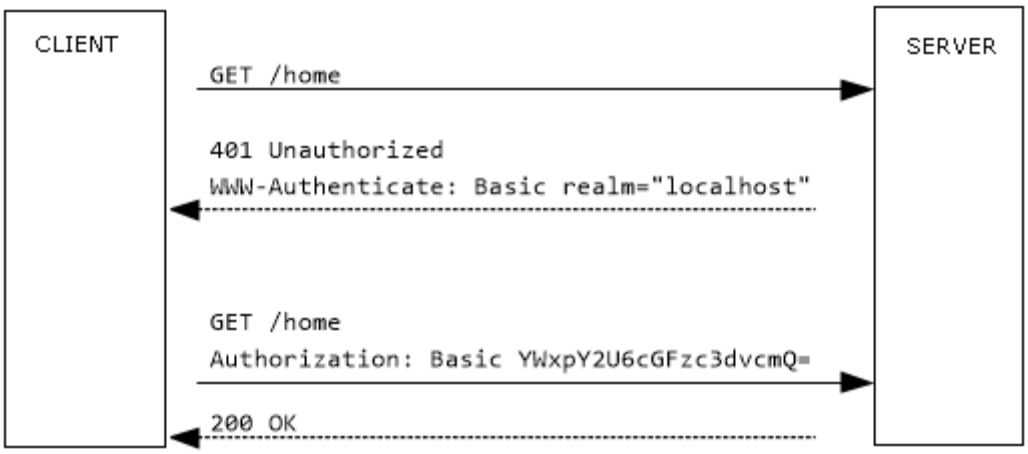

Рисунок 1.1 - Схема Basic автентифікації

Варто відзначити, що при використанні HTTP-автентифікації у користувача немає стандартної можливості вийти з веб-додатка, крім як закрити всі вікна браузера.

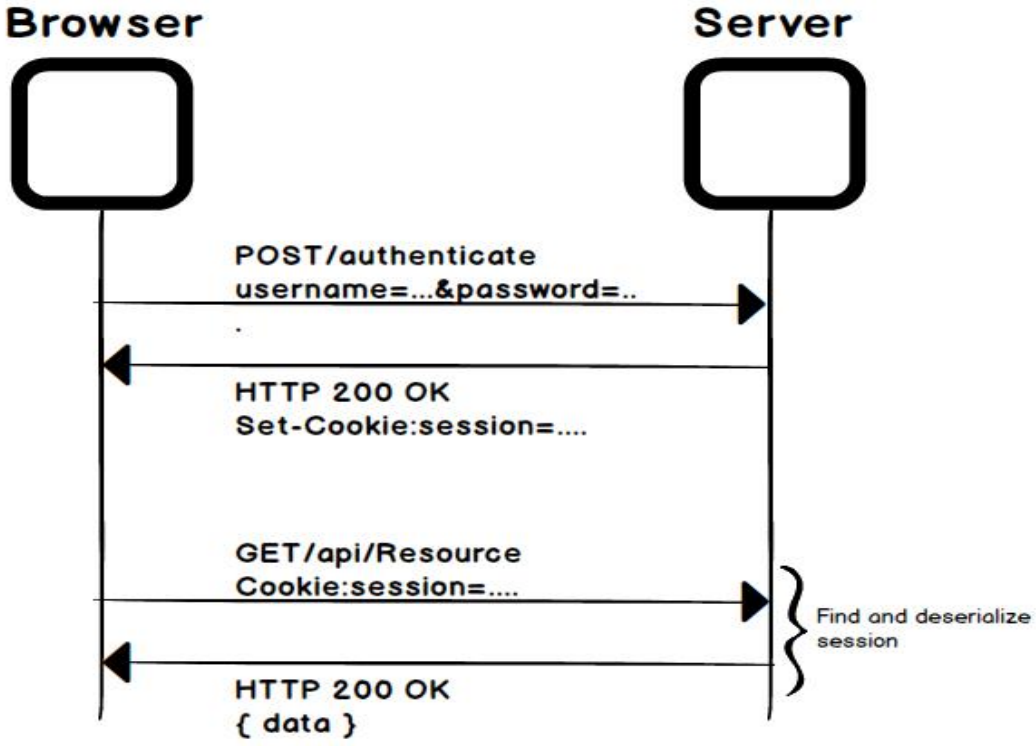

Рисунок 1.2 - Схема Forms автентифікації

Необхідно розуміти, що перехоплення ідентифікатора сесії часто дає аналогічний рівень доступу, що і знання облікових даних користувача. Томувсі

комунікації між клієнтом і сервером у разі автентифікації через форми повинні проводитися тільки по захищеному з'єднанню HTTPS.

Два протоколи, описаних вище, успішно використовуються для автентифікації користувачів на веб-сайтах. Але при розробці клієнт- серверних додатків для конкретних платформ (наприклад, iOS або Android), часто застосовуються нестандартні протоколи, в яких дані для автентифікації передаються в інших частинах запиту. Існує всього декілька місць, де можна передати облікові дані в HTTP запитах:

— адреса запиту - вважається небезпечним варіантом, оскільки рядки URL можуть запам'ятовуватися браузерами, проксі і веб-серверами;

— тіло запиту - безпечний варіант, але він застосовується лише для запитів, що містять тіло повідомлення (такі як POST, PUT, PATCH);

— заголовок запиту - оптимальний варіант, при цьому можуть використовуватися і стандартний заголовок Authorization, та інші довільні заголовки.

За простотою реалізації даного методу ховається багато недоліків і тому даний спосіб вважаться не дуже надійним. Паролі часто можна підібрати, а користувачі схильні використовувати прості і однакові паролі врізних системах, або записувати їх на клаптиках паперу. Якщо зловмисник зміг з'ясувати пароль, то користувач часто про це не дізнається. Крім того, розробники додатків можуть допустити ряд концептуальних помилок, що спрощують злом облікових записів. Нижче представлений список найбільш частих вразливостей в разі використання автентифікації по паролю:

— веб-додаток допускає передачу паролів по незахищеному HTTP-з'єднання або в рядку URL;

— веб-додаток не використовує безпечні хеш-функції для зберігання паролів користувачів;

— веб-додаток не надає користувачам можливість зміни пароля або не

нотифікує користувачів про зміну їх паролів;

— веб-додаток використовує вразливу функцію відновлення пароля, яку можна використовувати для отримання несанкціонованого доступу до інших облікових записів;

— веб-додаток створює ідентифікатори сесії таким чином, що вони можуть бути підібрані або передбачені для інших користувачів;

— веб-додаток допускає передачу ідентифікатора сесії по незахищеному HTTP з'єднанню, або в рядку URL;

— веб-додаток вразливий для атак з фіксацією сесії (не замінює ідентифікатор сесії при переході від анонімної сесії користувача в автентифіковану);

— веб-додаток не встановлює прапори HttpOnly і Secure для файлів cookies, що містять ідентифікатори сесії;

— веб-додаток не знищує сесії користувача після короткого періоду неактивності або не надає функцію виходу з автентифікованої сесії.

Автентифікація за одноразовими паролями зазвичай застосовується додатково до автентифікації по паролю для реалізації двох факторної автентифікації. У цій концепції користувачеві необхідно надати дані двох типів для входу в систему: щось, що він знає (наприклад, пароль), і щось, чим він володіє (наприклад, пристрій для генерації одноразових паролів). Наявність двох факторів дозволяє в значній мірі збільшити рівень безпеки, що може бути необхідно для певних видів веб-додатків.

У веб-додатках такий механізм автентифікації часто реалізується за допомогою розширення автентифікації за допомогою форм: після первинної автентифікації по паролю, створюється сесія користувача, проте в контекстіцієї сесії користувач не має доступу до додатка до тих пір, поки він не виконає додаткову автентифікацію за одноразовим паролем [5].

1.3.2 Автентифікація по сертифікатам

Сертифікат являє собою набір атрибутів, що ідентифікують власника. Центр сертифікації ключів виступає в ролі посередника, який гарантує справжність сертифікатів. Також сертифікат криптографічно пов'язаний з закритим ключем, який зберігається у власника сертифіката і дозволяє однозначно підтвердити факт володіння сертифікатом. На стороні клієнта сертифікат разом з закритим ключем можуть зберігатися в операційній системі, в браузері, в файлі, на окремому фізичному пристрої. Зазвичай закритий ключ додатково захищений паролем або PIN-кодом. Автентифікація за допомогою сертифіката відбувається в момент з'єднанняз сервером. Цей механізм також добре підтримується браузерами, які дозволяють користувачеві вибрати і застосувати сертифікат, якщо веб-сайт допускає такий спосіб автентифікації (рисунок 1.3).

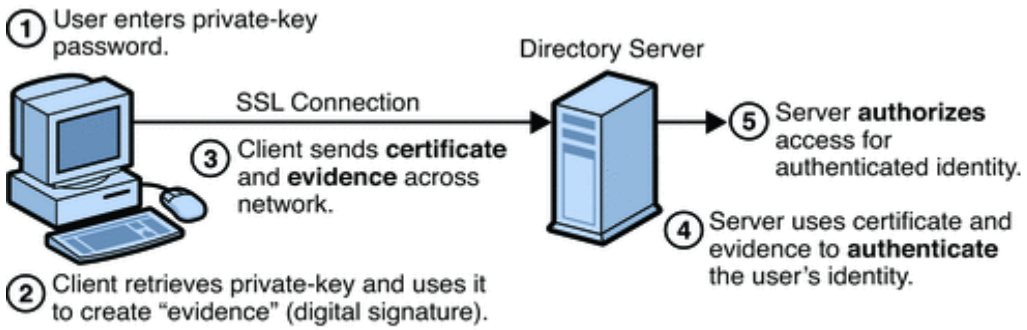

Рисунок 1.3 - Схема автентифікації по сертифікатах

Під час автентифікації сервер виконує перевірку сертифіката напідставі наступних правил:

1) Сертифікат повинен бути підписаний довіреним центром сертифікації ключів (перевірка ланцюжка сертифікатів).

2) Сертифікат повинен бути дійсним на поточну дату (перевірка терміну дії).

3) Сертифікат не повинен бути відкликаний відповідним центром сертифікації (перевірка списків виключення).

Після успішної автентифікації веб-додаток може виконати авторизацію запиту на підставі таких даних сертифіката, як subject (ім'я власника), issuer (емітент), serial number (серійний номер сертифіката) або thumbprint (відбиток відкритого ключа сертифіката).

Використання сертифікатів для автентифікації - більш надійний спосіб, ніж автентифікація за допомогою паролів. Це досягається створенням в процесі автентифікації цифрового підпису, наявність якого доводить факт застосування закритого ключа в конкретній ситуації. Однак труднощі з поширенням і підтримкою сертифікатів робить такий спосіб автентифікації малодоступним в широких колах [6].

1.3.3 Автентифікація по ключам доступу

Цей спосіб найчастіше використовується для автентифікації пристроїв, сервісів або інших додатків при зверненні до веб-сервісів. Тут в якості секрету застосовуються ключі доступу (access key, API key) - довгі унікальні рядки, що містять довільний набір символів, по суті замінюють собою комбінацію логін / пароль.

У більшості випадків, сервер генерує ключі доступу за запитом користувачів, які далі зберігають ці ключі в клієнтських додатках. Пристворенні ключа також можливо обмежити термін дії і рівень доступу, який отримає клієнтська програма при автентифікації за допомогою цього ключа. Гарний приклад застосування автентифікації по ключу – хмара Amazon Web Services. Припустимо, у користувача є веб-додаток, що дозволяє завантажувати і переглядати фотографії, і він хоче використовувати сервіс Amazon S3 для зберігання файлів. В такому випадку, користувач через консоль AWS може створити ключ, що маєобмежений доступ до хмари: тільки читання / запис його файлів в Amazon S3. Цей ключ в результаті можна застосувати для

автентифікації веб- додатка в хмарі AWS (рисунок 1.4).

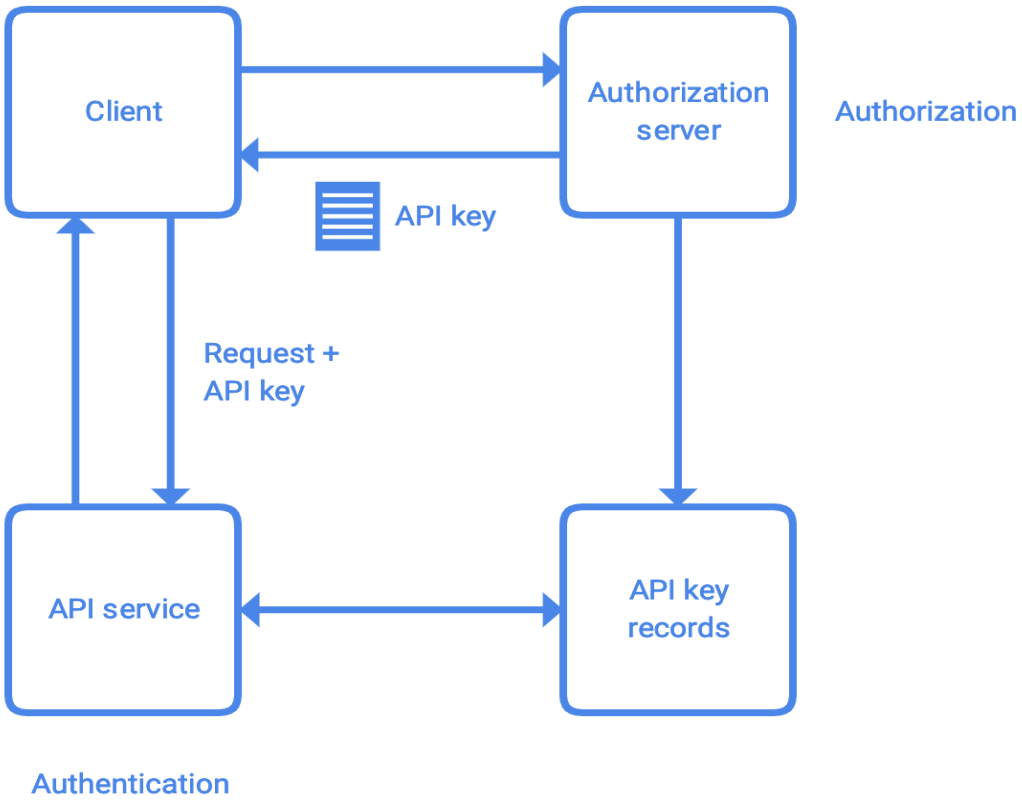

Рисунок 1.4 - Схема автентифікація по ключу

Використання ключів дозволяє уникнути передачі пароля користувача стороннім додаткам (в прикладі вище користувач зберіг в веб-додатку не свій пароль, а ключ доступу). Ключі мають значно більшу ентропією в порівнянні з паролями, тому їх практично неможливо підібрати. Крім того,якщо ключ був розкритий, це не призводить до компрометації основний облікового запису користувача - достатньо лише анулювати цей ключ і створити новий.

З технічної точки зору, тут не існує єдиного протоколу: ключі можуть передаватися в різних частинах HTTP-запиту: адресі, тілі запиту або в заголовках. Найбільш оптимальний варіант використання – заголовок запиту. Щоб уникнути перехоплення ключів, з'єднання з сервером має бути обов'язково захищене протоколом SSL/TLS (рисунок 1.5).

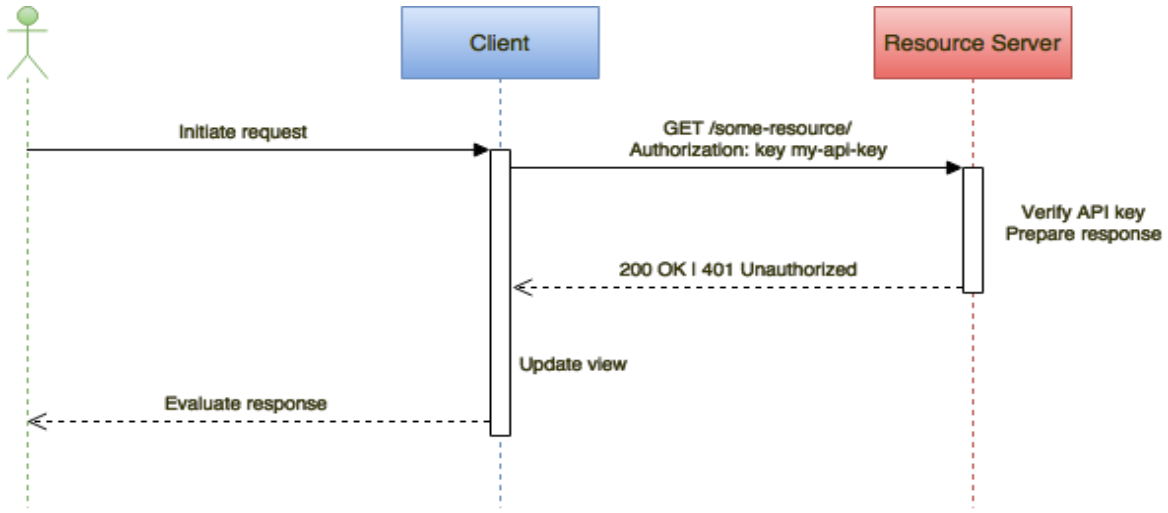

Рисунок 1.5 - Схема автентифікації по ключу доступу, переданого в HTTP заголовку

Крім того, існують більш складні схеми автентифікації по ключам для незахищених з'єднань. В цьому випадку, ключ зазвичай складається з двох частин: публічної і приватної. Публічна частина використовується для ідентифікації клієнта, а приватна частина дозволяє згенерувати, так званий електронний підпис, що використовується для декодування запиту направленого на сервер [7].

