

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

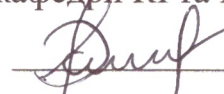
до випускної кваліфікаційної роботи бакалавра

на тему: «Проект підсистеми аутентифікації веб-додатку соціальної мережі
на основі технології єдиного входу»

Виконав студент 4-го курсу групи КБ-19-2
спеціальності 125 «Кібербезпека»

 Дмитро Москвін

Керівник старший викладач кафедри КІ та КБ

 Єлизавета Байлюк

Рецензент доцент кафедри КІ та КБ, к.т.н.

 Олександр Пірог

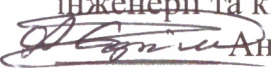
Житомир – 2023

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко

1 березня 2022 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Москвін Дмитро Володимирович

Тема роботи: «Проект підсистеми аутентифікації веб-додатку соціальної мережі на основі технології єдиного входу»

затверджена Наказом університету від 1 березня 2023 р. № ____

Термін здачі студентом закінченої роботи 19 червня 2023 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

- методи та технології аутентифікації.
- аутентифікація на основі технології єдиного входу.

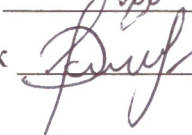
Обладнання та програмне забезпечення:

1. Персональний комп'ютер з операційною системою Windows 11
2. Інтегровані середовища розробки Microsoft Visual Studio та Visual Studio Code
3. Середовище SQL Server Management Studio
4. Сервіси Microsoft Azure, Azure DevOps

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

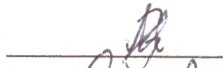
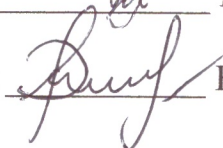
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Байлюк Є.М.	16.03.2023	16.03.2023
2	Байлюк Є.М.	14.04.2023	14.04.2023
3	Байлюк Є.М.	25.04.2023	25.04.2023

Студент  Москвін Д.В.

Керівник  Байлюк Є.М.

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	19.02.2023	Виконано
2	Аналіз літературних джерел	16.03.2023	Виконано
3	Аналіз об'єкта дослідження	25.03.2023	Виконано
4	Постановка задачі дослідження	03.04.2023	Виконано
5	Вибір технологій для реалізації	16.04.2023	Виконано
6	Проектування підсистеми аутентифікації	30.04.2023	Виконано
7	Проектування архітектури веб-додатку	05.05.2023	Виконано
8	Розробка веб-додатку	09.05.2023	Виконано
9	Запровадження підсистеми аутентифікації на базі розробленого додатку	11.05.2023	Виконано
10	Перевірка працездатності	12.05.2023	Виконано
11	Тестування, проведення атак та виявлення недоліків	26.05.2023	Виконано
12	Оформлення пояснювальної записки	08.06.2023	Виконано
13	Оформлення презентації	12.06.2023	Виконано
14	Отримання відгуку керівника та рецензії на кваліфікаційну роботу		Виконано
15	Захист		Виконано

Студент  Москвін Д.В.
Керівник  Байлюк Є.М.

АНОТАЦІЯ

Кваліфікаційна робота присвячена проектуванню та реалізації проекту підсистеми аутентифікації веб-додатку соціальної мережі на основі технології єдиного входу. Пояснювальна записка складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку. Текстова частина викладена на 72 сторінках друкованого тексту. Пояснювальна записка має 6 сторінок додатків. Список використаних джерел містить 25 найменувань і займає 2 сторінки. В роботі наведено 55 рисунків. Кваліфікаційна робота складається з трьох розділів.

В першому розділі було коротко розглянуто теоретичні аспекти підсистем аутентифікації та відповідних технологій, за допомогою яких вони реалізовані, технологію єдиного входу і протоколи OAuth 2.0 та OpenID Connect, описано поняття сеансу та особливості JWT маркерів.

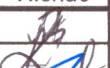
В другому розділі було розглянуто реалізації провайдерів аутентифікації від відомих корпорацій, розроблено критерії підбору та здійснено вибір. Спроектовано веб-додаток соціальної мережі з чотирирівневою архітектурою, що зробить проект чудовою основою для подальшого розвитку сервісу.

В третьому розділі здійснено необхідні налаштування провайдера аутентифікації. Реалізовано процес аутентифікації, оновлення та відливу токенів відповідно до рекомендацій та вимог документації. Розглянуто та проведено найпоширеніші атаки, які спрямовані на крадіжку конфіденційної інформації.

Висновки містять в собі результати виконаної роботи. У додатку наведено лістинг програмних модулів, які відповідають за базовий функціонал програми.

Ключові слова: ТЕХНОЛОГІЯ ЄДИНОГО ВХОДУ, ПРОВАЙДЕР ІДЕНТИФІКАЦІЇ, GOOGLE, OAUTH, OPENID CONNECT, СОЦІАЛЬНА МЕРЕЖА, ВЕБ-ДОДАТОК.

КІК.КР.Б – 125 – 23 – ПЗ

					КІК.КР.Б – 125 – 23 – ПЗ				
Зак.	Арк.	№ докум.	Підпис	Дата	Кваліфікаційна робота «Проект підсистеми аутентифікації веб-додатку соціальної мережі на основі технології єдиного входу»	Літ.	Арк.	Аркушів	
Розроб.		Москвін Д. В.		15.04.23					
Меріжник		Байлюк Є.М.		15.04.23			4	83	
Реценз.		Пірог О.В.		19.04.23		ФІКТ, гр. КБ-19-2			
Н. Контр.									
Затверд.		Єфіменко А. А.		16.05.23					

ANNOTATION

The qualification work is devoted to the design and implementation of the authentication subsystem project of the social network web application based on single sign-on technology. The explanatory note consists of a list of conventional abbreviations, an introduction, three sections, conclusions, a list of used literature and an appendix. The text part is laid out on 72 pages of printed text. The explanatory note has 6 pages of annexes. The list of used sources contains 25 items and occupies 2 pages. The work contains 55 pictures. The qualification work consists of three sections.

In the first chapter, the theoretical aspects of the authentication subsystems and the corresponding technologies with the help of which they are implemented and single sign-on technology were briefly considered. The OAuth 2.0 and OpenID Connect protocols are researched, the concept of a session and the features of JWT tokens are described.

In the second section, implementations of authentication providers from well-known corporations were considered, selection criteria were developed, and selection was made. A web application of a social network with a four-tier architecture has been designed, which will make the project an excellent basis for the further development of the service.

In the third section, the necessary settings of the authentication provider are made. The process of authentication, update and revocation of tokens has been implemented in accordance with the recommendations and requirements of the documentation. The most common attacks aimed at stealing confidential information are considered and carried out.

The conclusions contain the results of the work performed. The application lists the software modules that are responsible for the basic functionality of the application.