1.3.4 Автентифікація по токенах

Найчастіше застосовується при побудові розподілених систем Single Sign-On (SSO), де один додаток (resource server) делегує функцію автентифікації користувачів іншому додатку (authentication service). Типовий приклад цього способу - вхід в додаток через обліковий запис в соціальних мережах. Тут соціальні мережі є сервісами автентифікації, а додаток довіряє функцію автентифікації соціальним мережам.

Реалізація цього способу полягає в тому, що постачальник посвідчень(IP) надає достовірні відомості про користувача в вигляді токена, а постачальник послуг (SP) використовує цей токен для ідентифікації, автентифікації і

авторизації користувача (рисунок 1.6).

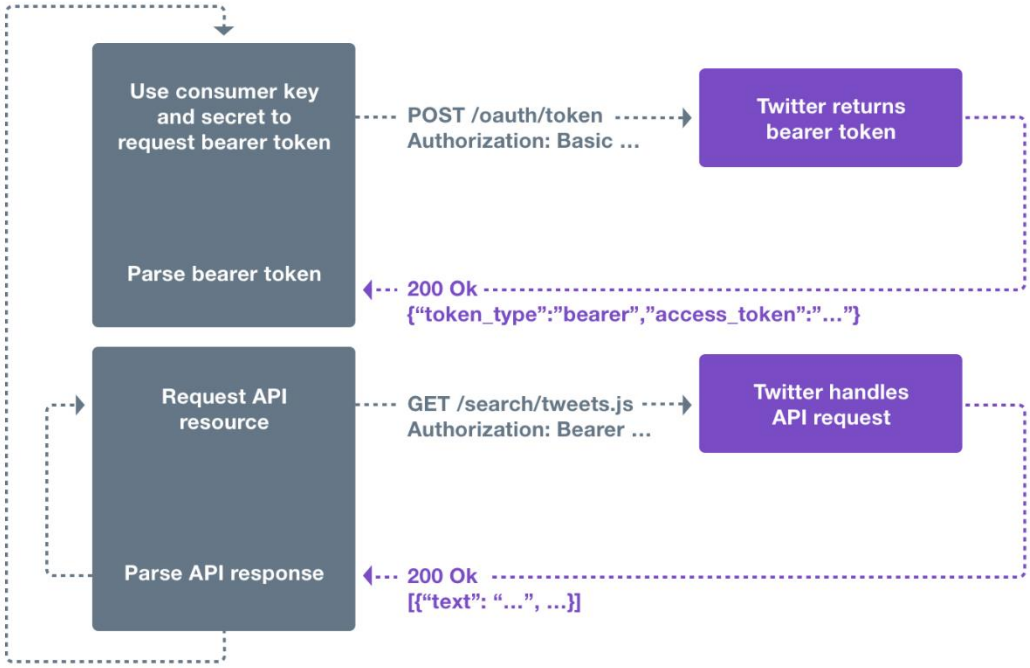

Рисунок 1.6 - Схема автентифікації «активного» клієнта за допомогою токена, переданого за допомогою Bearer схеми.

На загальному рівні, весь процес виглядає наступним чином:

1) клієнт автентифікується в постачальнику посвідчень одним із способів, специфічним для нього (пароль, ключ доступу, сертифікат);

2) клієнт просить постачальника посвідчень надати йому токен для конкретного SP-додатка. Постачальник посвідчень генерує токен і відправляє його клієнту;

3) клієнт автентифікується в постачальнику послуг за допомогою цього токена.

Процес, описаний вище, відображає механізм автентифікації активного клієнта, тобто такого, який може виконувати запрограмовану послідовність дій (наприклад, iOS / Android програми). Браузер — пасивний клієнт в тому сенсі, що він тільки може відображати сторінки, запитані користувачем. В цьому

випадку автентифікація досягається за допомогою автоматичного перенаправлення браузера між постачальником посвідчень та постачальником послуг.

Існує кілька стандартів, в точності що визначають протокол взаємодії між клієнтами (активними і пасивними) і постачальниками посвідчень та послуг і формат підтримуваних токенів. Серед найбільш популярних стандартів - OAuth, OpenID Connect і SAML.

Сам токен зазвичай являє собою структуру даних, яка містить інформацію: хто згенерував токен, хто може бути одержувачем токена, термін дії, набір відомостей про самого користувача. Крім того, токен додатково підписується для запобігання несанкціонованих змін і гарантій автентичності.

При автентифікації за допомогою токена постачальник послуг повинен виконати наступні перевірки:

1) Токен був виданий довіреним постачальником посвідчень.

2) Токен призначається поточному постачальнику послуг.

3) Термін дії токена ще не закінчився.

4) ·Токен справжній і не був змінений (перевірка підпису).

У разі успішної перевірки постачальник послуг виконує авторизацію запиту на підставі даних про користувача, що містяться в токені [8].

1.4 Види форматів токенів

Існує кілька поширених форматів для веб-додатків:

1) Simple Web Token (SWT) — найбільш простий формат, який представляє собою набір довільних пар ключ/значення в форматі кодування HTML form. Стандарт визначає кілька зарезервованих імен: Issuer, Audience, ExpiresOn і HMACSHA256. Токен підписується за допомогою симетричного ключа, таким чином постачальник посвідчень та захищений ресурс повинні

мати цей ключ для можливості створення /перевірки токена.

2) JSON Web Token (JWT) — містить три блоки, між якими ставлять крапку: заголовок, набір полів і підпис. Перші два блоки представлені в JSON форматі і додатково закодовані в формат base64. Набірполів містить довільні пари ключ / значення, до того ж стандарт JWTвизначає кілька зарезервованих імен (iss, aud, exp і інші). Підпис може генеруватися за допомогою симетричних і асиметричних алгоритмів електронного підпису.

3) Security Assertion Markup Language (SAML) - визначає токени (SAML assertions) в XML-форматі, що включає інформацію про емітента, про суб'єкта, необхідні умови для перевірки токена, набір додаткових тверджень про користувача. Підпис SAML-токенів здійснюється за допомогою асиметричної криптографії. Крім того, на відміну від попередніх форматів, SAML-токени містять механізм для підтвердження володіння токеном, що дозволяє запобігти перехоплення токенов через man-in-the- middle-атаки при використанні незахищених з'єднань [9].

2 АНАЛІЗ ТА ПОРІВНЯННЯ ПРОТОКОЛІВ OAUTH2.0 ТА OPENID

2.1 Протокол Oauth 2.0

2.1.1 Специфікація Oauth 2.0

OAuth 2.0 - це протокол делегування, засіб, що дозволяє людині, що контролює ресурс, дозволити програмному додатку доступ до ресурсу від свого імені, не видаючи себе за нього.

Програма запитує авторизацію у власника ресурсу та отримує маркери, які він може використовувати для доступу до ресурсу. Все це відбувається без необхідності додатку видавати себе за людину, яка контролює ресурс, оскільки токен явно представляє делеговане право доступу. Багато в чому токен OAuth можна вважати «valet key» для Інтернету. Не всі автомобілі мають ключі, але для тих, у кого вони є, ключ забезпечує додаткову безпеку ніж передача звичайного ключа. Ключ паркувальника автомобіля дозволяє власнику автомобіля надати обмежений доступ комусь, не передаючи при цьому повний контроль у вигляді ключа власника. Прості ключі камердинера обмежують доступ камердинера до запалювання та дверей, але не до багажника або бардачка. Більш складні ключі можуть обмежити максимальну швидкість автомобіля і навіть заглушити його, якщо він проїде більше ніж на встановлену відстань від початкової точки, посилаючи попередження власнику. Так само токени OAuth можуть обмежити доступ клієнта тільки тими діями, які власник ресурсу делегував.

Наприклад, припустимо, що у вас є сервіс хмарного сховища фотографій і сервіс фотодруку, і ви хочете мати можливість друкувати фотографії, які ви зберегли в сервісі сховища. На щастя, ваш сервіс друку може спілкуватися з сервісом хмарного сховища за допомогою API. Це чудово, за винятком того, що

ці два сервіси запущені різними компаніями, а це означає, що у вашому обліковому записі з сервісом сховища немає підключення до облікового запису з сервісом друку. Ми можемо використовувати OAuth, щоб вирішити цю проблему, дозволивши вам делегувати доступ до своїх фотографій у різних службах не передаючи свій пароль у сервіс другої компанії.

Хоча OAuth багато в чому байдужий до того, який тип ресурсу він захищає, він чудово підходить до сучасних RESTful веб-сервісів і добре працює як для веб-додатків, так і для настільних клієнтських програм. Його можна масштабувати з невеликого однокористувацького додатка до багатомільйонного інтернет-API.

Важливо розуміти, що OAuth не є протоколом автентифікації, хоча його можна використовувати для його створення. Транзакція OAuth сама по собі нічого не говорить вам про те, хто є користувачем і навіть чи він там взагалі є. Згадайте наш приклад фотодруку - фотопринтеру не потрібно знати, хто є користувачем, лише те, що хтось сказав (надав повноваження), що можна завантажити деякі фотографії.

Крім того, OAuth використовує автентифікацію в кількох місцях, зокрема автентифікацію власника ресурсу та клієнтського програмного забезпечення на сервері авторизації. Ця вбудована автентифікація сама по собі не робить OAuth протоколом автентифікації [10].

2.1.2 Ролі в Oauth 2.0

OAuth має на меті отримання право доступу від одного компонента системи до іншого. Зокрема, у світі OAuth клієнтська програма хоче отримати доступ до захищеного ресурсу від імені власника ресурсу (зазвичай кінцевий користувач). Специфікація OAuth визначає чотири ролі:

1) Власник ресурсу має доступ до API і може делегувати доступ до цього API. Власником ресурсу, як правило, є особа, і зазвичай передбачається, що вона має доступ до веб-браузера.

2) Захищений ресурс — це компонент, до якого має доступ власник ресурсу. Він може приймати багато різних форм, але здебільшого це якийсь веб-API. Хоча назва «ресурс» звучить так, ніби це щось для завантаження, API може дозволяти читання, запис та інші операції так само добре.

3) Клієнт — це частина програмного забезпечення, яка отримує доступ до захищеного ресурсу від імені власника ресурсу. Якщо ви веб-розробник, ви можете подумати, що ім'я «клієнт» означає веб-браузер, але цей термін тут використовується не так. Якщо ви розробник бізнес-додатків, ви можете думати про «клієнта» як про людину, яка оплачує ваші послуги, але ми говоримо не про це. У OAuth клієнтом є будь-яке програмне забезпечення, яке використовує API, що надає захищений ресурс.

4) Сервер авторизації — цей сервер надає клієнту маркери доступу після успішної автентифікації власника ресурсу та отримання авторизації [11].

2.1.3 Принцип роботи

OAuth — це протокол, призначений саме для авторизації: в OAuth кінцевий користувач делегує деякі повноваження на доступ до захищеного ресурсу для клієнтської програми для дії

від їхнього імені. Щоб це сталося, OAuth вводить ще один компонент у файл система: сервер авторизації.

Захищений ресурс довіряє серверу авторизації (AS) видавати клієнтам облікові дані безпеки спеціального призначення, які називаються маркерами доступу OAuth. Щоб отримати токен, виконуються наступні дії

1) Клієнт запитує авторизацію у власника ресурсу.

2) Коли власник ресурсу авторизує клієнта, клієнт отримує грант авторизації(спеціальний код).

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

4) Сервер авторизації автентифікує клієнта, перевіряє грант

авторизації та, якщо він дійсний, видає маркер доступу (токен доступу) і маркер оновлення (токен оновлення).

5) Клієнт запитує захищений ресурс у провайдера та автентифікується, представляючи доступ до токена.

6) Провайдер перевіряє доступ до токенів і, якщо він дійсний, обслуговує запит.

У жодному кроці в цьому процесі клієнтові не відкриваються облікові дані власника ресурсу: власник ресурсу автентифікується на сервері авторизації окремо від усього, що використовується для зв'язку з клієнтом. Клієнт також не має потужного ключа розробника: клієнт не може отримати доступ до чого-небудь самостійно, і замість цього має бути авторизований дійсним власником ресурсу, перш ніж отримати доступ до будь-яких захищених ресурсів. Це вірно, навіть якщо більшість клієнтів OAuth мають засоби автентифікації на сервері авторизації.

Зазвичай користувачеві ніколи не доводиться безпосередньо бачити маркер доступу або працювати з ним. Замість того, щоб вимагати від користувача створення токенів і вставлення їх у клієнти, протокол OAuth полегшує цей процес і робить його відносно простим для клієнта, щоб запитати маркер, а користувачу — авторизувати клієнта. Потім клієнти можуть керувати маркерами, а користувачі — клієнтськими додатками [12].

2.2 Протокол OpenID Connect

2.2.1 Специфікація OpenID Connect

OpenID Connect — це простий шар ідентифікації, побудований на основі протоколу OAuth 2.0, який дозволяє клієнтам перевіряти особу кінцевого користувача на основі автентифікації, виконаної сервером авторизації або постачальником ідентифікаційних даних (IdP), а також отримати базовий профіль - інформацію про кінцевого користувача у сумісній та REST-подібній

формі. OpenID Connect визначає RESTful HTTP API, використовуючи JSON як формат даних.

Основна відмінність між OpenID і OAuth полягає в тому, що OpenID є протоколом автентифікації, тоді як OAuth є протоколом авторизації. OpenID і OAuth є відкритими стандартами, які доповнюють один одного, але OpenID дозволяє автентифікувати користувачів довіреними сторонами. Стороною, що довіряє OIDC, є клієнтська програма OAuth 2.0, яка вимагає автентифікації користувача та заявки від постачальника OIDC. OAuth дозволяє сервером авторизації видавати маркери доступу стороннім клієнтам. OpenID Connect побудовано на профілі OAuth і надає додаткові можливості для передачі особи користувача за допомогою програми. Клієнти використовують OAuth для запиту доступу до API від імені користувача, але ніщо в протоколі OAuth не повідомляє клієнту інформацію про користувача. OpenID Connect надає клієнту доступ до додаткової інформації про користувача, наприклад, справжнього імені користувача, адреси електронної пошти, дати народження або іншої інформації профілю. Він нічого не знає про подію автентифікації, яка відбулася, а також про чи був автентифікований сам власник ресурсу. Він не може:

— сказати, що ви точно підтвердили автентифікацію;

— дізнайтися, як ви пройшли автентифікацію;

— збирати будь-яку корисну інформацію, яка може допомогти вам як клієнту покращити роботу.

Надання такої інформації, включаючи саму подію автентифікації та ким був кінцевий користувач, — це саме те, що робить OpenID Connect. Це дозволяє програмі — довіреній стороні — отримати доступ до криптографічно підписаних підтверджень особи. Переваги включають:

1) Це запевняє клієнтську програму, що автентифікація насправді відбулася за допомогою маркера ідентифікатора, який є засобом передачі атрибутів ідентифікації за допомогою підписаного JSON Web Token (JWS), який можна перевірити за допомогою криптографії з відкритим ключем. Це гарантує,

що делегування доступу Кінцевим користувачем підтримується криптографічно перевіреним твердженням.

2) Це також дозволяє клієнтській програмі знати «щось» про посвідчення кінцевого користувача, що важливо для об’єднання ідентифікаційних даних користувача. Федерація посвідчень зручна для клієнтських програм, оскільки вона ефективно дозволяє їм передавати багато аспектів ідентифікації користувача постачальнику, якому він довіряє, без шкоди для безпеки та доступу.

3) Наявність певного підтвердження автентифікації може допомогти розробникам додатків створити набагато більш безперебійну автентифікацію. Вони можуть (якщо це підтримується постачальником ідентифікаційних даних) використовувати існуючий дійсний маркер ідентифікатора, щоб підтвердити особу кінцевого користувача серверу авторизації. У таких сценаріях сервер авторизації може не запропонувати користувачеві пройти автентифікацію, що призведе до набагато більш гладкої роботи під час використання програми, яка використовує переваги цієї функції.

Є три учасники, які беруть участь у передачі повідомлень OIDC.

1) Кінцевий користувач — кінцевий користувач такий самий, як і власник ресурсу, у якого запитують ідентифікаційну інформацію.

2) Довірена сторона(RP) — довірена сторона така ж, як клієнт OAuth2.0, який вимагає автентифікації кінцевого користувача та заяв від постачальника OpenID.

3) OpenID Provider(OP) — постачальник OpenID такий самий, як сервер авторизації OAuth2.0, який здатний автентифікувати кінцевого користувача та надавати претензії довіреній стороні щодо події автентифікації та кінцевого користувача [13].

2.2.2 Принцип роботи OpenID Connect

Процес автентифікації в OpenId Connect відбувається у наступному

порядку:

1) Користувач намагається розпочати сеанс з вашим клієнтським додатком і перенаправляється на постачальника OpenID, передаючи ідентифікатор клієнта, унікальний для цієї програми.

2) Постачальник OpenID автентифікує та авторизує користувача.

3) Одноразовий код передається назад веб-серверу за допомогою заздалегідь визначеного URL перенаправлення.

4) Веб-сервер передає код, ідентифікатор клієнта та секрет клієнта кінцевій точці маркера OpenID Provider, а постачальник OpenID перевіряє код і повертає маркер доступу, разом з ID токеном.

5) Веб-сервер використовує маркер доступу, щоб отримати додаткові відомості про користувача та встановлює сеанс для користувача.

2.2.3 ID токен

Маркер OpenID Connect ID — це підписаний веб-токен JSON (JWT), який надається клієнтській програмі разом із звичайним маркером доступу OAuth. На відміну від маркера доступу, маркер ідентифікатора спрямовується на RP і призначений для аналізу ним.

Як і у випадку з підписаними маркерами доступу, маркер ідентифікатора містить набір заяв щодо сеансу автентифікації, включаючи ідентифікатор користувача (sub), ідентифікатор постачальника ідентифікаційних даних, який видав маркер (iss), та ідентифікатор клієнта, для якого створено цей токен (aud). Крім того, маркер ідентифікатора містить інформацію про власне часове вікно дії маркера (з вимогами exp та iat), а також будь-яку додаткову інформацію про контекст автентифікації, яку потрібно передати клієнту. Наприклад, токен може вказувати, як давно користувачеві був представлений механізм первинної автентифікації (auth_time) або який тип первинної автентифікації він використовував у IdP (acr).

Маркер ID видається на додаток до маркера доступу як поле id_token

відповіді кінцевої точки маркера, а не замість нього. Це є визнанням того факту, що ці два токени мають різну цільову аудиторію та використання. Підхід з двома маркерами дозволяє маркеру доступу залишатися непрозорим для клієнта, як у звичайному OAuth, одночасно дозволяючи аналізувати маркер ідентифікатора. Крім того, два токени також можуть мати різні життєві цикли, причому термін дії токена ідентифікатора часто закінчується швидше. Хоча маркер ідентифікатора представляє одну подію автентифікації і ніколи не передається зовнішній службі, маркер доступу можна використовувати для отримання захищених ресурсів ще довго після того, як користувач залишив сторінку додатка. Хоча це правда, що ви все ще можете використовувати маркер доступу, щоб запитати, хто авторизував клієнта в першу чергу, це не розповість вам нічого про присутність користувача.

Нарешті, сам маркер ідентифікатора підписується ключем постачальника ідентифікаційної інформації, додаючи ще один рівень захисту до заяв всередині нього на додаток до транспортного захисту TLS, який використовувався для отримання токена в першу чергу. Оскільки маркер ідентифікатора підписаний сервером авторизації, він також надає місце для додавання відокремлених підписів через код авторизації (c_hash) і маркер доступу (at_hash). Клієнт може перевіряти ці хеші, зберігаючи при цьому код авторизації та вміст маркера доступу непрозорим для клієнта, запобігаючи цілий клас ін’єкційних атак [14].

2.3 Порівняння OAuth і OpenID Connect

OpenID Connect і Oauth 2.0 передбачають:

1) Легкі для використання токени ідентичності. Клієнтські програми отримують ідентифікатор користувача, закодований у захищеному веб-маркері JSON (JWT), який називається маркером ідентифікатора.

2) Широке коло використання. Токени можуть використовуватися як для роботи з веб-програмами, так і з мобільними додатками.

3) Простота та можливості. OpenID Connect і Oauth 2.0 досить прості для інтеграції з основними програмами, а також пропонують функції та параметри безпеки, які можуть відповідати корпоративним вимогам.

Основна відмінність Oauth та OpenID полягає в тому, що Oauth - це протокол, який контролює авторизацію захищених ресурсів, таких як програми чи групи файлів. Натомість OpenID Connect є галузевим стандартом федеративної автентифікації. Хоча наразі є багато налаштувань для OAuth 2.0, які дозволяють використовувати цей протокол і для автентифікації також.

Таблиця 2.1 - Порівняння стандартів авторизації та автентифікації

	Oauth 2.0	OpenID Connect
Що це	Відкритий стандарт для авторизації з можливістю автентифікації	Відкритий стандарт для автентифікації
Рік заснування	2006	2014
Ким заснована	Twitter and Google	OpenID Foundation
Формат повідомлень	JSON	JSON

Ці протоколи визначають спеціальні ролі, які по значенню дуже схожі між собою, але мають різну назву (табл. 2.2).

Таблиця 2.2 - Назви ролей в різних протоколах

Роль	OAuth	OIDC
Кінцевий користувач	Resource Owner	End User
Додаток, що потребує доступ	Client	Relying Party (RP)

Постачальник послуг авторизації/автентифікації	Authorization Server (AS)	OpenID Provider (OP)
Браузер	User-Agent	User-Agent

Щодо токенів, то OpenID Connect використовує JWT токени, Oauth 2.0 може використовувати як JWT, так і Bearer токени, а SAML використовує токени в спеціальному форматі.

Таблиця 2.3 - Порівняння токенів

Токен	Формат	Коли використовується	Валідація
JWT	JSON	REST і Bearer токен	JSON Web Signature
Access Token	Випадкові строка або JWT	REST і Bearer токен	OAuth Introspection
ID Token	JWT	REST і Bearer токен	Json Web Signature

Oauth та OpenID дуже прості в конфігурації. Щоб використовувати Oauth достатньо лише зареєструвати клієнта на сервері авторизації, а OpenID взагалі дозволяє динамічну реєстрацію клієнтів, що значно спрощує розгортання додатків на серверах. До того ж, вони підтримують JSON що значно спрощує їх використання.

3 РЕАЛІЗАЦІЯ АВТЕНТИФІКАЦІЇ У ВЕБ-ДОДАТКУ НА ОСНОВІ ПРОТОКОЛУ OPENID

3.1 Технології, які використовуються в проекті

3.1.1 Мова програмування Java

Java – це широко використовувана об’єктно-орієнтована мова програмування та програмна платформа, яка працює на мільярдах пристроїв, включаючи ноутбуки, мобільні пристрої, ігрові консолі, медичні пристрої та багато інших. Правила та синтаксис Java засновані на мовах C і C++.

Однією з основних переваг розробки програмного забезпечення на Java є його портативність . Після того, як ви написали код для програми Java на комп’ютері, дуже легко перенести код на мобільний пристрій. Коли мова була винайдена в 1991 році Джеймсом Гослінгом з Sun Microsystems (пізніше придбана Oracle), основною метою була можливість «писати один раз і запускати де завгодно».

Java — це технологія, що складається як з мови програмування, так і з програмної платформи. Щоб створити програму за допомогою Java, вам потрібно завантажити Java Development Kit (JDK), доступний для Windows, macOS та Linux. Ви пишете програму мовою програмування Java, 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 працює без змін у будь-якій системі, яка підтримує JVM, що дозволяє запускати ваш Java-код будь-де.

Програмна платформа Java складається з JVM, Java API та повного середовища розробки. JVM аналізує та запускає (інтерпретує) байт-код Java. Java API складається з великого набору бібліотек, включаючи основні об'єкти, мережеві функції та функції безпеки; Генерація розширюваної мови розмітки

(XML); і веб-сервіси. У сукупності мова Java і програмна платформа Java створюють потужну, перевірену технологію для розробки корпоративного програмного забезпечення.

Технологія Java є ідеальною платформою для розробки веб-додатків, основою для цифрового бізнесу в будь-якій галузі. Сервери додатків Java — це веб-контейнери для компонентів Java, XML та веб-сервісів, які взаємодіють з базами даних і надають динамічний веб-контент. Сервери додатків Java утворюють стабільне середовище розгортання для корпоративних програм з такими можливостями, як керування транзакціями, безпека, кластеризація, продуктивність, доступність, підключення та масштабованість.

3.1.2 Фреймворк Spring Boot

Spring є найпопулярнішим фреймворком для розробки додатків для корпоративної мови Java. Мільйони розробників у всьому світі використовують Spring Framework для створення високопродуктивного, легко тестованого та багаторазового використання коду.

Spring Framework — це платформа Java з відкритим кодом. Спочатку він був написаний Родом Джонсоном і вперше був випущений під ліцензією Apache 2.0 у червні 2003 року. З самого початку він був лише фреймворком для інверсії залежностей, але дуже швидко у його склад увійшло багато різних модулів, які розширили його функціонал.

Основні функції Spring Framework можна використовувати при розробці будь-якого додатка Java, але є розширення для створення веб-додатків поверх платформи Java EE. Фреймворк Spring спрямований на те, щоб зробити розробку J2EE простішою у використанні, і просуває передові практики програмування, увімкнувши модель програмування на основі POJO.

Нижче наведено список кількох переваг використання Spring Framework:

1) Spring дозволяє розробникам розробляти програми корпоративного класу за допомогою POJO. Досягається це за рахунок інверсії залежностей,

тобто вам не потрібно піклуватися про те, як поставити залежності в компоненти, про це піклується Spring.

2) Spring організований модульно. Незважаючи на те, що кількість пакетів і класів значна, вам доведеться турбуватися лише про ті, які вам потрібні, а решту ігнорувати.

3) Spring не винаходить колесо, натомість він дійсно використовує деякі з існуючих технологій, як-от кілька фреймворків ORM, фреймворки журналів, таймери JEE, Quartz і JDK та інші технології перегляду.

4) Тестування програми, написаної за допомогою Spring, просте, оскільки в цю структуру переміщено код, що залежить від середовища. Крім того, використовуючи POJO в стилі Java Beanstyle, стає легше використовувати ін'єкцію залежностей для введення тестових даних.

5) Веб-фреймворк Spring — це добре розроблений веб-фреймворк MVC, який надає чудову альтернативу веб-фреймворкам, таким як Struts або іншим перероблениим або менш популярним веб-фреймворкам.

6) Контейнери IoC, як правило, легкі, особливо якщо порівнювати, наприклад, з контейнерами EJB. Це вигідно для розробки та розгортання програм на комп'ютерах з обмеженими ресурсами пам'яті та ЦП.

7) Spring забезпечує узгоджений інтерфейс керування транзакціями, який може зменшуватися до локальної транзакції (наприклад, за допомогою однієї бази даних) і до глобальних транзакцій (наприклад, за допомогою JTA).

Spring Framework складається з функцій, організованих приблизно в 20 модулів. Ці модулі згруповані в основний контейнер, доступ до даних/інтеграція, веб, AOP (аспектно-орієнтоване програмування), інструментарій та тест, як показано на наступній діаграмі (рисунок 3.1).

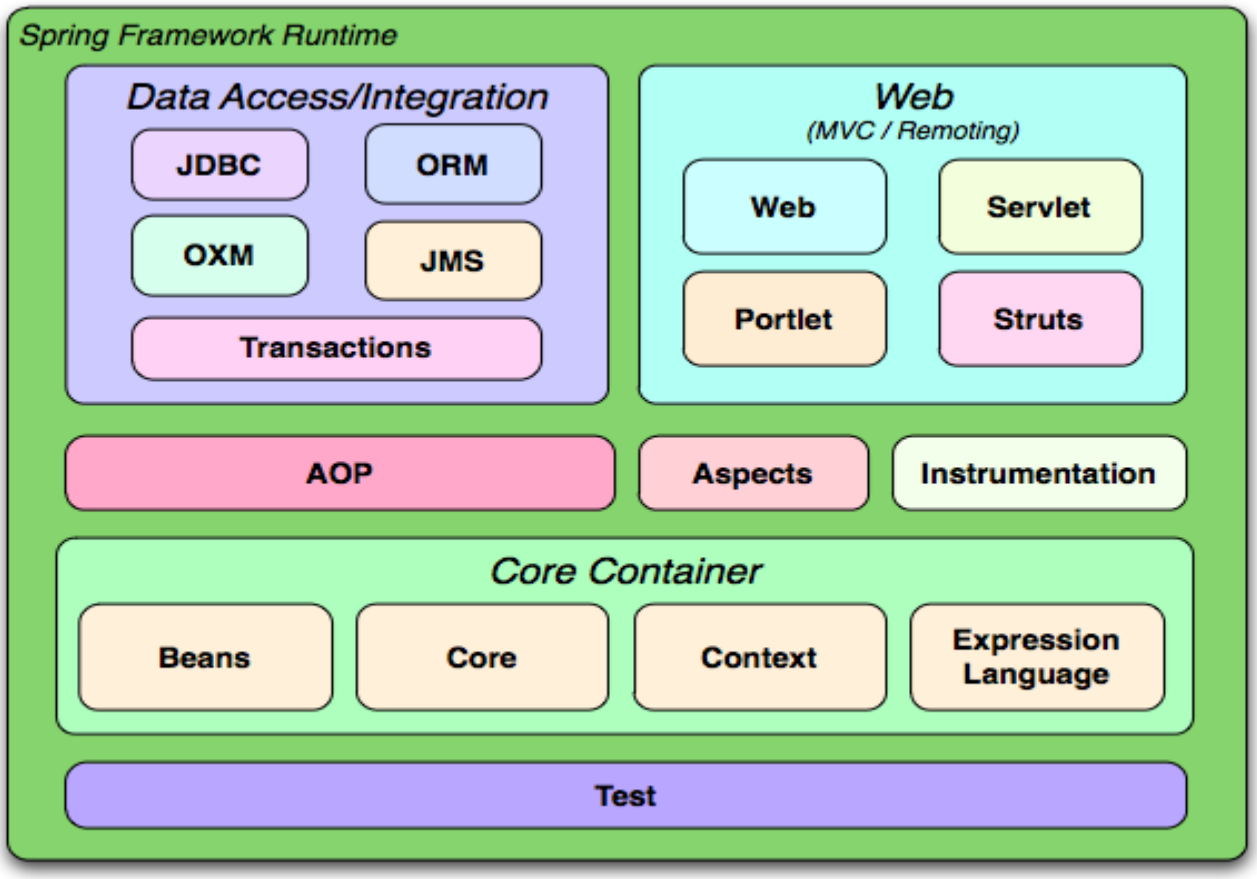

Рисунок 3.1 - Модулі в Spring Framework

Spring Boot побудований на основі фреймворку Spring, і він поставляється з багатьма залежностями, які можна підключити до програми Spring. Деякі приклади: Spring Kafka, Spring LDAP, Spring Web Services і Spring Security. Однак розробникам доводиться самостійно налаштовувати кожен залежний модуль, використовуючи велику кількість конфігураційних файлів XML або анотацій.

Фреймворк Spring зосереджується на забезпеченні гнучкості за допомогою функції ін’єкції залежностей. Це допомагає швидко впроваджувати необхідні залежності, а також розробляти вашу програму слабко пов’язаним способом.

Spring Boot, з іншого боку, зосереджений на скороченні довжини коду і надає вам простий спосіб запуску програми Spring. Ви можете розпочати роботу

з мінімальними конфігураціями без необхідності повного налаштування конфігурації Spring.

Деякі переваги використання цього фреймворку:

— він забезпечує гнучкий спосіб налаштування Java Beans, конфігурацій XML і транзакцій бази даних;

— він забезпечує потужну пакетну обробку та керує кінцевими точками REST;

— у Spring Boot все налаштовується автоматично, не потрібні ручні налаштування;

— він пропонує розробити додаток на основі анотацій;

— полегшує управління залежностями за допомогою так званих стартерів;

— він включає в себе вбудований контейнер сервлетів.

Spring Boot автоматично налаштовує вашу програму на основі залежностей, які ви додали до проекту за допомогою анотації @EnableAutoConfiguration. Наприклад, якщо база даних MySQL знаходиться на шляху до вашого класу, але ви не налаштували жодного підключення до бази даних, Spring Boot автоматично налаштовує базу даних у пам’яті. Точкою входу програми для весняного завантаження є клас, що містить анотацію @SpringBootApplication і основний метод. Spring Boot автоматично сканує всі компоненти, включені в проект, використовуючи анотацію @ComponentScan.

Управління залежностями є критичним аспектом будь-якого складного проекту. І робити це вручну не ідеально; чим більше часу ви витратили на це, тим менше часу у вас залишилося на інші важливі аспекти проекту.

До появи Spring Boot розробники Spring витрачали багато часу на управління залежностями. Стартери Spring Boot були створені для вирішення саме цієї проблеми. Стартери – це набір зручних дескрипторів залежностей, які ви можете включити у свою програму. Ви отримуєте залежності для всіх

необхідних вам Spring та пов’язаних з ними технологій, без необхідності шукати зразки коду і копіювати та вставляти множини дескрипторів залежностей.

3.1.3 Інструмент автоматизації збірки проекту Gradle

Gradle — це інструмент автоматизації збірки на основі Groovy і Kotlin. Це відкритий вихідний код і досить гнучкий, щоб створити практично будь-який тип програмного забезпечення. Він також підтримує як автоматичне завантаження залежностей, так і багато сховищ, включаючи Maven і Ivy.

Дві основні сутності в Gradle – це проекти та завдання. Кожна збірка Gradle складається щонайменше з одного проекту (наприклад, бібліотека JAR, програма Java, веб-додаток). Кожен проект складається з одного або кількох завдань. Завдання являє собою деяку атомарну роботу, яку виконує збірка. Кожне завдання може залежати від одного або кількох додаткових завдань, і це працює шляхом накладання кількох операцій одна за одною. Таким чином, легко досягти тривалої послідовності операцій, зберігаючи складність під контролем.