Keywords: SINGLE SIGN-ON, IDENTITY PROVIDER, GOOGLE, OAUTH, OPENID CONNECT, SOCIAL NETWORK, WEB APPLICATION.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	9
ВСТУП.....	10
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПІДСИСТЕМ АУТЕНТИФІКАЦІЇ	11
1.1. Поняття та засоби аутентифікації	11
1.1.1. Базова аутентифікація.....	11
1.1.2. Аутентифікація за допомогою Cookies	12
1.1.3. Багатофакторна автентифікація.....	12
1.1.4. Децентралізована аутентифікація.....	13
1.2. Технологія єдиного входу.....	14
1.2.1. Поняття єдиного входу	14
1.2.2. Поняття сеансу.....	18
1.3. Веб-токени JSON	19
1.4. Протокол авторизації OAuth 2.0	23
1.4.1. Призначення та принципи протоколу	23
1.4.2. Ролі OAuth 2.0.....	24
1.4.3. Принцип роботи OAuth 2.0	25
1.4.4. Типи грантів в OAuth 2.0	26
1.5. Протокол аутентифікації OpenID Connect	27
1.5.1. Опис протоколу	27
1.5.2. Неявний потік	28
1.5.3. Потік аутентифікації	29
1.5.4. Загальновідомий endpoint.....	30

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПІДСИСТЕМИ АУТЕНТИФІКАЦІЇ ВЕБ-ДОДАТКУ СОЦІАЛЬНОЇ МЕРЕЖІ.....	31
2.1. Вибір провайдера аутентифікації	31
2.1.1. Визначення критеріїв вибору та області застосування	31
2.1.2. Аутентифікація за допомогою Apple ID	32
2.1.3. Єдиний вхід в Azure Active Directory	35
2.1.4. Аутентифікація за допомогою Google	37
2.1.5. Порівняння провайдерів аутентифікації.....	38
2.2. Деталі процесу аутентифікації на основі технології єдиного входу	39
2.2.1. Аутентифікація користувача.....	39
2.2.2. Оновлення токенів.....	41
2.2.3. Відкликання токенів.....	42
2.2.4. Документ Discovery.....	43
2.2.5. Валідація ID токена.....	43
2.2.6. Механізм Proof-of-Proceession	45
2.3. Проектування архітектури веб-додатку	46
2.3.1. Чотирирівнева архітектура.....	47
2.3.2. Рівень представлення	48
2.3.3. Рівень обслуговування даних.....	50
2.3.4. Рівень бізнес-логіки	53
2.3.5. Рівень доступу до даних	54
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ АУТЕНТИФІКАЦІЇ НА ОСНОВІ ТЕХНОЛОГІЇ ЄДИНОГО ВХОДУ	56
3.1. Налаштування Google API Console.....	56

3.2. Аутентифікація користувачів, оновлення та відклик токенів	62
3.3. Перевірка працездатності підсистеми аутентифікації	67
3.4. Проведення атак та запровадження захисту.....	72
3.4.1. HTML Injection атака	72
3.4.2. JavaScript Injection атака	75
3.4.3. Уособлення облікового запису	77
ВИСНОВКИ.....	80
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	82
ДОДАТКИ.....	84
Додаток А	85

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SSO – Single Sign-On, Технологія єдиного входу

ПЗ – Програмне Забезпечення

IdP – Identity Provider, Постачальник ідентифікаційних даних

OIDC – OpenID Connect

SAML – Security Assertion Markup Language

JWT – JSON Web Token

API – Application Programming Interface

SPA – Single-Page Application

CSRF – Cross-Site Request Forgery

PL – Presentation Layer

DSL – Data Service Layer

BLL – Business Logic Layer

DAL – Data Access Layer

EF – Entity Framework

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

ВСТУП

Надійшла і правильно спроектована система аутентифікації є дуже важливою для соціальних мереж, адже її основна роль полягає в підтвердженні особи користувачів і захисті їх облікових записів. Будь-які вразливості можуть призвести до розголошення конфіденційних персональних даних або відкрити шлях до шахрайства та фальсифікації інформації. Наприклад, якщо надати змогу користувачеві створити кілька акаунтів у соціальній мережі без підтвердження своєї особи, він може використовувати дані облікові записи для поширення неправдивої або фейкової інформації.

Метою кваліфікаційної роботи є дослідження, проектування та реалізація підсистеми аутентифікації веб-додатку соціальної мережі на основі технології єдиного входу.

Завданням на кваліфікаційну роботу є:

- вивчення та аналіз методів та технологій аутентифікації;
- проектування підсистеми аутентифікації на основі технології єдиного входу та її реалізація у вигляді веб-додатку.

Предметом дослідження є методи та технології аутентифікації.

Об'єктом дослідження є аутентифікації на основі технології єдиного входу.

У процесі виконання кваліфікаційної роботи було спроектовано та реалізовано проект підсистеми аутентифікації на основі технології єдиного входу, яка покликана бути основою веб-додатку соціальної мережі. Дана підсистема містить необхідний інструментарій для контролю доступу користувачів до конфіденційних даних, обмеження доступу до відповідних сторінок, гарантує високий рівень безпеки та надає можливість подальшого розвитку функціоналу соціальної мережі.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПІДСИСТЕМ АУТЕНТИФІКАЦІЇ

1.1. Поняття та засоби аутентифікації

Аутентифікація[11] – це процес перевірки особи користувача для забезпечення захисту конфіденційної інформації, особливо в онлайн-сервісах, таких як банківські системи, електронні пошти, соціальні мережі та торгівельні сервіси.

Відсутність аутентифікації надасть будь-якому користувачу можливість надсилати, отримувати, змінювати або видаляти інформацію на сервісі без обмежень. Це може призвести до несанкціонованого доступу до конфіденційних даних та інформації, що може завдати шкоди окремим користувачам і навіть організаціям.

Процес аутентифікації засновується на введенні облікових даних, таких як імена користувачів і паролі, тобто конфіденційної інформації, відомої лише користувачеві, або використання біометричних даних, таких як відбитки пальців чи розпізнавання обличчя.

1.1.1. Базова аутентифікація

Базова аутентифікація представляє собою найпростіший метод обмеження доступу до веб-ресурсів. Основною перевагою цього методу є простота реалізації та використання, але даний метод має велику кількість недоліків.

Механізм аутентифікації запускається під час запиту браузером захищеного документа, при цьому не надаючи жодних ідентифікаційних даних. На такий запит сервер повертає відповідь із заголовком «401 Unauthorized» та пропонує здійснити ідентифікацію. Далі браузер показує вікно, в якому користувачу пропонується вказати своє ім'я і пароль. Після підтвердження вводу відправляється наступний запит до сервера, який вже містить рядок аутентифікації.

Варто зазначити, що персональні дані не відправляються в відкритому вигляді, вони кодуються за технологією Base64. Такі дані не читабельні та на перший погляд може здаватися, що такий метод є досить безпечним, але для декодування такого рядка достатньо лише використати відповідну функцію, яка зазвичай вбудована у більшість мов програмування.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

Також у даному механізмі існує цілий ряд особливостей, таких як, наприклад, кешування даних браузером, що дає можливість тимчасово зберігати дані та не запитувати кожного разу повторного введення користувачем ідентифікаційних даних, але таке «полегшення» може призвести до перехоплення персональних даних.

1.1.2. Аутентифікація за допомогою Cookies

Величезна кількість різноманітних онлайн ресурсів та сервісів, серед яких месенджери, форуми, ігри та інші, базуються на механізмі аутентифікації на основі Cookies, що представляє собою наступний принцип:

- користувач надає свій логін та пароль у відповідні текстові поля на сторінці входу та підтверджує їх відправку на сервер;
- сервер отримує введені ідентифікаційні дані, перевіряє їх і, у випадку їх коректності та відповідності, повертає на клієнт Cookies, що містить деякий ідентифікатор сесії. Такі Cookies можуть як досить тривало зберігатись, так і обмежувати свій термін дії лише поточною сесією браузера;
- усі наступні запити користувача на сервер будуть містити Cookies з ідентифікатором сесії, який сервер перевіряє у своїй базі і, за його наявності, «впізнає» користувача та надає доступ до відповідних дій або даних.

Даний підхід є більш захищеним ніж базовий, адже запити не містять у собі персональні дані, але у разі, коли дані, що вводяться, погано фільтруються або не фільтруються зовсім, викрасти Cookies стає досить просто. Внаслідок крадіжки та підробки Cookies, зловмисник може автентифікуватись замість іншого користувача. Одним з методів додаткового захисту є встановлення зв'язку Cookies сесії з IP-адресою, з якого користувач авторизувався в системі. Проте IP-адресу теж можна підробити використовуючи IP-спуфінг, тому дана методика все одно не може забезпечити абсолютний захист.