Gradle — це сучасний інструмент конструювання, який вирішує проблеми, які стоять перед нами в інших інструментах, таких як ANT і Maven. Будівельний інструмент дозволить нам досягти мети автоматизації проекту. Отже, безпека, доступність, універсальність, розширюваність та ефективність не повинні бути скомпрометовані. Він розроблений для подолання недоліків Maven і Ant і підтримує різні IDE.

Gradle реалізує широкий спектр стратегій, щоб зробити ваші збірки швидше:

— інкрементальна компіляція аналізує залежності між вашими джерелами та класами та перекомпілює лише ті, на які вплинули зміни;

— розумний аналізатор шляхів до класів Gradle уникає непотрібної компіляції, якщо двійковий інтерфейс бібліотеки не змінився;

— краще моделювання залежностей за допомогою плагіна Java Library,

зменшує розмір шляху до класу компіляції, що має великий позитивний вплив на продуктивність;

— Поєднання декларативної та програмної моделі для збірки. Можна декларативно об’явити залежності, але програмно написати завдання, що збільшує гнучкість.

3.1.4 База даних PostgreSQL

PostgreSQL — це потужна система об'єктно-реляційних баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження даних. Витоки PostgreSQL сягають 1986 року в рамках проекту POSTGRES в Каліфорнійському університеті в Берклі і має понад 30 років активної розробки на базовій платформі.

PostgreSQL заслужив міцну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти відкритого коду, що стоїть за програмним забезпеченням, щоб постійно надавати продуктивні та інноваційні рішення. PostgreSQL працює на всіх основних операційних системах, має ACID-сумісність з 2001 року.

PostgreSQL постачається з багатьма функціями, які допомагають розробникам створювати програми, адміністраторам — захищати цілісність даних і створювати відмовостійкі середовища, а також допомагають вам керувати даними незалежно від того, наскільки великий чи маленький набір даних. Окрім того, що PostgreSQL є безкоштовним і відкритим вихідним кодом, він дуже розширюваний. Наприклад, ви можете визначати власні типи даних, створювати власні функції, навіть писати код з різних мов програмування без перекомпіляції бази даних!

PostgreSQL намагається відповідати стандарту SQL, якщо така відповідність не суперечить традиційним функціям або може призвести до

неправильних архітектурних рішень. Багато функцій, необхідних стандартом SQL, підтримуються, хоча іноді вони мають дещо відмінний синтаксис або функції. З часом можна очікувати подальших кроків до відповідності. Починаючи з версії 14 у вересні 2021 року, PostgreSQL відповідає щонайменше 170 із 179 обов'язкових функцій для відповідності SQL:2016 Core.

3.1.5 Docker

Docker — це програмна платформа, яка дозволяє швидко створювати, тестувати та розгортати програми. Docker упаковує програмне забезпечення в стандартизовані блоки, які називаються контейнерами, які містять все необхідне для роботи програмного забезпечення, включаючи бібліотеки, системні інструменти, код і середовищі виконання. Використовуючи Docker, ви можете швидко розгортати програми в будь-якому середовищі та знати, що ваш код працюватиме.

Контейнери — невеликі та легкі середовища виконання, які спільно використовують ядро операційної системи, працюють ізольовано один від одного. Хоча контейнери використовувалися в системах Linux і Unix протягом певного часу, Docker, проект із відкритим кодом, запущений у 2013 році, допоміг популяризувати цю технологію, полегшивши розробникам пакування свого програмного забезпечення, щоб «будувати один раз і запускати де завгодно».

3.1.6 Утиліта для створення та виконання запитів Postman

API, будучи основою програмного забезпечення, вимагало відповідних методів тестування. Кількість методів тестування задовольняє ранню потребу в розпізнаванні та усуненні помилок, які можуть бути серйозною перешкодою в подальшому розвитку.

Простими словами, Postman – це комп'ютерний додаток, який використовується для тестування API. Postman надсилає запит API на веб-сервер і отримує відповідь, яка б вона не була. Під час надсилання та отримання запитів

у Postman не потрібно додаткової роботи чи налаштування фреймворку. Широко використовується тестувальниками та розробниками для кращого тестування програми.

Postman із багатьма функціями та простим у роботі інтерфейсом використовувався мільйонами тестувальників. Використовуючи його простий і зручний інтерфейс, ви можете легко відправити запит, просто заповніть необхідні дані, виберіть метод HTTP і натисніть кнопку «Відправити». Іншою найбільш широко використовуваною функцією є автоматизація, яка дозволяє налаштовувати тести та писати набори тестів.

3.2 Розробка програмного забезпечення для сервісу автентифікації

Спочатку необхідно створити проект в Intellij Idea. Для цього необхідно натиснути кнопку «Файл» і з випадаючого списку обрати «Створити новий проект». У новому вікні вибираємо вкладку «Spring Initializr» та натискаємо кнопку «Далі». Після цього заповняємо всю мета інформацію про проект як показано на рисунку та натискаємо кнопку «Далі» (рисунок 3.2).

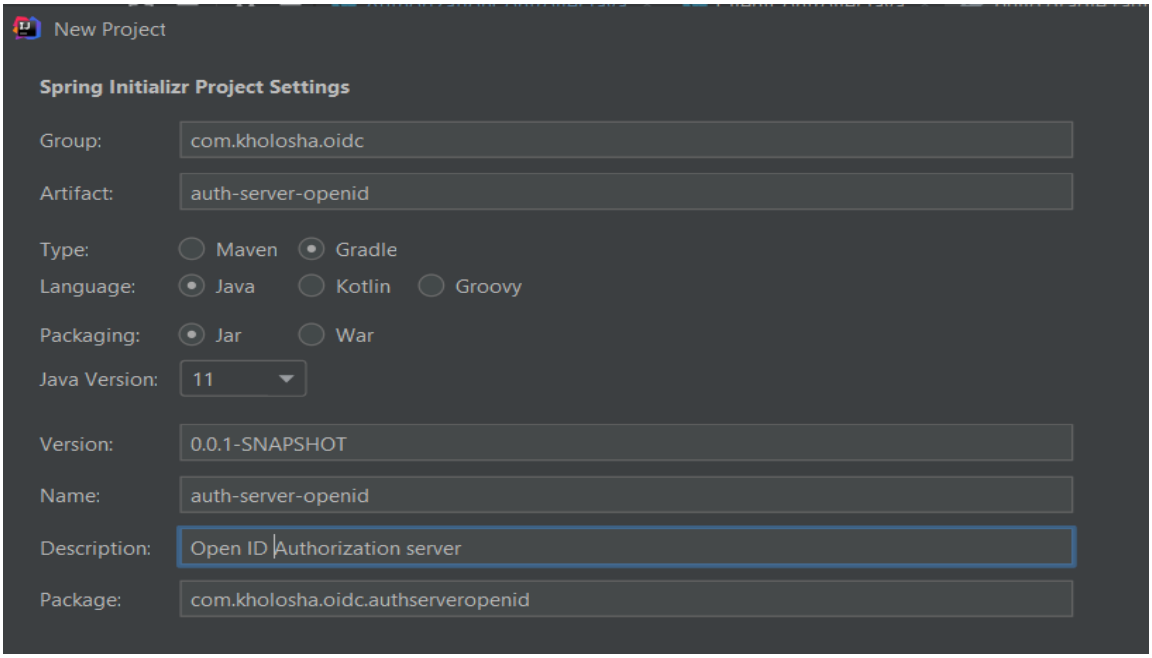

Рисунок 3.2 - Задання метаінформації про проект

Після цього вибираємо залежності, які нам необхідні у проекті (рисунок 3.3):

Рисунок 3.3 - Обрані залежності

Після того, як обрали залежності – натискаємо кнопку «Фініш» та створюємо проект. Необхідно додати сторонні залежності до файлу build.gradle (рисунок 3.4):

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springdoc:springdoc-openapi-ui:1.5.10'
    implementation 'com.nimbusds:nimbus-jose-jwt:9.13'
    implementation 'org.apache.commons:commons-lang3'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.h2database:h2'
    runtimeOnly 'mysql:mysql-connector-java'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'
    testImplementation('org.springframework.boot:spring-boot-starter-test')
    testImplementation 'org.springframework.security:spring-security-test'
    testImplementation 'io.rest-assured:spring-mock-mvc'

    //lombok
    compileOnly 'org.projectlombok:lombok:1.18.4'
    annotationProcessor 'org.projectlombok:lombok:1.18.4'
}
```

Рисунок 3.4 - Залежності проекту в файлі build.gradle

Також необхідно додати конфігурацію сервера до файлу application.yml. Тут конфігурується доступ зчитувач JSON повідомлень, порт сервера, на якому він буде працювати та доступ до бази даних, в нашому випадку це PostgreSQL. Також тут винесені властивості щодо конфігурації токенів (формат, час життя і т.д.) (рисунок 3.5 та рисунок 3.6).

```yaml
spring:
  jpa:
    open-in-view: false
  jackson:
    date-format: com.fasterxml.jackson.databind.util.StdDateFormat
    default-property-inclusion: non_null
  datasource:
    url: jdbc:postgresql://localhost:5432/oidc
    username: ${POSTGRES_USERNAME:postgres}
    password: ${POSTGRES_PASSWORD:postgres}
server:
  port: 9090
  servlet:
    context-path: /auth
  error:
    include-stacktrace: never
```

Рисунок 3.5 - Конфігурація сервера автентифікації

```yaml
auth-server:
  issuer: http://localhost:${server.port}${server.servlet.context-path}
  access-token:
    default-format: jwt
    lifetime: 10m
  id-token:
    lifetime: 5m
  refresh-token:
    lifetime: 8h
    max-lifetime: 8h
```

Рисунок 3.6 - Винесення властивостей токенів

Згідно до завдання створимо структуру проекту, як показано на рисунку 3.7:

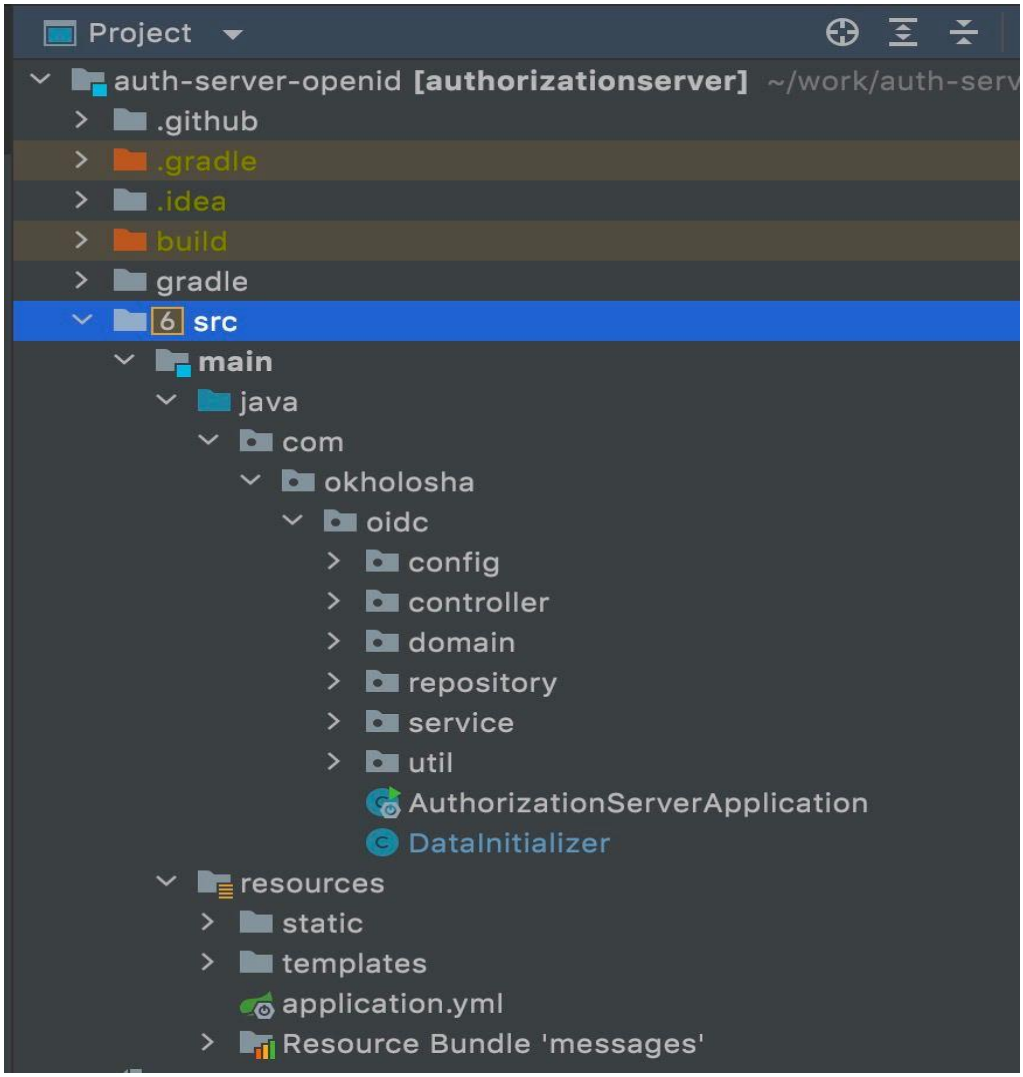

Рисунок 3.7 - Структура проекту згідно завдання

Зібрати проект в виконуючий файл можна за допомогою команди «gradlew build». Ця команда скомпілює код у виконуючий код та збудує jar-архів, який можна буде запустити на будь якому комп’ютері (рисунок 3.8).

Рисунок 3.8 - Скомпільований jar-архів

3.2.1 Конфігурація додатка

У проекті знаходиться декілька класів, які конфігурують наш додаток. Перший з них відповідає за генерування унікального ідентифікатора для сутностей та токенів. Другий відповідає за створення біна для шифрування паролю, щоб в базі зберігалось тільки його хеш значення, з якого неможливо відтворити пароль (рисунки 3.9 та 3.10).

```
@Configuration
public class IdGeneratorConfiguration {

  @Bean
  public IdGenerator idGenerator() { return new JdkIdGenerator(); }
}
```

Рисунок 3.9 - Створення екземпляра для генерації унікальних ідентифікаторів

```
@Configuration
public class PasswordEncoderConfiguration {

  @Bean
  public PasswordEncoder passwordEncoder() { return PasswordEncoderFactories.createDelegatingPasswordEncoder(); }
}
```

Рисунок 3.10 - Створення екземпляра для гешування паролів

Також тут ще два класи, які конфігурують захист щодо доступу API сервера. Конфігурується API отримання токена і API управління користувачами та клієнтами (рисунки 3.11 та 3.12). Клас AuthorizationServerApplication є стартовою точкою додатка (рисунок 3.13).

```
@Configuration
@Order(1)
public static class PublicEndpoints extends WebSecurityConfigurerAdapter {

    private final RegisteredClientDetailsService registeredClientDetailsService;

    public PublicEndpoints(@Autowired @Qualifier("registeredClientDetailsService")
                                RegisteredClientDetailsService registeredClientDetailsService) {
        this.registeredClientDetailsService = registeredClientDetailsService;
    }

    @Override
    public void configure(WebSecurity web) {
        web.ignoring().mvcMatchers( ...mvcPatterns: "/token", "/introspect", "/revoke", "/userinfo",
                "/.well-known/openid-configuration", "/jwks");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.requestMatchers(r -> r.mvcMatchers( ...patterns: "/token", "/introspect", "/revoke", "/userinfo",
                        "/.well-known/openid-configuration", "/jwks"))
                .authorizeRequests(a -> {
                            a.mvcMatchers(POST,  ...mvcPatterns: "/introspect").hasRole("CLIENT");
                            a.mvcMatchers(POST,  ...mvcPatterns: "/revoke").hasRole("CLIENT");
                            a.mvcMatchers(POST,  ...mvcPatterns: "/token").hasRole("CLIENT");
                            a.anyRequest().denyAll();
                        })
                .httpBasic(withDefaults())
                .userDetailsService(this.registeredClientDetailsService)
                .cors(withDefaults())
                .csrf()
                .disable();
    }
}
```

Рисунок 3.11 - Конфігурація захисту API для управління токенами

```
@Configuration
@Order(2)
public static class ApiEndpoints extends WebSecurityConfigurerAdapter {

    private final EndUserDetailsService endUserDetailsService;

    public ApiEndpoints(@Autowired @Qualifier("endUserDetailsService")
                        EndUserDetailsService endUserDetailsService) {
        this.endUserDetailsService = endUserDetailsService;
    }

    protected void configure(HttpSecurity http) throws Exception {
        http.requestMatchers(r -> r.mvcMatchers( ...patterns: "/api/**"))
                .authorizeRequests(a -> {
                            a.mvcMatchers( ...patterns: "/api/**").hasRole("ADMIN");
                            a.anyRequest().denyAll();
                        })
                .httpBasic(withDefaults())
                .userDetailsService(this.endUserDetailsService)
                .cors(withDefaults())
                .csrf()
                .disable();
    }
}
```

Рисунок 3.12 - Конфігурація захисту управління користувачами та клієнтами

```
@EnableConfigurationProperties(AuthorizationServerConfigurationProperties.class)
@SpringBootApplication
public class AuthorizationServerApplication {

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

Рисунок 3.13 - Точка старту додатка

3.2.2 Сутності домену

Сутність — це об’єкт домену. Як правило, сутність представляє таблицю в реляційній базі даних, і кожен екземпляр сутності відповідає рядку в цій таблиці. В проекті усього три основних сутності – це токен (рисунки 3.14 та

3.15), користувач (рисунок 3.16) та клієнт (рисунок 3.17), та, який має два підвиди – JWT, та прозорий. До JWT токенів можна віднести токен доступу та ID токен, а до прозорих – токен відновлення.

```java
@MappedSuperclass
@Getter
@Setter
@ToString
@NoArgsConstructor
@EqualsAndHashCode(callSuper = true)
public abstract class Token extends AbstractPersistable<Long> {

  @NotBlank
  @Size(max = 2000)
  private String value;
  @NotNull
  private LocalDateTime expiry;
  private boolean revoked;

}
```

Рисунок 3.14 - Сутність токен

```java
@Entity
@DiscriminatorValue("jwt")
@Setter
@ToString
@EqualsAndHashCode(callSuper = true)
public class JsonWebToken extends Token {

  @NotNull
  private boolean accessToken;

  public boolean isAccessToken() { return accessToken; }

  public boolean isIdToken() { return !accessToken; }

}
```

Рисунок 3.15 - JWT токен

```java
@Entity
@EntityListeners(AuditingEntityListener.class)
@Getter
@Setter
@ToString(callSuper = true)
@NoArgsConstructor
@EqualsAndHashCode(callSuper = true)
public class UserEntity extends ResourceEntity {

    @Column(unique = true)
    @NotBlank
    @Size(min = 3, max = 50)
    private String userName;
    @NotBlank
    @Size(min = 1, max = 100)
    private String familyName;
    @NotBlank
    @Size(min = 1, max = 100)
    private String givenName;
    @Size(max = 100)
    private String userType;
    @Size(max = 50)
    private String locale;
    @NotNull
    private boolean active;
    @NotBlank
    @Size(min = 8, max = 255)
    private String password;
    private String email;
    @ElementCollection
    private Set<String> roles = new HashSet<>();
```

Рисунок 3.16 - Сутність користувач

```java
@Entity
@Getter
@Setter
@ToString(callSuper = true)
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(callSuper = true)
public class RegisteredClient extends AbstractPersistable<Long> {

  private UUID identifier;
  @NotBlank
  @Size(max = 100)
  @Column(unique = true)
  private String clientId;
  @Size(max = 100)
  private String clientSecret;
  @NotNull
  @Enumerated(EnumType.STRING)
  private AccessTokenFormat accessTokenFormat;
  @NotEmpty
  @ElementCollection(fetch = FetchType.EAGER)
  private Set<GrantType> grantTypes = new HashSet<>();
  @NotEmpty
  @ElementCollection(fetch = FetchType.EAGER)
  private Set<String> redirectUris = new HashSet<>();
  @NotEmpty
  @ElementCollection(fetch = FetchType.EAGER)
  private Set<String> corsUris = new HashSet<>();
```

Рисунок 3.17 - Сутність клієнт

3.2.3 Модуль автентифікації

Модуль автентифікації відповідає за автентифікацію користувача, видачу токена доступу, ID токена, токена відновлення, перевірку токена на актуальність, а також відповідає за маркування токена як не валідного.

Запити на цей ендпоінт приходять, коли користувач вводить логін та пароль і натискає кнопку «Sign In». Якщо логін або пароль невірні – користувач отримає повідомлення про це. Якщо логін і пароль вірні – відбудеться реєстрація

коду для отримання токену и редирект на адресу редиректу, яка вказана у конфігурації клієнта.

```
/unused, SpringMVCViewInspection/
@PreAuthorize("isAuthenticated()")
@GetMapping(OpenIdEndpoints.AUTHORIZATION_ENDPOINT)
public String authorizationRequest(
        @RequestParam("response_type") @Pattern(regexp = "code") String responseType,
        @RequestParam("scope") String scope,
        @RequestParam("client_id") String clientId,
        @RequestParam("redirect_uri") URI redirectUri,
        @RequestParam(name = "state", required = false) String state,
        @AuthenticationPrincipal EndUserDetails endUserDetails) {
    if (endUserDetails == null || endUserDetails.getIdentifier() == null) {
        throw new BadCredentialsException("No user");
    }
    Optional<RegisteredClient> registeredClient =
            registeredClientService.findOneByClientId(clientId);
    if (registeredClient.isEmpty()) {
        throw new InvalidClientIdError(clientId);
    }
    RegisteredClient client = registeredClient.get();
    if (!client.getRedirectUris().contains(redirectUri.toString())) {
        throw new InvalidRedirectUriError(redirectUri.toString());
    }
    if (!client.getGrantTypes().contains(GrantType.AUTHORIZATION_CODE)) {
        return redirectError(redirectUri, errorCode: "unauthorized_client",
                errorDescription: "The client is not authorized to request an authorization code using this method",
                state);
    }
    Set<String> scopes = new HashSet<>(Arrays.asList(scope.split( regex: " ")));
    AuthorizationCode authorizationCode = authorizationCodeService.createAndStoreAuthorizationState(
            clientId, redirectUri, scopes,
            endUserDetails.getIdentifier() != null ? endUserDetails.getIdentifier().toString() : "");

    return "redirect:" + redirectUri + "?code=" + authorizationCode.getCode() + "&state=" + state;
}
```

Рис. 3.18 Ендпоінт авторизації

Після надання гранта клієнту, клієнт відправляє запит з кодом на ендпоінт для отримання токену, зазвичай /token, та отримує токен доступу, токен відновлення, та ID токен, який містить інформацію про користувача, який автентифікувався. Також тут є функціонал для відновлення застарілого токену доступу, для цього треба вказати параметр grant_type як REFRESH_TOKEN та додати до запиту токен відновлення (рисунок 3.19).

```java
@ResponseBody
@PostMapping(OpenIdEndpoints.TOKEN_ENDPOINT)
public ResponseEntity<TokenResponse> getToken(
        @RequestHeader(name = "Authorization", required = false) String authorizationHeader,
        @ModelAttribute("token_request") TokenRequest tokenRequest) {
    if (tokenRequest.getGrant_type().equalsIgnoreCase(GrantType.AUTHORIZATION_CODE.getGrant())) {
        return authorizationCodeTokenEndpointService.getTokenResponseForAuthorizationCode(authorizationHeader, tokenRequest);
    } else if (tokenRequest.getGrant_type().equalsIgnoreCase(GrantType.REFRESH_TOKEN.getGrant())) {
        return refreshTokenEndpointService.getTokenResponseForRefreshToken(authorizationHeader, tokenRequest);
    } else {
        return ResponseEntity.badRequest().body(new TokenResponse( error: "unsupported_grant_type"));
    }
}
```

Рисунок 3.19 - Ендпоінт отримання токену

Після цього можна перевірити токен на актуальність, за допомогою ендпоінту /introspect, який поверне актуальний статус токена (рисунок 3.20).

```java
@ResponseBody
@PostMapping(OpenIdEndpoints.INTROSPECTION_ENDPOINT)
public ResponseEntity<IntrospectionResponse> introspect(
        @RequestHeader("Authorization") String authorizationHeader,
        @ModelAttribute("introspection_request") IntrospectionRequest introspectionRequest) {
    try {
        if (AuthenticationUtil.fromBasicAuthHeader(authorizationHeader) == null) {
            return reportInvalidClientError();
        }
        String tokenValue = introspectionRequest.getToken();
        if (tokenValue == null || tokenValue.isBlank()) {
            return ResponseEntity.ok(new IntrospectionResponse( active: false));
        }
        JsonWebToken jsonWebToken = tokenService.findJsonWebToken(tokenValue);
        if (jsonWebToken == null) {
            return ResponseEntity.ok(new IntrospectionResponse( active: false));
        }
        return ResponseEntity.ok(getIntrospectionResponse(jsonWebToken));
    } catch (BadCredentialsException ex) {
        return reportInvalidClientError();
    }
}
```

Рисунок 3.20 - Ендпоінт валідації токена

Якщо користувач захоче «анулювати токен» тобто зробити його недійсним – необхідно буде відправити запит на ендпоінт /revoke. Після цього токен стане не дійсним і ендпоінт /introspect поверне статус токена, який показує що він невалідний (рисунок 3.21).

```
@ResponseBody
@PostMapping(OpenIdEndpoints.REVOCATION_ENDPOINT)
public ResponseEntity<RevocationResponse> revoke(
        @ModelAttribute("revocation_request") RevocationRequest revocationRequest) {
    JsonWebToken jsonWebToken = tokenService.findJsonWebAccessToken(revocationRequest.getToken());
    if (jsonWebToken == null) {
        return ResponseEntity.badRequest().body(new RevocationResponse( status: null,  error: "invalid_request"));
    }
    tokenService.remove(jsonWebToken);
    return ResponseEntity.ok(new RevocationResponse( status: "ok",  error: null));
}
```

Рисунок 3.21 - Ендпоінт для видалення токена

Також, користувач може не використовувати ендпоінт для перевірки підпису кожного токена доступу, він може запросити публічний ключ для перевірки підпису токена через /jwks ендпоінт, а потім використовувати його для локальної перевірки підпису (рисунок 3.22).

```
@CrossOrigin(originPatterns = "*", allowCredentials = "true", allowedHeaders = "*")
@RestController
@RequestMapping("/jwks")
@AllArgsConstructor
public class JwksController {

  private final JwtPki jwtPki;

  @GetMapping
  public Map<String, Object> jwksEndpoint() { return jwtPki.getJwkSet().toJSONObject(); }
}
```

Рисунок 3.22 - Ендпоінт /jwks

Для того, щоб отримати всю інформацію про сервер автентифікації, клієнт може відправити запит на /.well-known/openid-configuration ендпоінт. Це може бути використано для автоматичної конфігурації клієнта, щоб він використовував цей сервер, як постачальника ідентифікаційних даних (рисунок 3.23).

```java
@CrossOrigin(originPatterns = "*", allowCredentials = "true", allowedHeaders = "*")
@RestController
@RequestMapping(OpenIdEndpoints.DISCOVERY_ENDPOINT)
@AllArgsConstructor
public class DiscoveryController {

  private final JwtPki jwtPki;

  @GetMapping
  public Discovery discoveryEndpoint() {
    Discovery discovery = new Discovery();
    discovery.setAuthorization_endpoint(jwtPki.getIssuer() + OpenIdEndpoints.AUTHORIZATION_ENDPOINT);
    discovery.setIssuer(jwtPki.getIssuer());
    discovery.setToken_endpoint(jwtPki.getIssuer() + OpenIdEndpoints.TOKEN_ENDPOINT);
    discovery.setIntrospection_endpoint(jwtPki.getIssuer() + OpenIdEndpoints.INTROSPECTION_ENDPOINT);
    discovery.setRevocation_endpoint(jwtPki.getIssuer() + OpenIdEndpoints.REVOCATION_ENDPOINT);
    discovery.setUserinfo_endpoint(jwtPki.getIssuer() + OpenIdEndpoints.USER_INFO_ENDPOINT);
    discovery.setJwks_uri(jwtPki.getIssuer() + "/jwks");
    discovery.getGrant_types_supported().add(GrantType.AUTHORIZATION_CODE.getGrant());
    discovery.getResponse_types_supported().add("code");
    discovery.getScopes_supported().add(Scope.OPENID.name().toLowerCase());
    discovery.getScopes_supported().add(Scope.OFFLINE_ACCESS.name().toLowerCase());
    discovery.getScopes_supported().add(Scope.PROFILE.name().toLowerCase());
    discovery.getScopes_supported().add(Scope.EMAIL.name().toLowerCase());
    discovery.getResponse_modes_supported().add("query");
    discovery.getResponse_modes_supported().add("form_post");
    discovery.getSubject_types_supported().add("public");
    discovery.getId_token_signing_alg_values_supported().add("RS256");
    discovery.getToken_endpoint_auth_methods_supported().add("client_secret_basic");
```

Рисунок 3.23 - Ендпоінт /.well-known/openid-configuration

3.2.4 Модуль користувачів та клієнтів

Для того, щоб створити користувача – необхідно надіслати POST запит на /api/users ендпоінт. Це може бути використано для реєстрації нових користувачів (рисунок 3.24).

```java
@PostMapping(USER_ENDPOINT)
public ResponseEntity<UserResource> createUser(@Valid @RequestBody CreateUserResource createScimUserResource) {
    UserEntity scimUserEntity = createScimUserResourceMapper.mapResourceToEntity(createScimUserResource);
    scimUserEntity = userService.createUser(scimUserEntity);
    return ResponseEntity.ok().body(scimUserResourceMapper.mapEntityToResource(scimUserEntity));
}
```

Рисунок 3.24 - Ендпоінт для реєстрації нових користувачів

Також наявні ендпоінти для отримання всіх користувачів та конкретного користувача за допомогою ідентифікатора (рисунок 3.25).

```
@GetMapping(USER_ENDPOINT)
public List<UserListResource> getAllUsers() {
    return userService.findAllUsers().stream()
            .map(scimUserListResourceMapper::mapEntityToResource)
            .collect(Collectors.toList());
}

@GetMapping(USER_ENDPOINT + "/{userId}")
public ResponseEntity<UserResource> getUser(@PathVariable("userId") UUID userIdentifier) {
    return userService.findUserByIdentifier(userIdentifier).map(ue ->
            ResponseEntity.ok().body(scimUserResourceMapper.mapEntityToResource(ue)))
            .orElse(ResponseEntity.notFound().build());
}
```

Рисунок 3.25 - Ендпоінти для отримання списку користувачів або окремого користувача за допомогою ідентифікатора

Так іноді необхідно корегувати інформацію про користувача, то для цього спеціально створено ендпоінт /api/users, який приймає PUT запити (рисунок 3.26).

```
@PutMapping(USER_ENDPOINT + "/{userId}")
public ResponseEntity<UserResource> updateUser(@PathVariable("userId") UUID userIdentifier,
                                               @RequestBody @Valid UserResource scimUserResource) {
    UserEntity ue = userService.updateUser(userIdentifier, scimUserResource);
    return ResponseEntity.ok().body(scimUserResourceMapper.mapEntityToResource(ue));
}
```

Рисунок 3.26 - Ендпоінт для корегування інформації про користувача

Також було створено ендпоінт /api/users, який приймає DELETE запити для видалення користувача з бази даних, щоб він вже не зміг автентифікуватися за допомогою своїх ідентифікаційних даних (рисунок 3.27).

```
@ResponseStatus(NO_CONTENT)
@DeleteMapping(USER_ENDPOINT + "/{userId}")
public void deleteUser(@PathVariable("userId") UUID userIdentifier) { userService.deleteUser(userIdentifier); }
```

Рисунок 3.27 - Ендпоінт для видалення користувачів

Також в нашому додатку є можливість отримати список клієнтів або інформації про конкретного клієнта за допомогою його ідентифікатора, але користувач, який надсилає запит повинен мати права адміністратора. Для цього необхідно надіслати GET запит на /api/clients ендпоінт (рисунок 3.28).

```
@RestController
@RequestMapping("/api/clients")
@AllArgsConstructor
public class ClientController {

  private final RegisteredClientService registeredClientService;

  @GetMapping
  public List<RegisteredClientResource> clients() {
    return registeredClientService.findAll().stream()
        .map(RegisteredClientResource::new)
        .collect(Collectors.toList());
  }

  @GetMapping("/{clientId}")
  public ResponseEntity<RegisteredClientResource> client(@PathVariable("clientId") UUID clientId) {
    return registeredClientService.findOneByIdentifier(clientId)
            .map(c -> ResponseEntity.ok(new RegisteredClientResource(c)))
            .orElse(ResponseEntity.notFound().build());
  }
```

Рисунок 3.28 - Ендпоінти для отримання списка клієнтів або інформації про клієнта

Для того, щоб створити нового клієнта – необхідно надіслати POST запит на адресу /api/clients (рисунок 3.29).

```
@PostMapping
public ResponseEntity<RegisteredClientResource> registerNewClient(
        @RequestBody @Valid ModifyRegisteredClientResource modifyRegisteredClientResource) {
  RegisteredClient registeredClient = new RegisteredClient(modifyRegisteredClientResource);
  registeredClient = this.registeredClientService.create(registeredClient);
  return ResponseEntity.ok().body(new RegisteredClientResource(registeredClient));
}
```

Рисунок 3.29 - Ендпоінт для реєстрації нового клієнта

Для того, щоб оновити дані про клієнту – необхідно надіслати PUT запит на адресу /api/clients (рисунок 3.30).

Рисунок 3.30 - Ендпоінт для оновлення клієнта

Для того, щоб видалити клієнта – необхідно надіслати DELETE запит на адресу /api/clients (рисунок 3.31).

Рисунок 3.31 - Ендпоінт для видалення клієнта

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

3.2.3 Інструкція користувача

Для запуску програми, на комп’ютері повинна бути встановлена версія Java 14. Перед запуском необхідно, щоб база даних була запущена, там мі будем зберігати дані про користувачів, клієнтів та токени. Ми використовуємо утиліту Docker для запуску контейнера, в якому буде працювати база даних PostgreSQL. Для старта контейнера необхідно виконати команду «docker run -p 5432:5432 -e POSTGRES_PASSWORD=postgres --name test -d postgres».