1.1.3. Багатофакторна автентифікація

Для підвищення безпеки варто використовувати одночасно декілька методів аутентифікації, проте потрібно розуміти дуже важливу деталь: не кожна комбінація

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

декількох методів є багатофакторною аутентифікацією, обов'язкове використання чинників різного походження[1]:

- властивість, яка притаманна самому суб'єкту, наприклад, відбитки пальців, райдужна оболонка ока, капілярні візерунки, послідовність ДНК;
- інформація, яку знає лише суб'єкт, наприклад, пароль (кодове слово) або пін-код;
- річ, яку має суб'єкт, наприклад, електронна або магнітна картка, флеш-накопичувач.

Один з найбільш надійних на сьогоднішній день методів багатофакторної автентифікації заснований на використанні токенів, тобто персональних апаратних пристроїв, які дозволяють генерувати і зберігати ключі шифрування, забезпечуючи тим самим строгу аутентифікацію.

Використання класичних «багаторазових» паролів є серйозною уразливістю у випадку взаємодії з чужими комп'ютерами та дана проблема підштовхнула виробників світового ринку аутентифікації до створення апаратних генераторів одноразових паролів, які виконують генерацію нового одноразового значення за розкладом (наприклад, кожну хвилину), або за запитом, тобто при натисканні на кнопку. Далі на стороні сервера здійснюється спеціальна перевірка правильності введеного значення шляхом програмного обчислення поточного значення одноразового пароля. Після чого користувач вводить постійний пароль, що є частиною принципу багатофакторною аутентифікації.

1.1.4. Децентралізована аутентифікація

Можна виділити наступні види децентралізованої аутентифікації:

- OpenID – відкрита система аутентифікації користувачів, завдяки якій користувач може мати один логін-пароль для різних веб-ресурсів. Передача даних зазвичай здійснюється через HTTPS, але існує альтернативний варіант, що полягає у підписі повідомлень та передачі ключа для цифрового підпису за допомогою алгоритму Діффі-Хеллмана. Серед вразливостей OpenID можна виділити схильність до фішингових атак, тобто перенаправлення зловмисником користувача на

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

підроблений сайт провайдера аутентифікації, який буде запрошувати користувача ввести його логін і пароль. Також даний метод є вразливим до атаки «людина посередині». Да даний момент OpenID активно використовується і надається такими гігантами, як Google, IBM, Microsoft, PayPal, Yahoo та інші;

- OAuth – дозволяє одному інтернет-сервісу отримати доступ до даних користувача на іншому інтернет-сервісі. Протокол використовується в таких системах, як Twitter, Google та інші;

- OpenID Connect[2] – певною мірою гібридний протокол, що об'єднує в собі OpenID і OAuth.

Одним з головних мінусів таких систем є те, що злом дає доступ відразу до багатьох сервісів.

1.2. Технологія єдиного входу

1.2.1. Поняття єдиного входу

Технологія єдиного входу (Single Sign-On або SSO) – метод аутентифікації, завдяки якому користувачі можуть безпечно аутентифікуватися одразу в декількох сервісах, додатках та сайтах, використовуючи один і той самий набір облікових даних[7].

Єдиний вхід (SSO) полягає у вході в одну програму, а всі подальші входи в інші сервіси виконуються автоматично, незалежно від того, які платформи, технології чи домену використовує користувач[12].

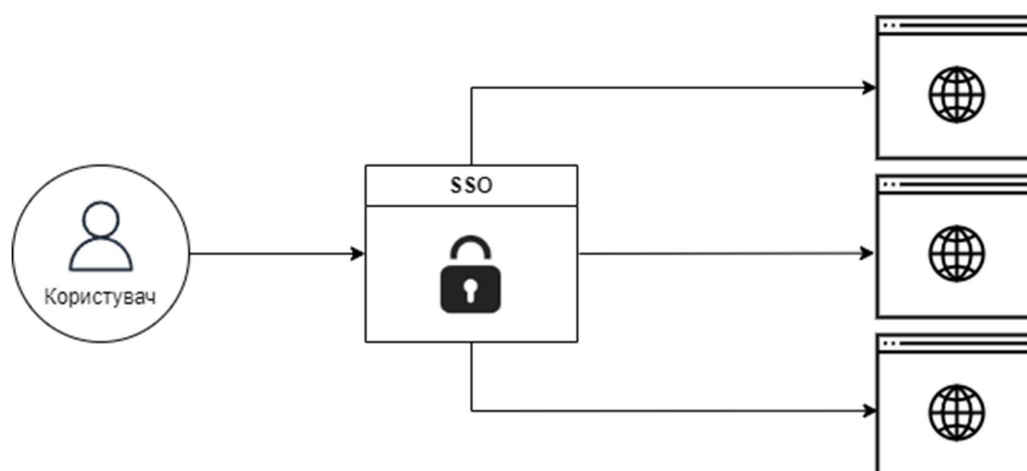


Рисунок 1.1 – Візуалізація принципу роботи технології єдиного входу

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

Наприклад, під час входу в такий сервіс Google, як Meet, аутентифікація в Gmail, YouTube, Keep та інших додатках Google буде пройдена автоматично. Аналогічно, під час виходу з одного з вищевказаних сервісів, буде одночасно виконано вихід з усіх інших, що ще відомо як одноразовий вихід.

Single Sign-On ґрунтується на налаштуванні довірчих відносин між програмою, тобто identity provider (провайдер послуг), та системою управління доступами. Такі довірчі відносини часто мають за основу обмін сертифікатом між системою керування доступом та провайдером послуг. Призначення даного сертифікату полягає в позначенні ідентифікаційної інформації, яку відсилає система керування доступами провайдеру послуг. Такий підхід може гарантувати провайдеру послуг, що інформація надійшла із надійного джерела. У SSO ідентифікаційні дані представлені у форми токенів, що містять значення інформації про користувача, такі як email або ім'я користувача.

Порядок аутентифікації зазвичай виглядає наступним чином:

1. Користувач заходить у додаток або на сайт, доступ до якого він хоче отримати, тобто до провайдера послуг.
2. Провайдер послуг відправляє запит на автентифікацію користувача, серед якої є токен, у якому міститься інформація про користувача (наприклад, email адреса) системі управління доступами (SSO).
3. Система управління доступами перевіряє виконує перевірку на наявність актуальної сесії користувача. Якщо так, немає потреби здійснювати повторний вхід в обліковий запис, користувачеві надається доступ до провайдера послуг, відразу приступаючи до кроку 5.
4. Користувач надає ідентифікаційні дані, які вимагає система керування доступами для аутентифікації. Це може бути як звичайний логін і пароль, так і багатофакторна аутентифікація.
5. Система управління доступами повертає токен у разі схвалення наданих ідентифікаційних даних, тим самим підтверджуючи успішну аутентифікацію.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

6. Повернений токен повертається у браузер користувача, а звідти на провайдер послуг.

7. Токен, отриманий провайдером послуг, підтверджується згідно з довірчими відносинами, встановленими між провайдером послуг та системою керування доступами під час початкового налаштування.

8. Користувачеві надається доступ до провайдера послуг.



Рисунок 1.2 – Візуалізація процесу аутентифікації

Якщо користувач намагається отримати доступ до іншого сайту, аналогічні довірчі відносини повинні бути встановлені відповідно до методу SSO. У цьому випадку процедура аутентифікації також складатиметься з описаних вище кроків.

Існує кілька причин для впровадження SSO. Методи єдиного входу можуть спростити управління логінами і паролями як для користувачів, так і для адміністраторів. Користувачам більше не потрібно знати всі свої облікові дані, а потрібно запам'ятати лише складніший пароль; SSO дозволяє користувачам отримувати доступ до додатків швидше.