Запустити сервер можна командою java –jar build/libs/ authorizationserver-1.0.0-SNAPSHOT.jar. Також можна задати POSTHRES_USERNAME та

POSTGRES_PASSWORD як змінні середовища, для того, 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 – ip-адреса комп’ютеру (рисунок 3.32).

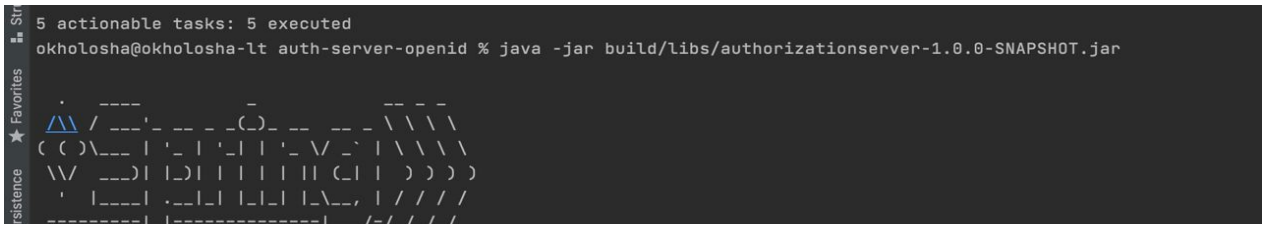

Рисунок 3.32 - Команда для запуску сервера з термінала

Після цього можна побачити, що сервер Tomcat стартував на порті 9090 із контекстним шляхом до сервлетів /auth (рисунок 3.33).

Рисунок 3.33 - Записи у журналі подій про старт сервера авторизації

Щоб отримати токен, ми заздалегідь підготували клієнта з client_id: «client-id» та client_secret: «client-secret», та користувача з правами адміністратора з логіном «admin» та паролем «admin». Спершу необхідно отримати грант авторизації від клієнта, це можна зробити за допомогою перенаправлення клієнта на сервер авторизації, з переданими параметрами, переданими в посиланні: localhost:9090/auth/authorize?response_type=code &scope=openid&client_id=client-id&state=8& redirect_uri=http://localhost:8080/demo-client/login/oauth2/code/demo.

Параметр response_type із значенням code означає, що вернути у відповідь сервер авторизації повинен код, за допомогою якого можна отримати токен.

Параметр scope із значенням openid, говорить, що необхідно дозволити клієнту запитувати інформацію про власника ресурсу. Параметр redirect_uri вказує куди повинен бути перенаправлений запит після успішної авторизації. Параметр state – це випадково згенероване число яке перевіряється за стороні клієнта, щоб уникнути XSRF атак.

Спробуємо перейти в браузері за посиланням, вказаним вище (рисунок 3.34):

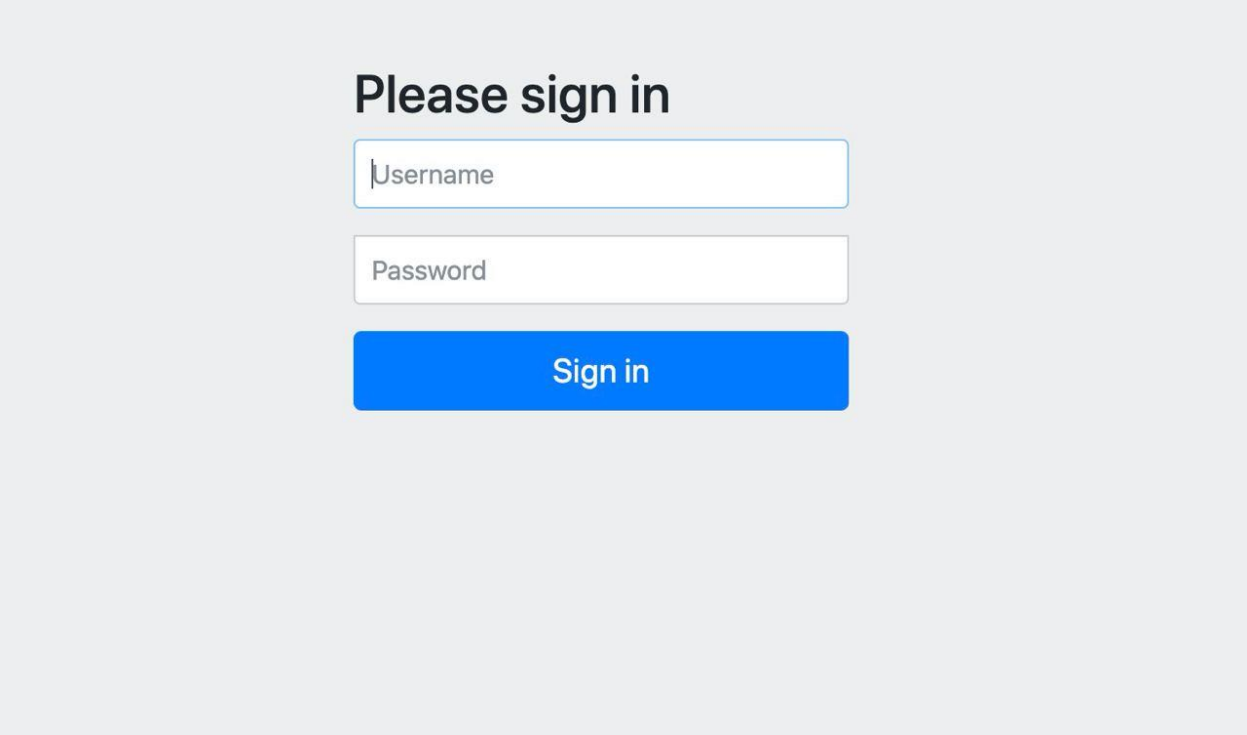

Рисунок 3.34 - Сторінка логіну користувача

Після того, як користувач вводить правильні ідентифікаційні дані, його буде перенаправлено на посилання вказане в параметрі redirect_uri (рисунок 3.35).

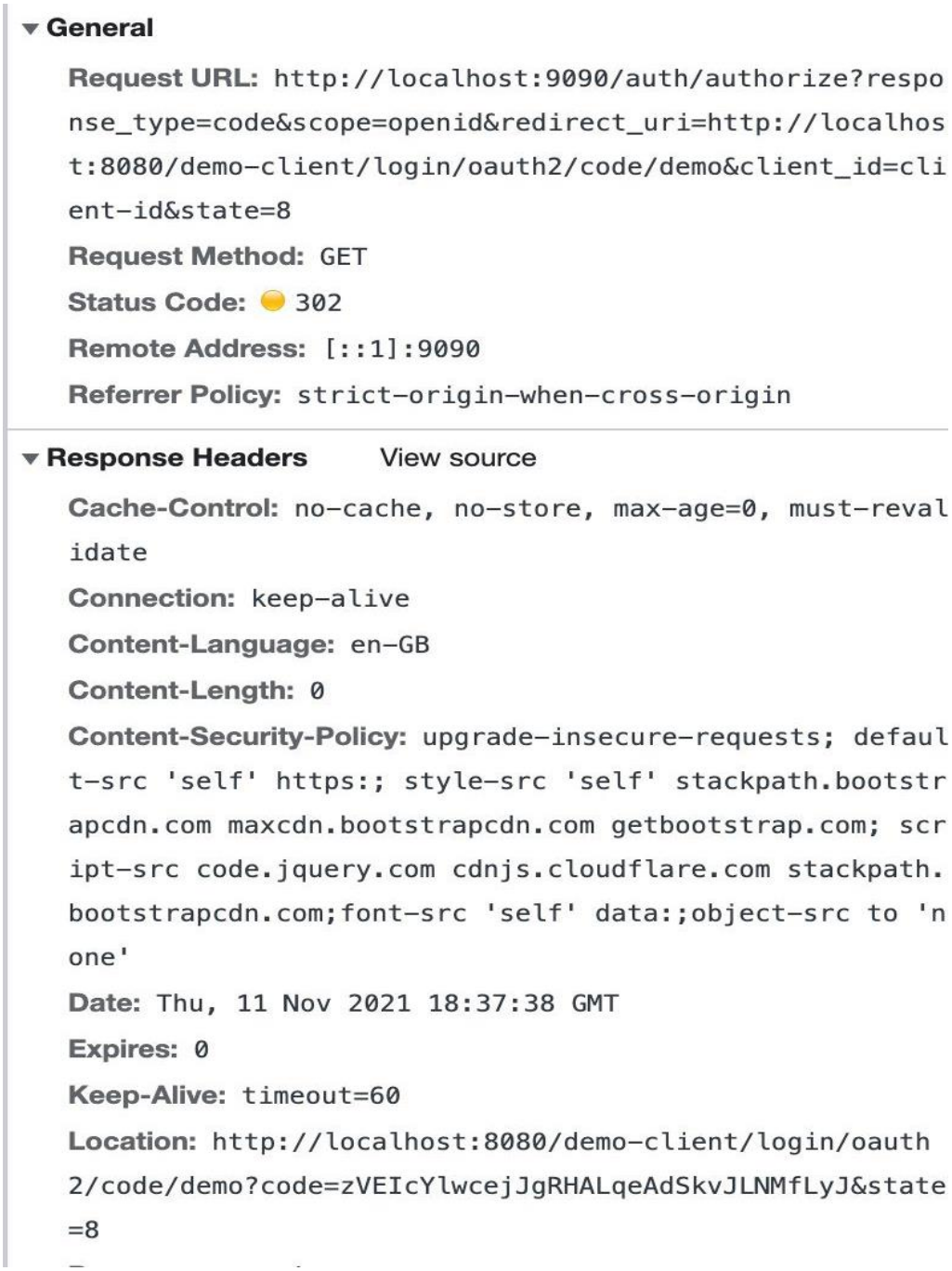

Рисунок 3.35 - Параметри запиту, який було перенаправлено на ендпоінт клієнта

Необхідно звернути увагу параметр «Location», в якому передана адреса перенаправлення браузера. У параметрах адреси ми можемо побачити параметр code та параметр state для підтвердження правильності запиту. Після цього беремо параметр code та формуємо запит для отримання токену. параметр

grant_type необхідно вказати як AUTHORIZATION_CODE, що означатиме, що клієнт хоче обміняти код авторизації на токен, redirect_uri для валідації клієнта з метою підвищення рівня захисту, code представляє собою код авторизації, який було отримано після успішної автентифікації клієнта, client_id та client_secret – ідентифікаційні дані клієнта (рисунок 3.36).

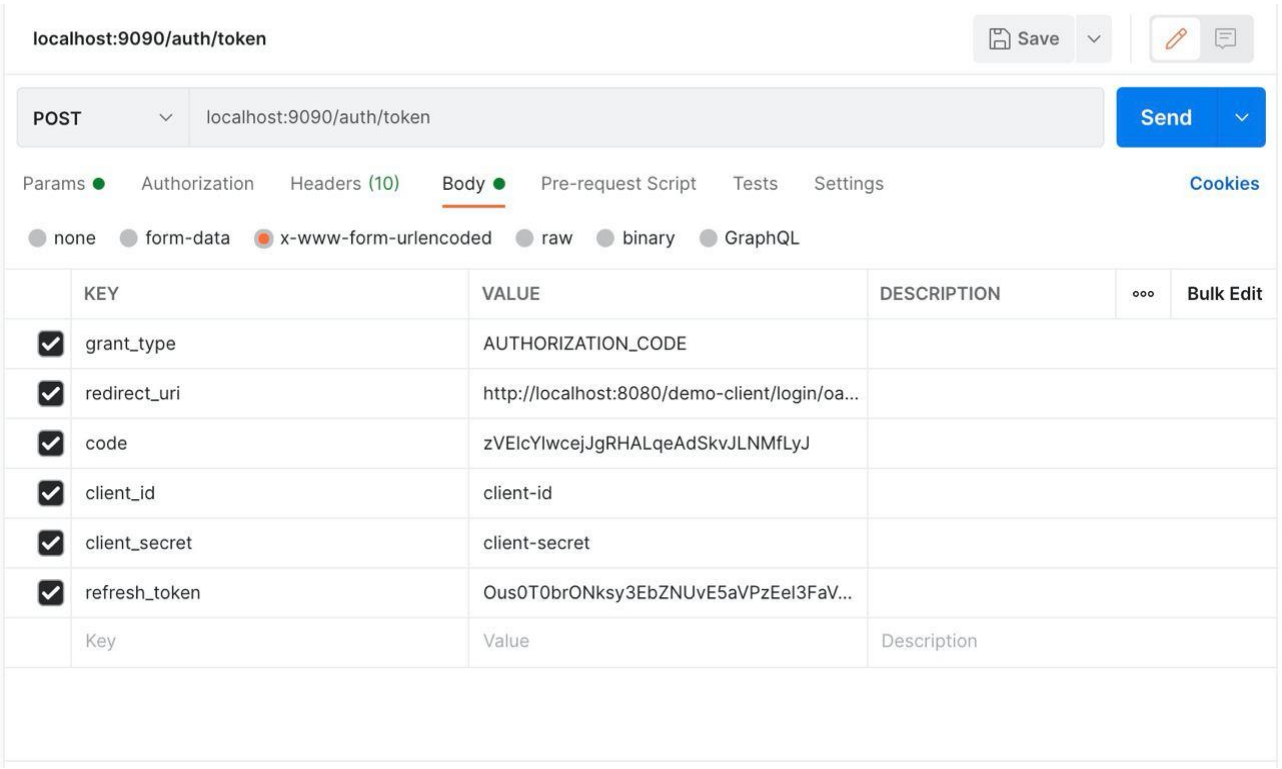

Рисунок 3.36 - Формування запиту для отримання токену

Відправляємо запит та отримуємо токен доступу, токен відновлення та ID токен (рисунок 3.37).

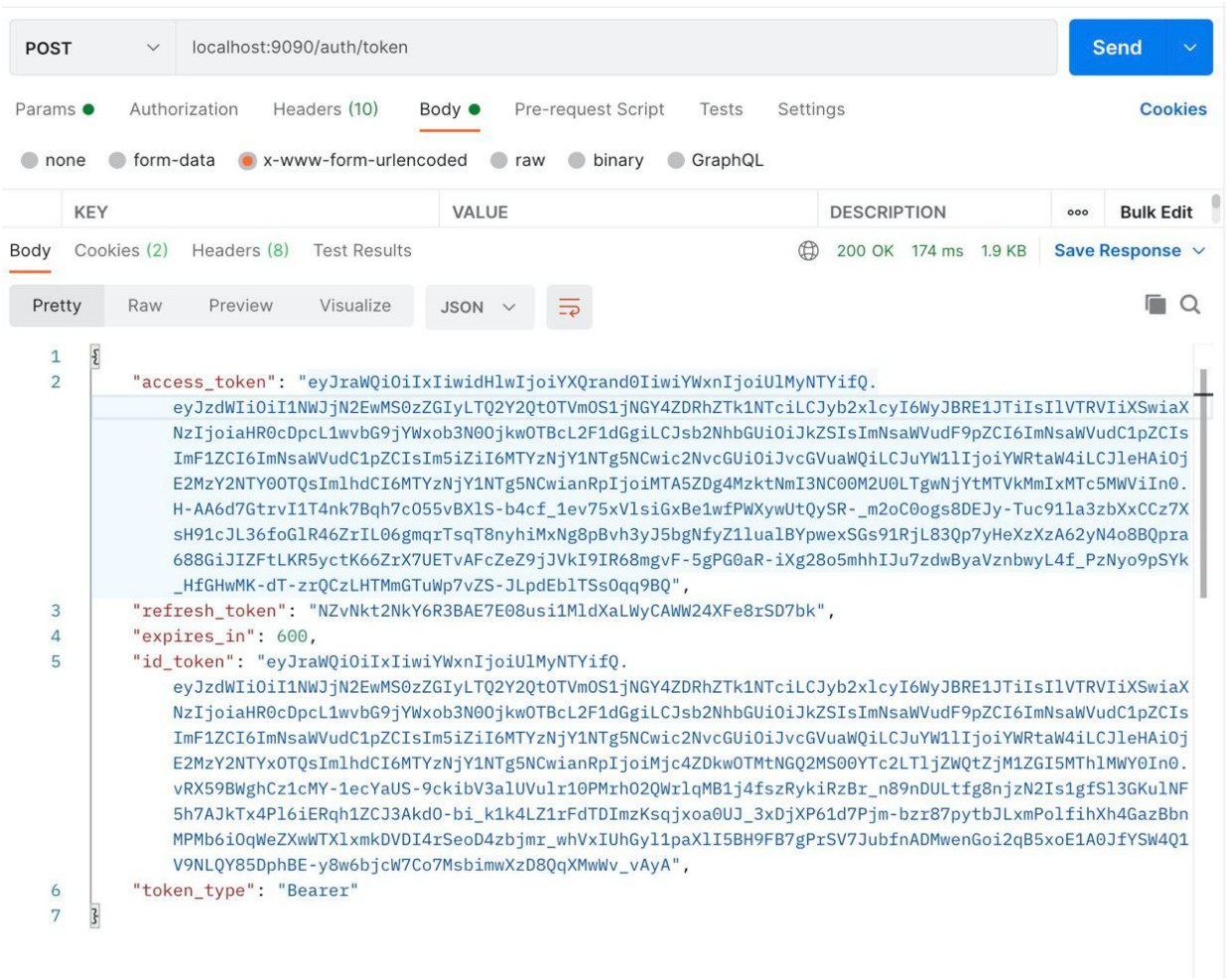

Рисунок 3.37 - Отримані токени доступу, відновлення та ID токен

Тепер сформуємо запит для перевірки. Необхідно підставити значення в попереднього запиту з поля acess_token, в поле token у поточному запиті, також в заголовки необхідно додати «Authorization: Basic Y29uZmlkZW50aWFsLWp3dDpkZW1v». Це client_id:client_secret, які були зашифровані в Base64 формат (рисунки 3.38 та 3.39).

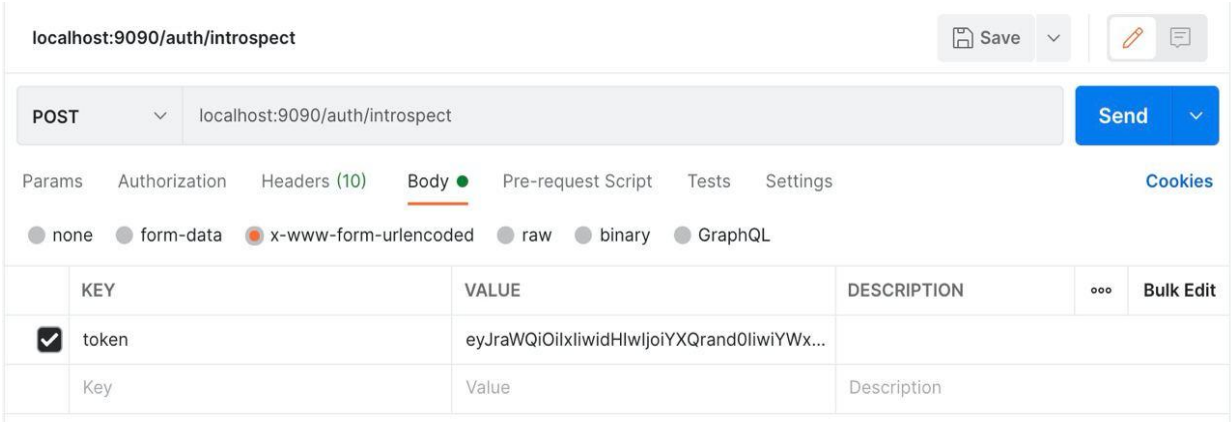

Рисунок 3.38 - Формування запиту для перевірки токена доступу

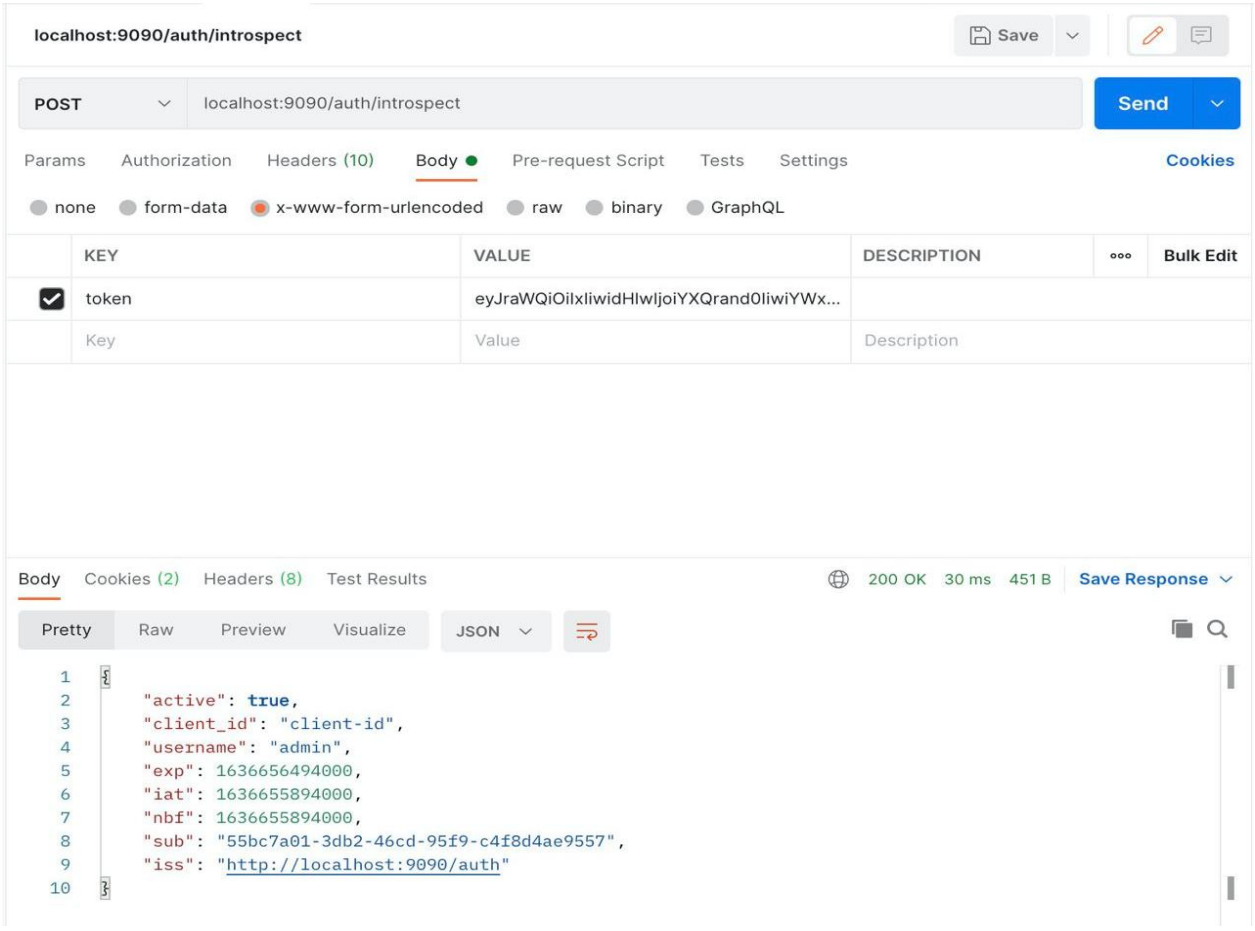

Рисунок 3.39 - Відповідь з валідації токена

Як бачимо з відповіді, токен валідний. Поле active: true означає, що токен активний. Поле client_id повертає ідентифікатор поточного клієнта, username

повертає ідентифікаційне ім’я користувача, який був авторизований та для якого було створено токен, exp – час, коли ID токен буде невалідний та не можливий для обробки, iat – час, коли JWT токен був запитаний, nbf – час, до якого токен не доступний для обробки, sub – ідентифікатор суб’єкта(користувача), iss – ідентифікатор сервера автентифікації.

Щоб отримати інформацію про користувача, який надав грант на отримання токена, то необхідно запитати ендпоінт /userinfo. Вказавши в заголовку «Authorization: Bearer <ID token>», де ID token це ID токен, який ми отримали разом із токеном доступу. Підготуємо запит і додамо заголовок (рисунок 3.40):

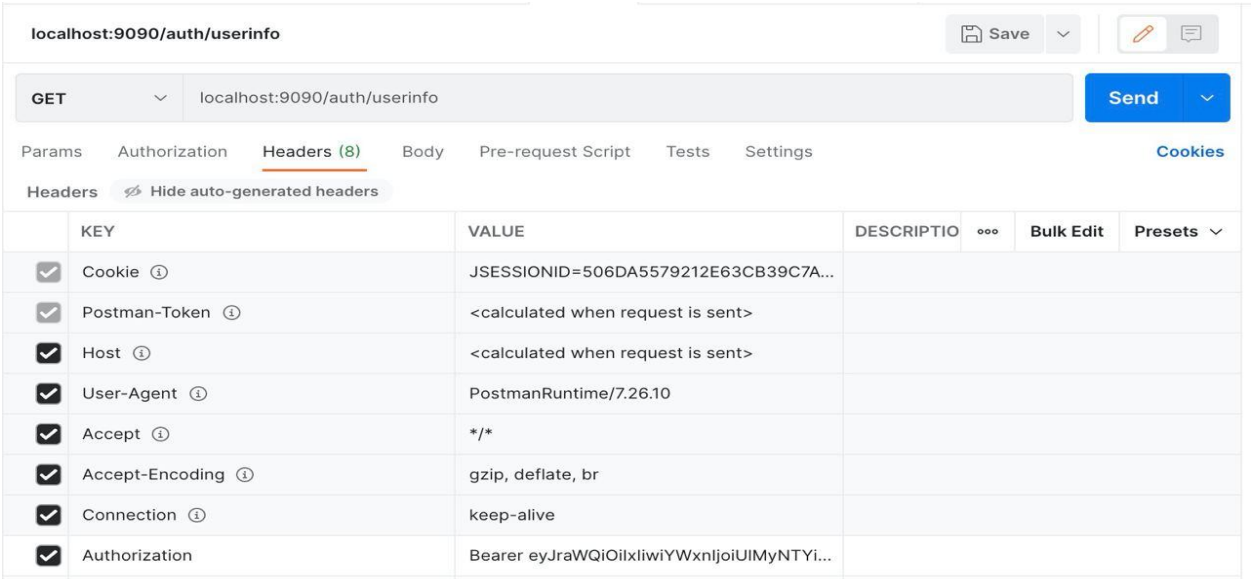

Рисунок 3.40 - Формування запиту для отримання інформації про користувача

Як бачимо, мі отримали інформацію про користувача. Sub- ідентифікатор користувача в системі, name – юзернейм користувача, given_name – ім’я користувача, family_name – фамілія користувача, email – його імейл, roles – ролі користувача в системі, тут ми можемо бачити, що користувач також є і адміністратором (рисунок 3.41).

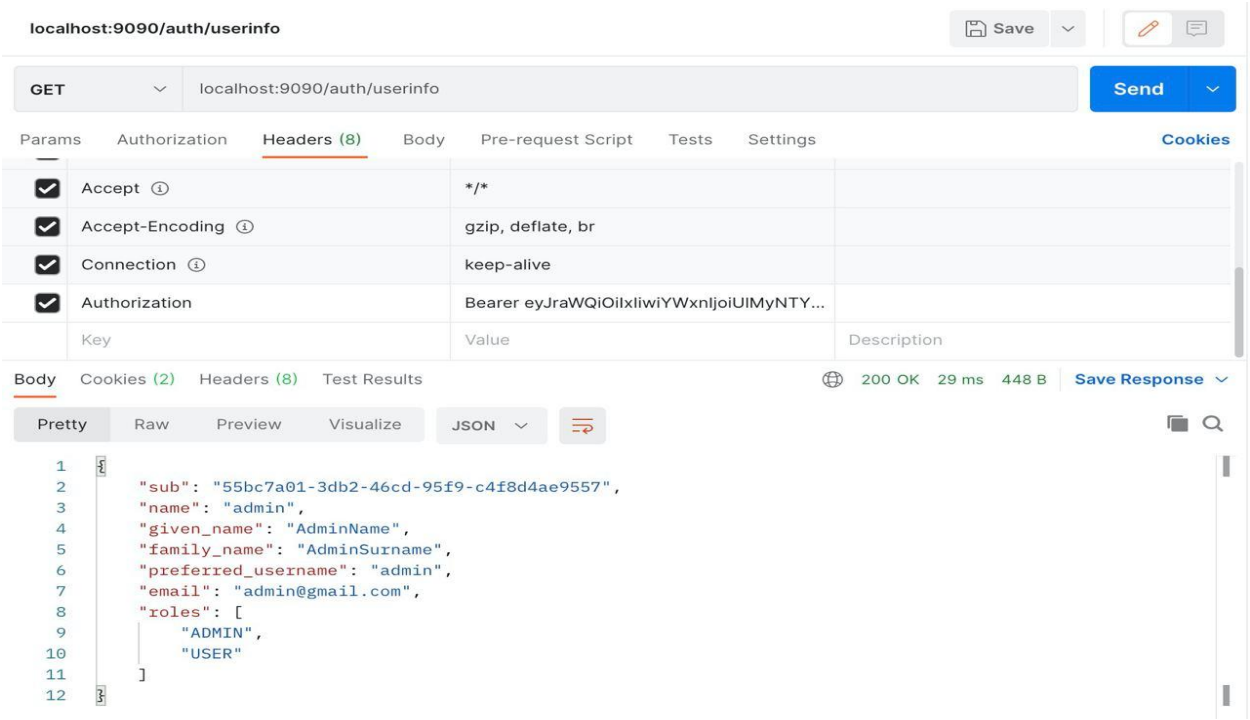

Рисунок 3.41 - Отримання інформації про поточного користувача

Тепер спробуємо видалити токен, сформувавши запит на ендпоінт /revoke. У поле token необхідно передати параметр токен доступу, який було отримано раніше (рисунок 3.42).

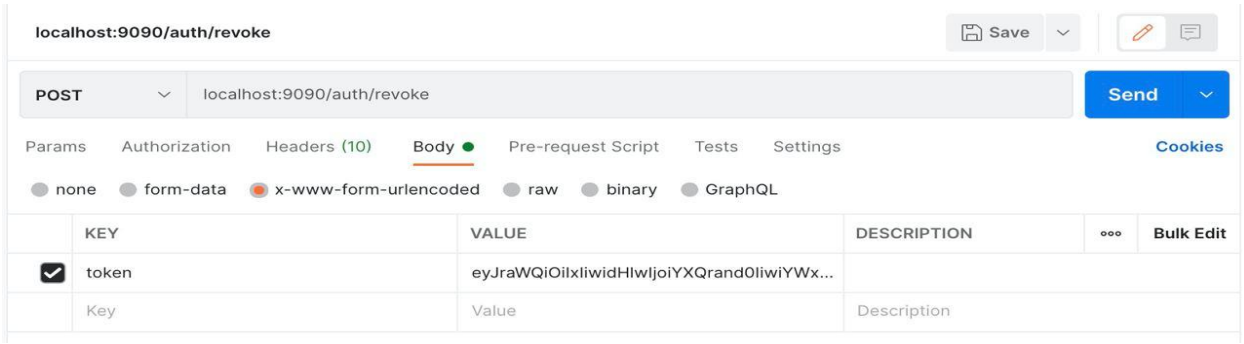

Рисунок 3.42 - Формування запиту на / revoke ендпоінт

Теперь надішлемо запит и подивимося на результат (рисунок 3.43):

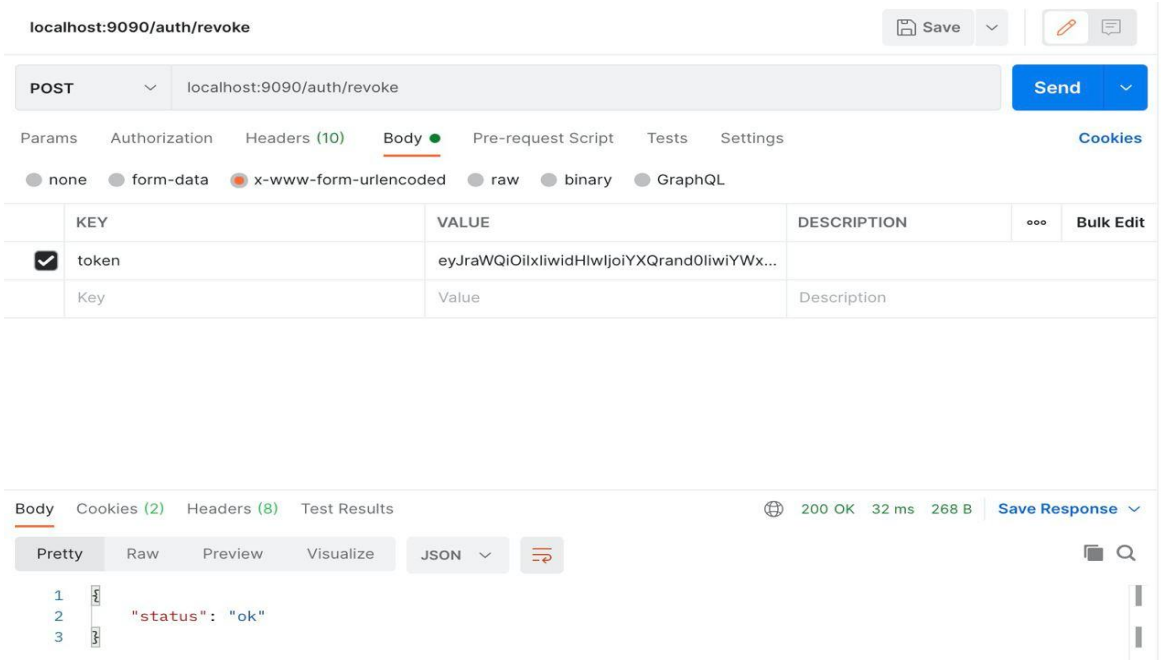

Рисунок 3.43 - Видалення токену

Як бачимо зі стасуса, токен видалився. Спробуємо перевірити, що буде, якщо токен буде невалідним. Ми взяли той же токен, але зараз він вже видалений (рисунок 3.44):

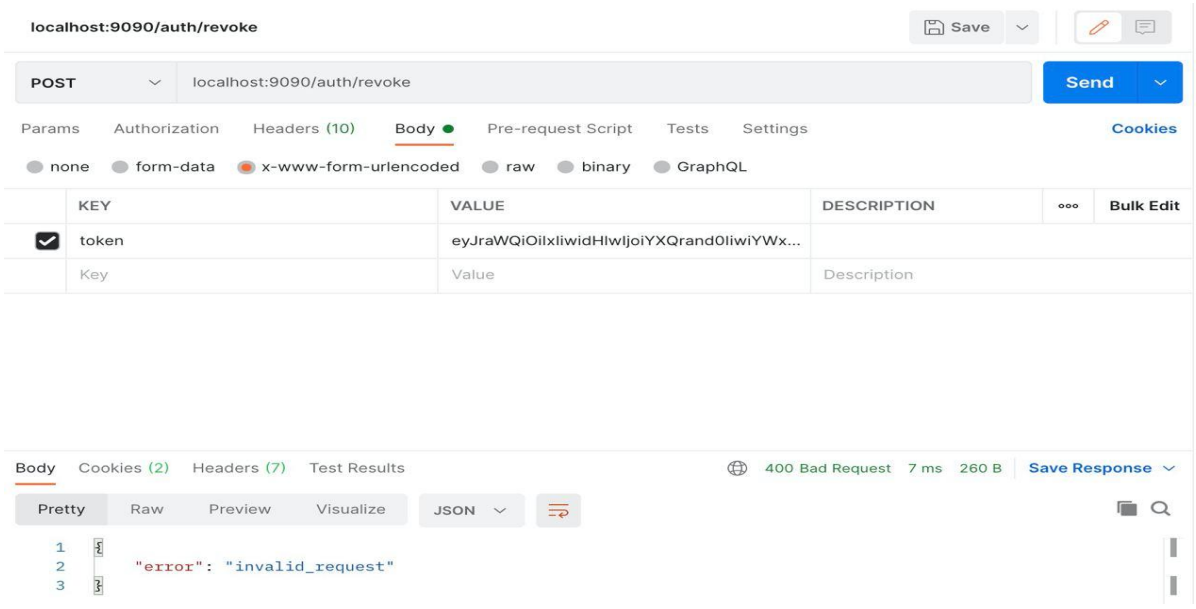

Рисунок 3.44 - Видалення невалідного токена

Як бачимо, відповідь з сервера означає, що ми передали невалідний токен. Тепер спробуємо перевірити цей токен, чи дійсний він (рисунок 3.45).