SSO також скорочує час, який служба підтримки витрачає на відновлення паролів. Адміністратори можуть централізовано контролювати такі фактори, як складність пароля та багатофакторна автентифікація (MFA). Адміністратори також можуть швидше відкликати права на вхід, якщо користувач залишає організацію.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

Однак SSO має деякі недоліки. Наприклад, якщо знадобитися заблокувати доступ лише до певних додатків. Тому важливо вибирати рішення SSO, яке може, наприклад, вимагати додаткових факторів аутентифікації перед входом користувача в систему або забороняти доступ до певних додатків, поки не буде встановлено безпечне з'єднання.

Важливо розуміти різницю між SSO та сховищем або менеджером паролів, який іноді плутають з SSO, але в контексті Same Sign-On означає «однаковий/тотожний вхід», а не «єдиний вхід». Коли йдеться про зберігання паролів, користувачі можуть мати ім'я користувача та пароль, але їм потрібно вводити їх щоразу, коли вони переходять до нового додатку або веб-сайту. У такій системі зберігаються ідентифікаційні різні додатків і вводяться за потреби. У цьому випадку між додатком і сховищем паролів немає довірчих відносин.

SSO дозволяє користувачам отримувати доступ до всіх схвалених компанією сайтів і додатків увійшовши в систему один раз, без потреби здійснювати повторну аутентифікації. Це стосується як хмарних, так і локально встановлених додатків, доступ до яких зазвичай здійснюється через саму службу SSO (також відому як служба аутентифікації).

SSO є частиною ширшої концепції під назвою Federated Identity Management, і тому SSO іноді називають Federated SSO; FIM просто відносить до довірчих відносин, встановлених між двома або більше доменами або системами управління ідентифікацією. Єдиний вхід (SSO) – це функція/функції, доступні в архітектурі FIM.

OAuth 2.0 – це спеціальна програмна платформа, яка є частиною архітектури FIM, яка ґрунтується на довірчих відносинах, надаючи доменам ідентифікаційну інформацію користувача.

OpenID Connect (OIDC) – це рівень аутентифікації, накладений на базу OAuth 2.0, щоб забезпечити функціональність SSO[2].

Security Access Markup Language (SAML) – це відкритий стандарт, розроблений для забезпечення функціональності SSO. Даний стандарт вважається застарілим та поступово замінюється на користь OIDC.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

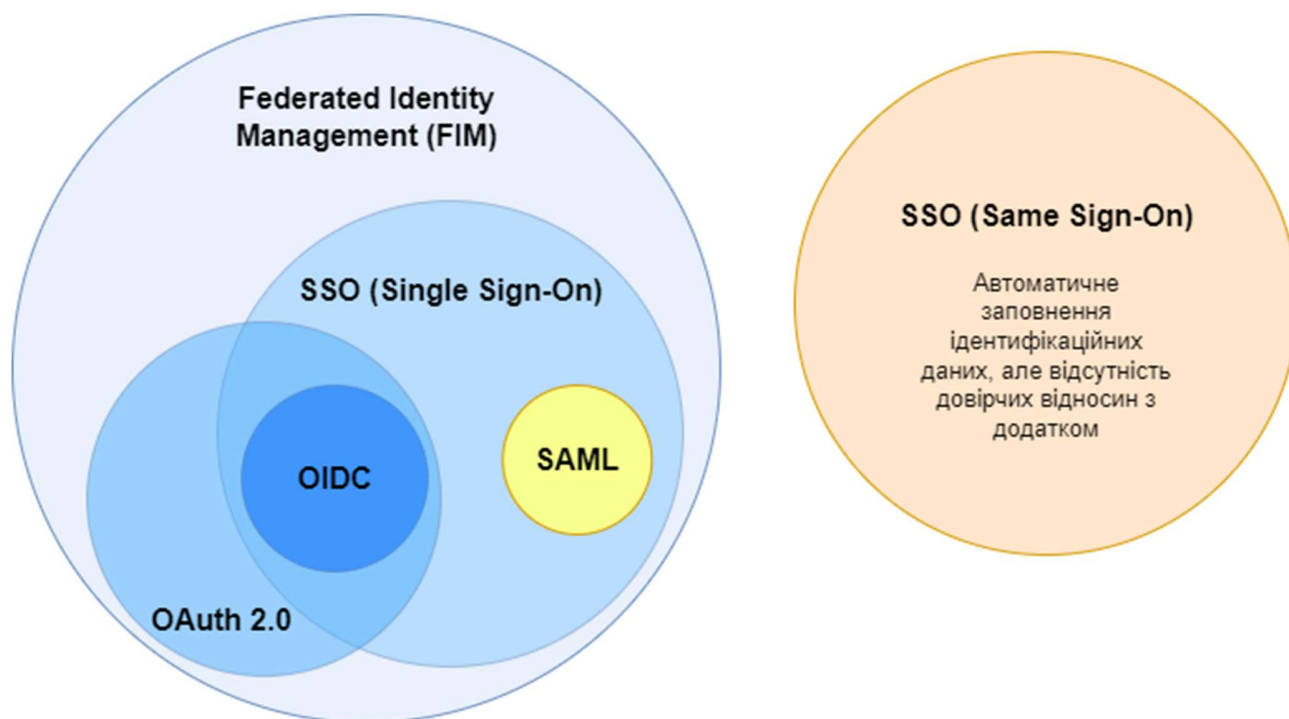


Рисунок 1.3 – Візуалізація зв'язків між вищеописаними поняттями

Система Same Sign-On, також відома як SSO, насправді, зовсім не схожа на Single Sign-on, адже вона не передбачає встановлення довірчих відносин між аутентифікованими сторонами[13]. Такий підхід ґрунтується на дублюванні та передачі ідентифікаційних даних в інші системи, коли це необхідно. Даний метод буде гіршим в плані безпеки за будь-яке рішення єдиного входу.

1.2.2. Поняття сеансу

Сеанс (сесія) – це серія взаємодій між користувачем і додатком, які відбуваються протягом певного періоду часу. Одна сесія може містити кілька дій (наприклад, перегляд сторінок, події, соціальні взаємодії та транзакції електронної комерції), які будуть тимчасово зберігатися, поки користувач підключений.

За замовчуванням сесія завершується, коли користувач залишає веб-сайт або закриває браузер. Щоб користувачеві не доводилося входити в систему щоразу, коли він повертається, додаток може зберігати інформацію про сеанс у файлі cookie або Local Storage, щоб потім відновити сесію. Сесія закінчується коли користувач виходить з системи або коли досягається ліміт часу. Сеанси також зазвичай завершуються, коли користувач скидає пароль, електронну пошту або номер телефону.

SAML ENCODED

PASTE A TOKEN HERE

SAML DECODED

☒ Prettify (not editable)

Expand

PHNhbWxwOjIic3BvbmlhHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wleIEPSJfNjlxYzRjNGFhWQ2MGM3NjhjYzllCBWZXXJzaW9uPSlyLjAilElzc3VlSW5zdGFudD0IMjAxNC0xMCOxNFQxND0zMjoxN1oicEZXN0aW5hdGlvb0iaHR0cHM6Ly9hcHAuYXV0aDAuY29tL3Ric3Ric9ZyYw1scCI+PHNhbWw6SXNzdWVyiHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lc2p0YzptQU1MOjluMDphc3NlcnRpb24iPnVybjptYXR1Z2l0LmF1dGgwLmNvbTwwc2FtbDpJc3N1ZXI+PHNhbWxwOIN0YXR1cz48c2FtbHA6U3RhdHVzQ29kZSBWYXw1ZT0idXJuOm9hc2l2Om5hbWVzOnRjOINBTUw6M14wOnN0YXR1czpTdWNjZXNzli8+PC9zYW1scDpTdGF0dXM+PHNhbWw6QXNzZXJ0aW9uIiHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lc2p0YzptQU1MOjluMDphc3NlcnRpb24iIlFZlcnNpb249IjluMCIgSUQ9Ii81Vks3TFQ3RmxpVWtrYVYVf1vZyZyNGJyRjBERzVFM1g3NilgSXNzdWVJbnN0YV50PSlyMDU0LEwLTE0VDE0OjMyOjE3LjllMVoiPjxzYW1sOkkzc3Vlcj51cm46bWF0dWdpdC5hdXRoMCS5b208L3NhbWw6SXNzdWVYpJxTaWduYXR1cmUgeG1sbmM9Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvMDkveG1sZHNpZyMiPjxTaWduZWZWRjbmZvPjxYV5vbmliYXVxpmF0aW9uTWV0aG9klEFsZ29yaXRobT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8xMCS94bWwtZXhlLWMxNG4jli8+PFNpZ25hdHvyZU1ldGhvZCBbbGdvcmliOaG09Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvMDkveG1sZHNpZyNyc2Etc2hhMSlvpjxSZWZlcmVuY2UgVGVJPSlJxZW5zdMVDdG5GVA2thUXVXN0YnJGMERHNUUzWdC2lJ48VHJhbnNmb3Jtcz48VHJhbnNmb3JtIi