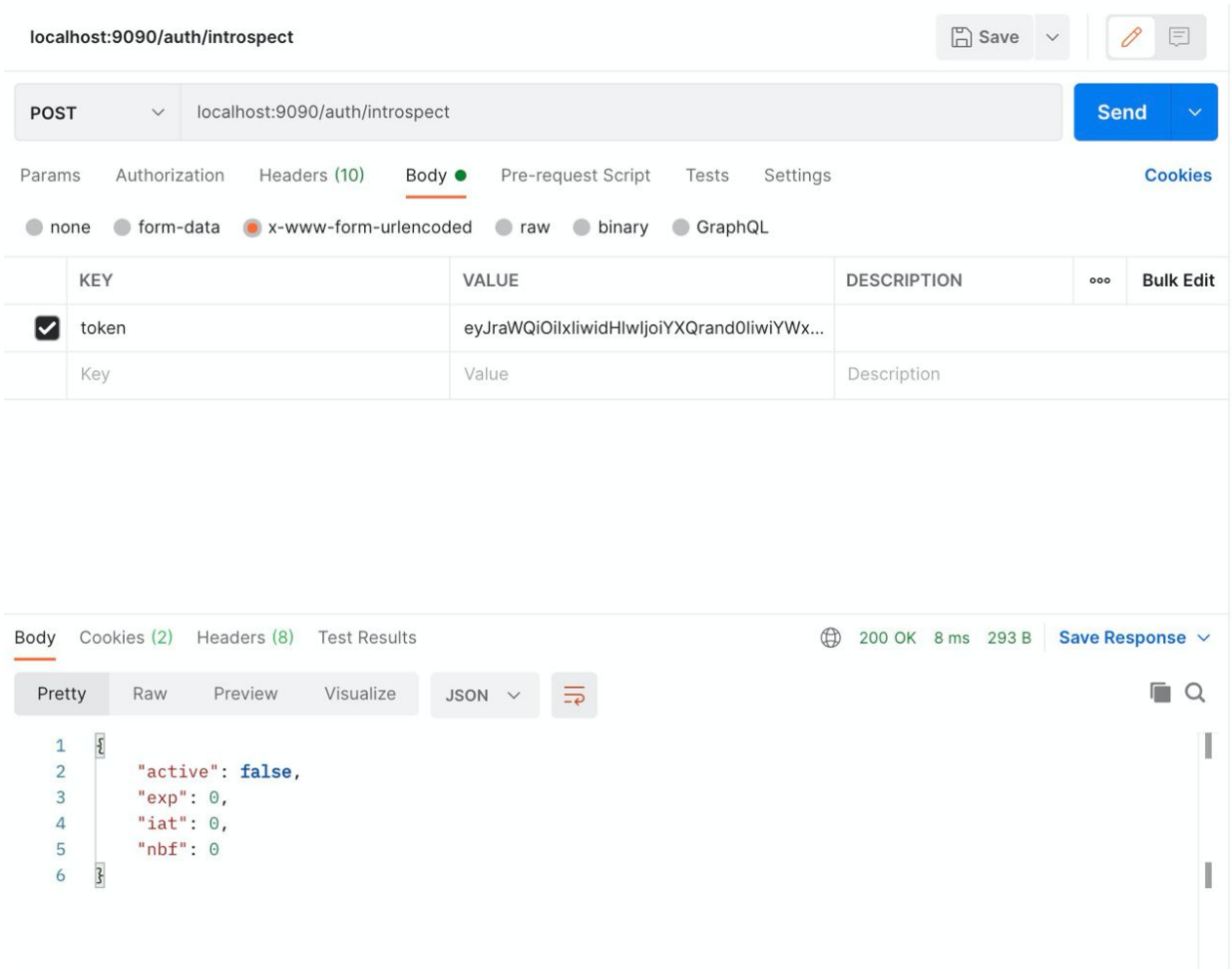

Рисунок 3.45 - Перевірка недійсного токену

Як мі бачимо, токен недійсний, отже можна зробити висновок, що система працює нормально.

Спробуємо відновити токен доступу за допомогою токена відновлення. Параметр grant_type змінюємо значення на REFRESH_TOKEN та додаємо в параметр refresh_token значення токена відновлення, яке прийшло нам разом з токеном доступу, решту параметрів залишаємо такими ж, які використовували для отримання токена (рисунок 3.46).

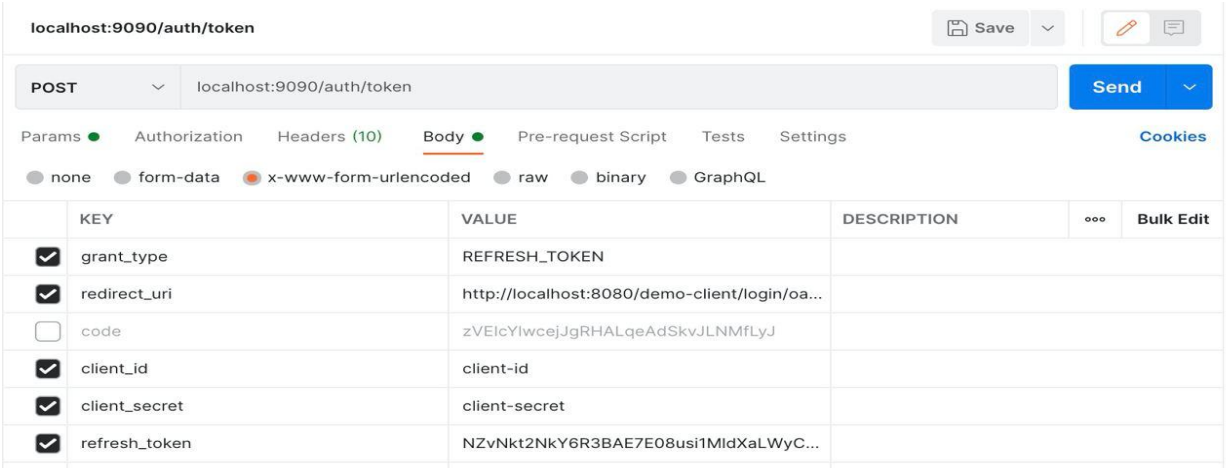

Рисунок 3.46 - Формування запиту для відновлення токена доступу

Відправляємо запит, і бачимо. Що нам прийшов новий токен доступу і новий токен відновлення (рисунок 3.47).

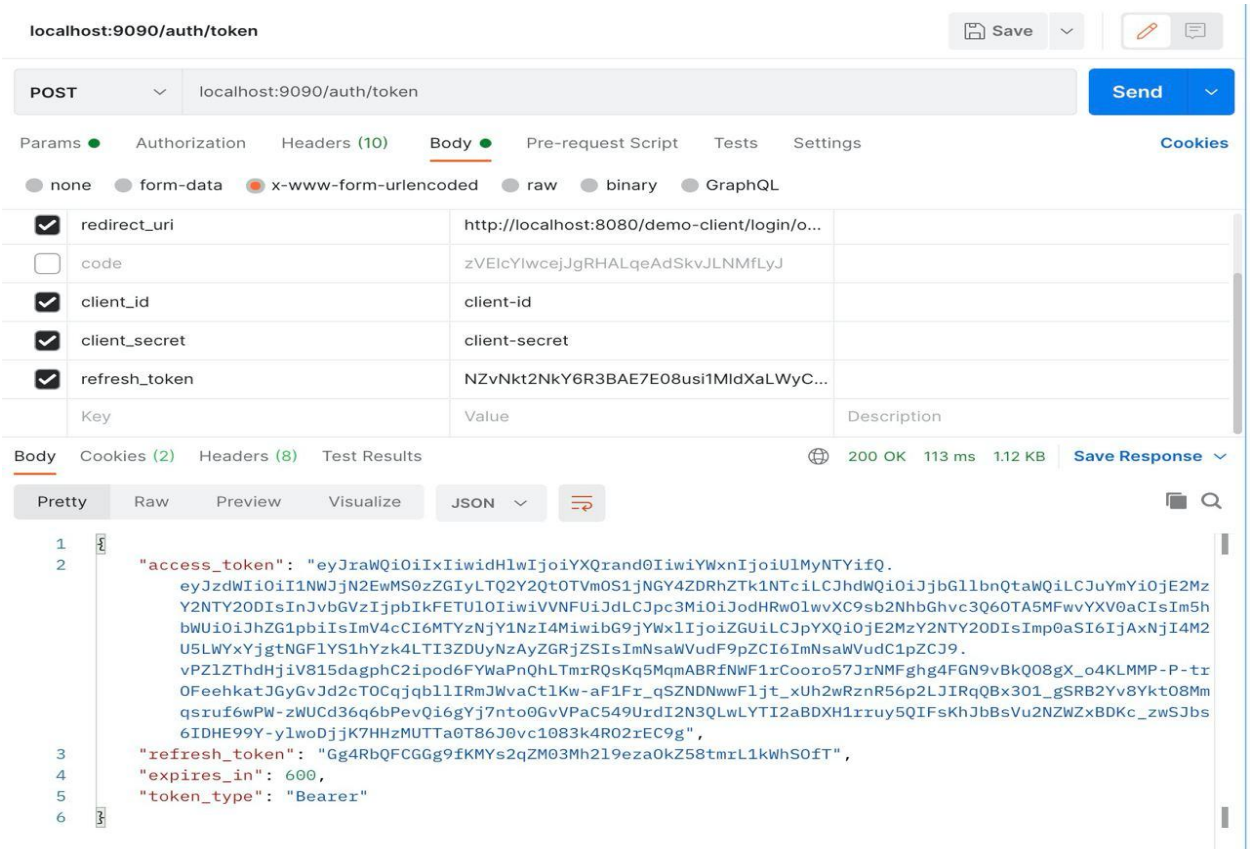

Рисунок 3.47 - Відновлення токена доступу

Тепер перевіримо дійсність нового токена доступу. Як бачимо з відповіді – токен дійсний (рисунок 3.48).

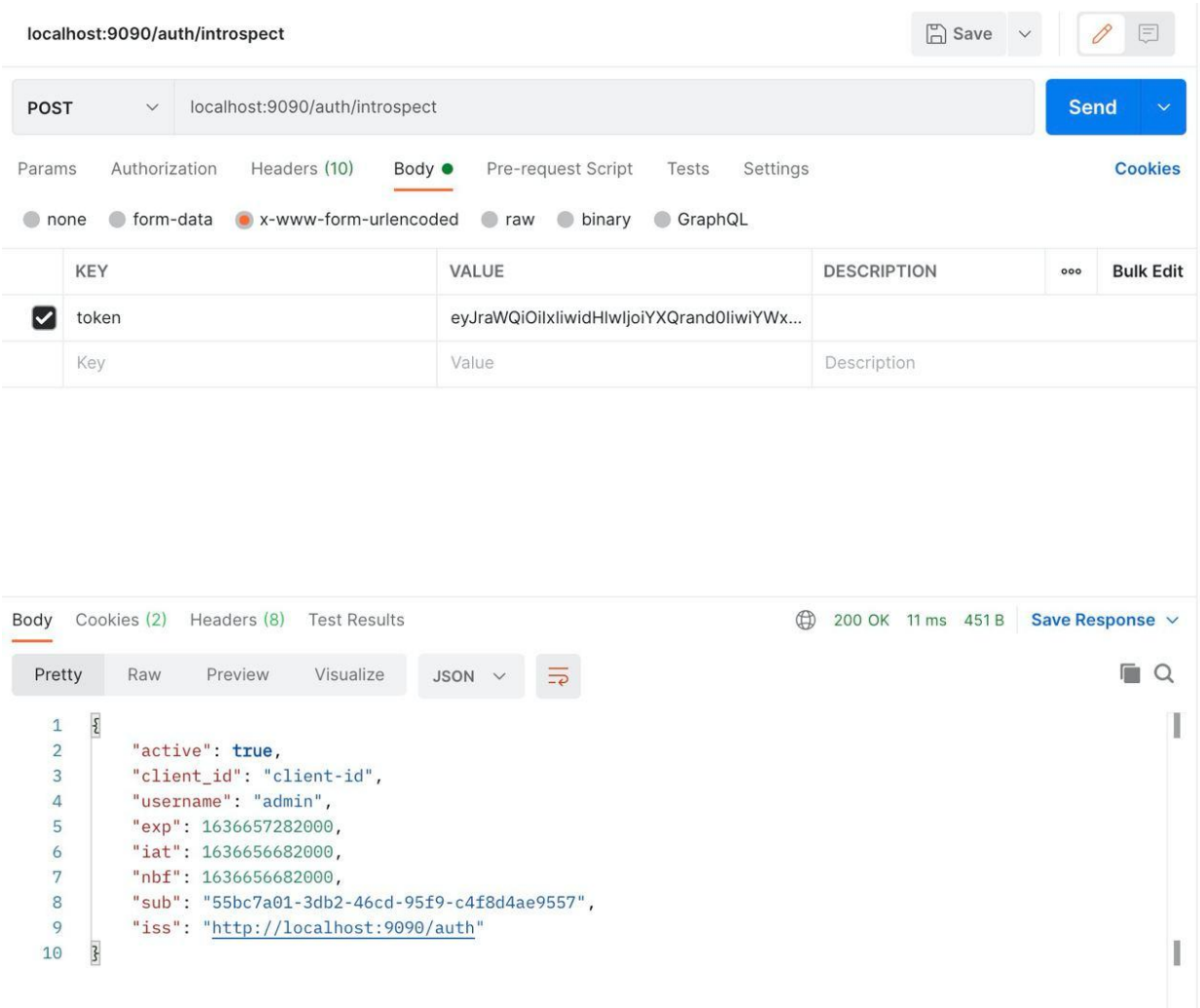

Рисунок 3.48 - Валідація нового токена доступу

ВИСНОВКИ

В ході написання кваліфікаційної роботи було розроблено програмне забезпечення сервісу автентифікації на основі протоколу OpenID. Результатом розробки є програма, яка автентифікує користувачів та надає їм токени за допомогою REST-запитів.

Метою роботи було розробити програмний продукт, який можна використовувати в цілях делегації автентифікації користувачів у клієнтах та можливість реалізувати SSO.

В ході кваліфікаційної роботи було проведено наступні роботи:

— було проаналізовано протоколів OAuth та OpenID Connect;

— було проаналізовано види автентифікації;

— було розглянуто та проаналізовано методи автентифікації;

— було проаналізовано технології та середовище виконання;

— було розроблено програмне забезпечення сервера автентифікації OpenID Connect;

— було проведено практичне тестування написаного додатка.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

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-vs-authorization-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

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 с. – (CreateSpace Independent Publishing Platform). – (ISBN 1979718474).

14) Siriwardena P. OpenID Connect in Action / Prabath Siriwardena., 2020. – 400 с. – (Manning). – (SBN 9781617298974).