1 <saml:Response
2 xmlns:saml="urn:oasis:names:tc:SAML:2.0:protocol" ID="_621c4
3 <saml:Issuer
4 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:matu
5 </saml:Issuer>
6 <saml:Status>
7 <saml:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status
8 </saml:Status>
9 <saml:Assertion
10 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="

11 <saml:Issuer>urn:matugit.auth0.com</saml:Issuer>
12 <Signature
13 xmlns="http://www.w3.org/2000/09/xmldsig#">
14 <SignedInfo>
15 <CanonicalizationMethod Algorithm="http://www.w3.org/2

16 <SignatureMethod Algorithm="http://www.w3.org/2000/0

17 <Reference URI="#_5VK7LT7FliUkkaQuW6r4brFODG5E3)

18 <Transforms>
19 <Transform Algorithm="http://www.w3.org/2000/09/x

20 <Transform Algorithm="http://www.w3.org/2001/10/xr

21 </Transforms>
22 <DigestMethod Algorithm="http://www.w3.org/2000/09

23 <DigestValue>ZDkfGO3H1Tu50hawzQVjsACzJwc=</Dig

24 </Reference>
25 </SignedInfo>
26 <SignatureValue>1Fgpt7AaHcME2gTA158achvGQVqDwHSH

Рисунок 1.5 – Маркер SAML в зашифрованому та дешифрованому стані

- безпека: JWT може використовувати для підпису пару відкритих/приватних ключів у формі сертифіката X.509, або також може бути симетрично підписаний спільним ключем за допомогою алгоритму HMAC. І хоча токени SAML можуть використовувати пари відкритих/приватних ключів, підписати XML за допомогою цифрового підпису без введення незрозумілих прогалин у безпеці дуже складно порівняти з простотою підпису JSON;

- поширеність: аналізатори JSON поширені в більшості мов програмування, оскільки вони перетворюється безпосередньо на об'єкти. І навпаки, XML не має природного форматування документа в об'єкт. Це полегшує роботу з JWT в порівнянні з SAML;

- обробка: JWT використовується в масштабі Інтернету, тобто його легше обробляти на пристроях користувача, особливо мобільних.

JWT використовують у наступних цілях:

- аутентифікація: під час успішного входу користувача за допомогою своїх облікових даних повертається маркер ідентифікатора;

- авторизація: після успішного входу користувача у систему програма може запросити доступ до окремих сторінок, послуг або ресурсів (наприклад, API) від імені цього користувача. Для цього в кожному запиті повинен бути переданий маркер доступу, який може бути у формі JWT. Система єдиного входу (SSO) широко використовує JWT через невеликі накладні витрати на формат і його можливість легко використовувати в різних доменах;
- обмін інформацією: JWT є хорошим способом безпечної передачі інформації між сторонами, оскільки такі маркери можуть бути підписані, а отже дає гарантію, що відправники є тими, за кого себе видають. Крім того, структура JWT дозволяє перевірити, чи не було підроблено його вміст.

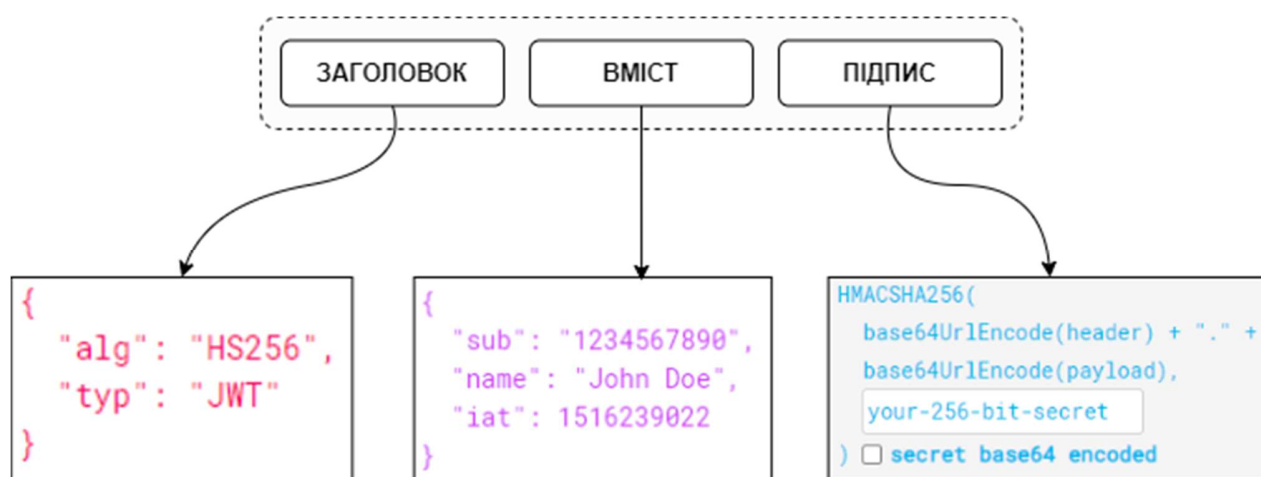


Рисунок 1.6 – Структура JWT токена

Інформація, що міститься в об'єкті JSON, може бути перевірена та довірена, оскільки вона має цифровий підпис. Хоча JWT також можна зашифрувати для забезпечення секретності між сторонами, але маркери, видані провайдерами єдиного входу, є веб-підписами JSON (JWS), тобто вони підписані, а не зашифровані. Таким чином, ми зосередимося на підписаних токенах, які можуть перевірити цілісність клеймів (полів з відповідними значеннями), що містяться в них, тоді як зашифровані токени приховують ці клейми від інших сторін.

Загалом JWT можна підписувати за допомогою секрету (з алгоритмом HMAC) або пари відкритих/закритих ключів за допомогою RSA чи ECDSA (вибір алгоритму залежить від використаного протоколу аутентифікації, наприклад, Auth0 підтримує

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

лише HMAC і RSA). Коли токени підписуються за допомогою пар відкритих/ закритих ключів, підпис також засвідчує, що лише сторона, яка володіє закритим ключем, є тією, яка його підписала.

Перш ніж використовувати отриманий JWT, він повинен бути перевірений за допомогою свого підпису. Варто зауважити, що успішно перевірений маркер означає лише те, що інформація, яка міститься в маркері, не була змінена ніким іншим. Це не означає, що інші не змогли побачити вміст, який зберігається у вигляді звичайного тексту. Через це ніколи не варто зберігати конфіденційну інформацію всередині JWT і необхідно використовувати інші шляхи, щоб гарантувати, що JWT не перехоплюються, наприклад, надсилаючи JWT лише через HTTPS, використовуючи наступні практики та рекомендації:

- ключ підпису слід розглядати як будь-які інші облікові дані та відкривати лише тим службам, яким він потрібен;
- токени підписані для захисту від маніпуляцій і легко декодуються. Варто додавати мінімальну кількість клеймів до вмісту маркеру для найкращої продуктивності та безпеки;
- технічно, коли маркер підписаний, він дійсний назавжди, якщо ключ підпису не змінено або термін дії не встановлено явно. Це може створити потенційні проблеми, тому варто запровадити стратегію для закінчення терміну дії та/або відкликання токенів;
- не варто надсилати маркери через з'єднання без HTTPS, оскільки ці запити можуть бути перехоплені, а маркери скомпрометовані;
- рекомендується зменшити непотрібні зворотні переходи, які розширюють поверхню атаки програми та оптимізувати обмеження маркерів плану (де це можливо), зберігаючи маркери доступу, отримані з сервера авторизації. Замість того, щоб запитувати новий маркер, краще використовувати збережений маркер під час майбутніх викликів, доки термін його дії не закінчиться. Спосіб зберігання маркерів залежатиме від характеристик програми: типові рішення включають бази даних (для

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

програм, яким потрібно виконувати виклики API незалежно від наявності сеансу) і сеанси HTTP (для програм, які мають вікно активності, обмежене інтерактивним сеансом).

JWT токени є альтернативою файлів cookie. Як правило, односторінкові програми, мобільні додатки iOS та Android і веб-інтерфейси API краще працюють з аутентифікацією на основі маркерів. Традиційні веб-програми на стороні сервера традиційно використовують аутентифікацію на основі файлів cookie.

Аутентифікація на основі маркерів реалізується шляхом генерації маркера під час аутентифікації користувача та встановлення цього маркера в заголовок «Authorization» кожного наступного запиту до API. Дуже бажано, щоб цей токен був чимось стандартним, наприклад веб-токени JSON, оскільки в такому випадку не потрібно реалізовувати створювати власну криптографію.

За допомогою обох підходів можливо отримати однаковий обсяг інформації від користувача. Це контролюється параметром «scope», надісланим у запиті на вхід.

Даний параметр задає клейми, які повернуться від провайдера аутентифікації та будуть необхідні для подальшої роботи, наприклад, «openid name email family_name address phone_number» (у форматі стрічки з пробілом в якості роздільника).

Також можна комбінувати аутентифікацію на основі маркера з аутентифікацією на основі файлів cookie. Але варто зауважити, що файли cookie працюватимуть нормально тільки якщо веб-додаток та API обслуговуються з одного домену.

1.4. Протокол авторизації OAuth 2.0

1.4.1. Призначення та принципи протоколу

OAuth 2.0, що розшифровується як «відкрита авторизація», – це стандарт, який дозволяє веб-сайту або програмі отримувати доступ до ресурсів, розміщених іншими веб-додатками, від імені відповідного користувача[15]. У 2012 році він замінив OAuth 1.0 і зараз є фактичним галузевим стандартом для онлайн-авторизації. OAuth 2.0 надає доступ за згодою та обмежує дії, які клієнтська програма може виконувати з ресурсами від імені користувача, ніколи не передаючи облікові дані користувача.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Хоча Інтернет є основною платформою для OAuth 2.0, специфікація також описує, як обробляти цей вид делегованого доступу до інших типів клієнтів (програми на базі веб-переглядача, серверні веб-програми, мобільні додатки, підключені пристрої тощо).

OAuth 2.0 є протоколом авторизації та не можна його плутати з протоколом аутентифікації. Він розроблений, перш за все, як засіб надання доступу до набору ресурсів, наприклад, віддалених API або даних користувачів.

OAuth 2.0 використовує маркери доступу, тобто фрагменти даних, які надають дозвіл на доступ до ресурсів від імені кінцевого користувача. OAuth 2.0 не визначає конкретний формат для маркерів доступу, однак частіше всього використовується формат JSON Web Token (JWT), що дозволяє токенам містити відповідні дані. Крім того, маркери доступу зазвичай мають термін дії для покращення безпеки.

1.4.2. Ролі OAuth 2.0

Ідея ролей є частиною основної специфікації структури авторизації OAuth2.0. Вони визначають основні компоненти даної системи, а саме[25]:

- власник ресурсу: користувач або система, яка володіє захищеними ресурсами та може надати до них доступ;
- клієнт: система, якій потрібен доступ до захищених ресурсів. Для доступу до ресурсів клієнт повинен мати відповідний маркер доступу;
- сервер авторизації: цей сервер отримує запити від клієнта на маркери доступу та видає їх після успішної аутентифікації та згоди власника ресурсу. Сервер авторизації надає дві кінцеві точки: Authorization endpoint, яка обробляє інтерактивну аутентифікацію та згоду користувача, і Token endpoint, яка бере участь у взаємодії між сервісами;
- сервер ресурсів: сервер, який захищає ресурси користувача та отримує запити на доступ від клієнта. Він приймає та перевіряє маркер доступу та повертає на клієнт відповідні ресурси.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4.3. Принцип роботи OAuth 2.0

Перед початку використання OAuth 2.0 клієнт повинен отримати власні облікові дані, а саме ідентифікатор та секрет клієнта від сервера авторизації, щоб ідентифікувати та аутентифікувати себе під час запиту маркера доступу.

За допомогою OAuth 2.0 запити на доступ ініціюються клієнтом, наприклад, мобільною програмою, веб-додатком, програмою для Smart-TV, десктопною програмою. Запит маркера, обмін і відповідь здійснюють у наступному загальному порядку:

- клієнт запитує авторизацію від сервера авторизації, надаючи ідентифікатор клієнта та секрет; він також надає області та URI кінцевої точки (redirect URI), куди потрібно повернути маркер доступу або код авторизації;
- сервер авторизації аутентифікує клієнта та перевіряє, чи дозволені запитані області;
- власник ресурсу взаємодіє з сервером авторизації, щоб надати доступ;
- сервер авторизації перенаправляє назад на клієнт або код авторизації, або маркер доступу, залежно від типу дозволу. За потреби сервер також може повернути маркер оновлення;
- за допомогою маркера доступу клієнт запитує доступ до ресурсу у сервера ресурсів.

Сервер авторизації OAuth 2.0 може не повертати безпосередньо маркер доступу після того, як власник ресурсу авторизувався. Для кращої безпеки можна повернути код авторизації, який потім обмінюється на маркер доступу. Крім того, сервер авторизації може також видавати маркер оновлення з маркером доступу. На відміну від токенів доступу, токени оновлення зазвичай мають необмежений термін дії і за їх допомогою можна отримати нові токени доступу, коли сплине їх термін дії. Оскільки токени оновлення мають такі властивості, варто зберігати їх безпечно.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4.4. Типи грантів в OAuth 2.0

У OAuth 2.0 грант – це набір кроків, які клієнт має виконати, щоб отримати авторизацію доступу до ресурсу. Структура авторизації надає кілька типів грантів для вирішення різних сценаріїв[16]:

- грант коду авторизації: сервер авторизації повертає клієнту одноразовий код авторизації, який потім обмінюється на маркер доступу. Це найкращий варіант для традиційних веб-програм, де обмін може безпечно відбуватися на стороні сервера. Потік коду авторизації може використовуватися односторінковими програмами (SPA) і мобільними/нативними програмами. Однак тут секрет клієнта не може бути надійно збережений, тому аутентифікація під час обміну обмежена використанням лише ідентифікатора клієнта;

- неявний грант: спрощений потік, у якому маркер доступу повертається безпосередньо клієнту. У неявному потоці сервер авторизації може повернути маркер доступу як параметр в URI зворотного виклику або як відповідь на повідомлення форми. Перший варіант зараз не підтримується через можливий витік токенів;

- грант коду авторизації з ключем підтвердження для обміну кодами (PKCE): цей процес авторизації подібний до гранту коду авторизації, але з додатковими кроками, які роблять його більш безпечним для мобільних/нативних програм і SPA;

- тип гранту облікових даних власника ресурсу: цей грант вимагає, щоб клієнт спочатку отримав облікові дані власника ресурсу, які передаються на сервер авторизації, тому він обмежений клієнтами, яким можна повністю довіряти. Його перевага полягає в тому, що перенаправлення на сервер авторизації не використовується, тому його можна застосовувати у випадках використання, коли перенаправлення неможливо;

- тип гранту облікових даних клієнта: використовується для неінтерактивних програм, наприклад, автоматизованих процесів, мікросервісів тощо. У цьому випадку програма аутентифікується за допомогою свого ідентифікатора клієнта та секрету;

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

- потік авторизації пристрою: грант, який дозволяє використовувати додатки на пристроях з обмеженим введенням, наприклад на смарт-телевізорах;
- грант токена оновлення: потік, який передбачає обмін токена оновлення на новий токен доступу.

1.5. Протокол аутентифікації OpenID Connect

1.5.1. Опис протоколу

OpenID Connect (OIDC) – це безпечний механізм, який дозволяє застосунок зв'язатися зі службою ідентифікації, щоб отримати необхідні дані про користувача та повернути їх назад у програму, забезпечивши повний захист даних[17].

За своєю суттю, OpenID Connect починав свій шлях як механізм, що надає користувачеві можливість створити єдиний обліковий запис для аутентифікації на безлічі не пов'язаних один з одним інтернет-ресурсів, тобто в його основі лежить OAuth. Саме тому в ньому є багато схожих деталей з OAuth таких як ідентифікатор, секрет клієнта і URI перенаправлення, які зберігаються в системі керування доступами. Усі вони призначені для забезпечення передачі даних від конкретної людини конкретному додатку. Також наявні scopes, або області видимості. Для OIDC існує особлива область видимості openid, яку необхідно використовувати в якості першого звернення до системи керування доступами.

Порядок роботи з OpenID Connect багато в чому залежить від типу клієнта. Наприклад, якщо це JavaScript-додаток, де все може переглядатися будь-ким, хто використовує інструменти розробника в браузері, а також відсутній бекенд-механізм, варто реалізувати неявний потік для OpenID Connect.

Якщо ж це більш традиційний додаток, де інформація передається на фронтенд (і є ймовірність, що хтось може її перехопити), але в той же час використовується бекенд, щоб передавати інформацію службі керування доступами абсолютно конфіденційно, краще реалізовувати потік аутентифікації[11].

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

1.5.2. Неявний потік

Даний потік розроблений для передачі основної інформації про користувача додатку.

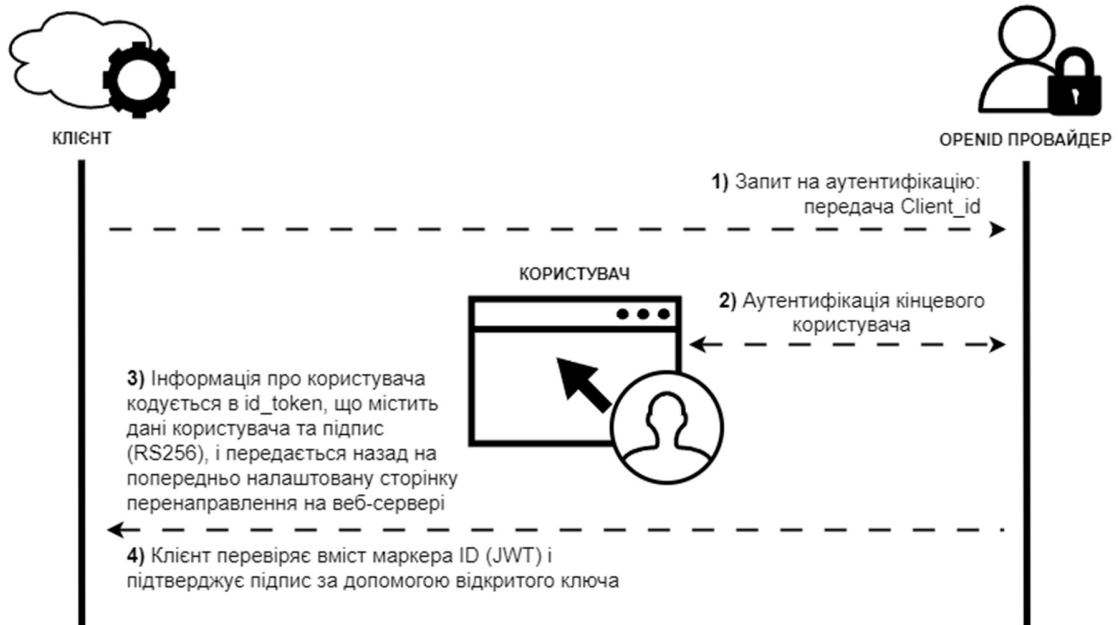


Рисунок 1.7 – Візуалізація неявного потоку

У специфікації OIDC звертається особлива увага, що запитувана інформація про користувача повинна бути упакована в JWT (JSON токен), зашифрована або підписана. Більш того, інформація про ці кроки (як і коли упакована/зашифрована/підписана) повинна бути включена до набору даних, що надсилаються назад.

Зрештою, вимоги до неявного потоку такі: він повинен використовувати схему відкритого/закритого ключа для шифрування або підпису інформації про користувача, а також утримувати у повній конфіденційності `client_secret`.

Таким чином, додаток може переконатися в тому, що дані про користувача (клейми) надходять з надійного джерела (що це не переадресація на підроблений URL) і приватна інформація (наприклад, `client_secret`) повністю захищена від сторонніх очей.

Такий підхід зменшує ризики в разі крадіжки відкритого ключа та ідентифікатора користувача, але ці дані не мають абсолютно ніякої цінності, оскільки тільки система управління доступами володіє необхідною інформацією (URL-адресою переадресації та секретним ключем), щоб використовувати її за призначенням.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

1.5.3. Потік аутентифікації

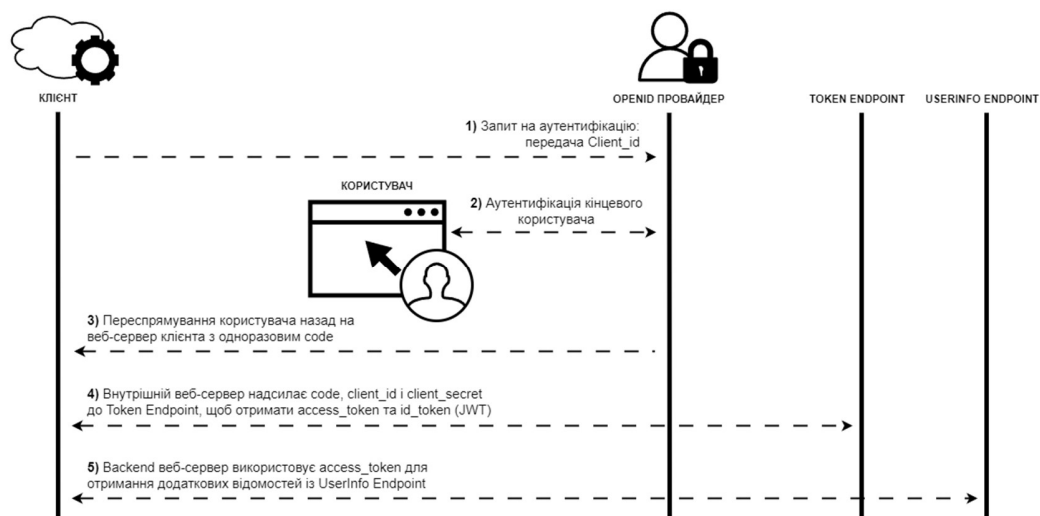


Рисунок 1.8 – Візуалізація потоку аутентифікації

Даний потік розроблений, щоб функціонувати як стандартний three-legged OAuth. У результаті звичайний токен доступу OAuth секретним чином повертається до веб-додатку через виклики, зроблені на серверній частині.

У такому потоці замість передачі даних користувача, служба управління доступами відправляє спеціальний одноразовий маркер, який можна обміняти на токен доступу OAuth через бекенд сервіс. Такий обмін повинен включати ідентифікатор та секрет клієнта на додаток до коду, як і в стандартному потоці OAuth.

Потім токен доступу OAuth має бути використаний веб-сервісом для підтвердження автентичності, а також для запиту різноманітної інформації про користувача (це визначається запитаними типами scopes і клеймами).

Зазвичай клейми попередньо затверджуються адміністратором, який відповідає за налаштування програми, але на аматорському рівні (де безпосередньо кінцевий користувач підтверджує доступ до додатку) вони виступають у вигляді запитів на дозвіл (scopes) під час початкового потоку аутентифікації.

OIDC визначає цілий ряд клеймів, деякі з яких мають досить вузьке визначення (наприклад, тільки email адресу користувача), в той час, як інші є досить великими і можуть повертати будь-яку запитану інформацію системі керування доступами (таку, як, наприклад, інформація про профілі користувача).

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Варто відзначити, що системи OIDC також надають refresh-токен (якщо веб-сервісу необхідний більш тривалий доступ до інформації про користувача), так само, як і персоналізовані клейми.

1.5.4. Загальновідомий endpoint

Клієнту потрібно володіти чималим обсягом інформації, щоб правильно виконувати різні рукописання OIDC та обмінюватися інформацією. Також важливим є питання куди він повинен звернутися, щоб отримати ключі доступу, які використовуються для підпису або шифрування цієї інформації.

Щоб даний спростити процес, провайдери OIDC можуть підтримувати загальновідомий endpoint. У такому разі OIDC самостійно визначає набір опцій, які можуть бути передані клієнту для налаштування вручну. І хоча частина цієї інформації навряд чи зміниться, інші частини спеціально призначені для періодичного оновлення.

Наприклад, запис "jwks_uri": "https://www.googleapis.com/oauth2/v3/certs" у endpoint провайдера Google визначає URL, до якого клієнт може отримати доступ для отримання інформації про будь-які ключі JWK, що використовуються Google, у форматі, встановленому специфікацією OIDC.

Одне з значень, що повертаються для кожного ключа – це ключовий id (kid), який можна використовувати для того, щоб визначити, чи криптографічний ключ змінився. Простіше кажучи, перевібивши цей показник/значення, клієнт може визначити, чи змінився відкритий ключ системи керування доступом з моменту його останнього використання, і автоматично оновитися, щоб використовувати цю інформацію.

Висновки до першого розділу

В даному розділі було розглянуто теоретичні відомості та визначено популярні методи аутентифікації, зокрема за допомогою технології єдиного входу. Було визначено відмінності між усіма методами та їх особливості. Докладно розглянуто принцип роботи OAuth 2.0 та поняття JWT маркерів. В результаті проведеного аналізу можна зробити висновок, що в рамках даного проекту варто реалізувати систему аутентифікації на основі протоколу OpenID Connect та потоку аутентифікації, який потребує серверну частину.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПІДСИСТЕМИ АУТЕНТИФІКАЦІЇ ВЕБ-ДОДАТКУ СОЦІАЛЬНОЇ МЕРЕЖІ

2.1. Вибір провайдера аутентифікації

2.1.1. Визначення критеріїв вибору та області застосування

Отже, Single sign-on (SSO) – це метод аутентифікації, який дозволяє користувачам отримувати доступ до різноманітних систем та додатків, використовуючи всього лише один набір облікових даних. Тобто, що користувачам більше не потрібно запам'ятовувати багато різних паролів для різних систем або вводити облікові дані кожного разу, коли вони переходять до нової системи. Даний підхід вирізняється з поміж інших своєю зручністю, підвищеною безпекою та універсальністю.

Існує величезна кількість провайдерів аутентифікації, кожен з яких орієнтований на окремі сценарії використання та має свої переваги та недоліки. Варто обирати провайдера не тільки за критеріями популярності та функціоналу, а і за тим, на яку групу користувачів даний продукт орієнтовано. Наприклад, досить поширений провайдер Okta представляє собою надзвичайно потужний інструмент, адже він підтримує різні способи та протоколи аутентифікації та має багато корисного функціоналу: панель додатків, з якої можна перейти на потрібний ресурс та одразу виконати вхід шляхом автовведення логіну та паролю або використовуючи технологію єдиного входу, якщо така підтримується веб-додатком[18].

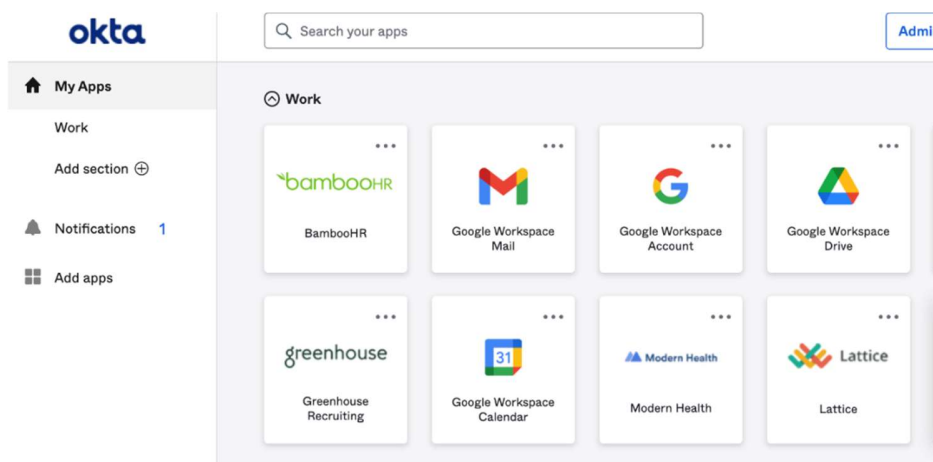


Рисунок 2.1 – Панель додатків в Okta

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

Також список додатків може корегуватися адміністратором, навіть метод аутентифікації може бути змінено відповідно до налагоджених політик.

Але варто розуміти, що поточним завданням є спроектувати підсистему аутентифікації для соціальної мережі, а Okta спрямована на ринок B2B (business-to-business), тобто для користування ресурсом нам потрібно мати обліковий запис Okta, що вже досить рідко зустрічається серед звичайних користувачів. Зазвичай даних продукт використовується інтернет-магазинами, які працюють з обмеженим колом клієнтів. Наявний інструментарій дозволяє досить просто додавати та адмініструвати облікові записи користувачів, які повинні мати доступ до веб-додатку організації.

Отже, в контексті даного проекту варто виділити наступні критерії вибору провайдеру аутентифікації:

- поширеність та зручність: соціальні мережі орієнтовані на широку аудиторію та варто вибрати провайдер, в якому з високою ймовірністю користувач вже буде мати обліковий запис, а отже, буде готовий одразу створити акаунт у нашому додатку;
- мультиплатформність: веб-додаток має бути доступний на різних пристроях з різними операційними системами і варто мінімізувати проблеми, з якими можуть зіштовхнутися користувачі при переході, наприклад, з Android на iOS;
- безпека: провайдер повинен мати відповідно сертифікацію та відповідати всім вимогам принципу технології єдиного входу.

Відповідно до вищевказаних критеріїв варто розглянути такі сервіси, як Apple ID, Google Identity та Azure Active Directory від Microsoft, адже саме ці інструменти здаються найбільш ефективними для вирішення визначених потреб.

2.1.2. Аутентифікація за допомогою Apple ID

Аутентифікація за допомогою Apple дозволяє користувачам виконувати вхід у додаток на всіх платформах за допомогою двофакторної аутентифікації Apple ID[8].

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

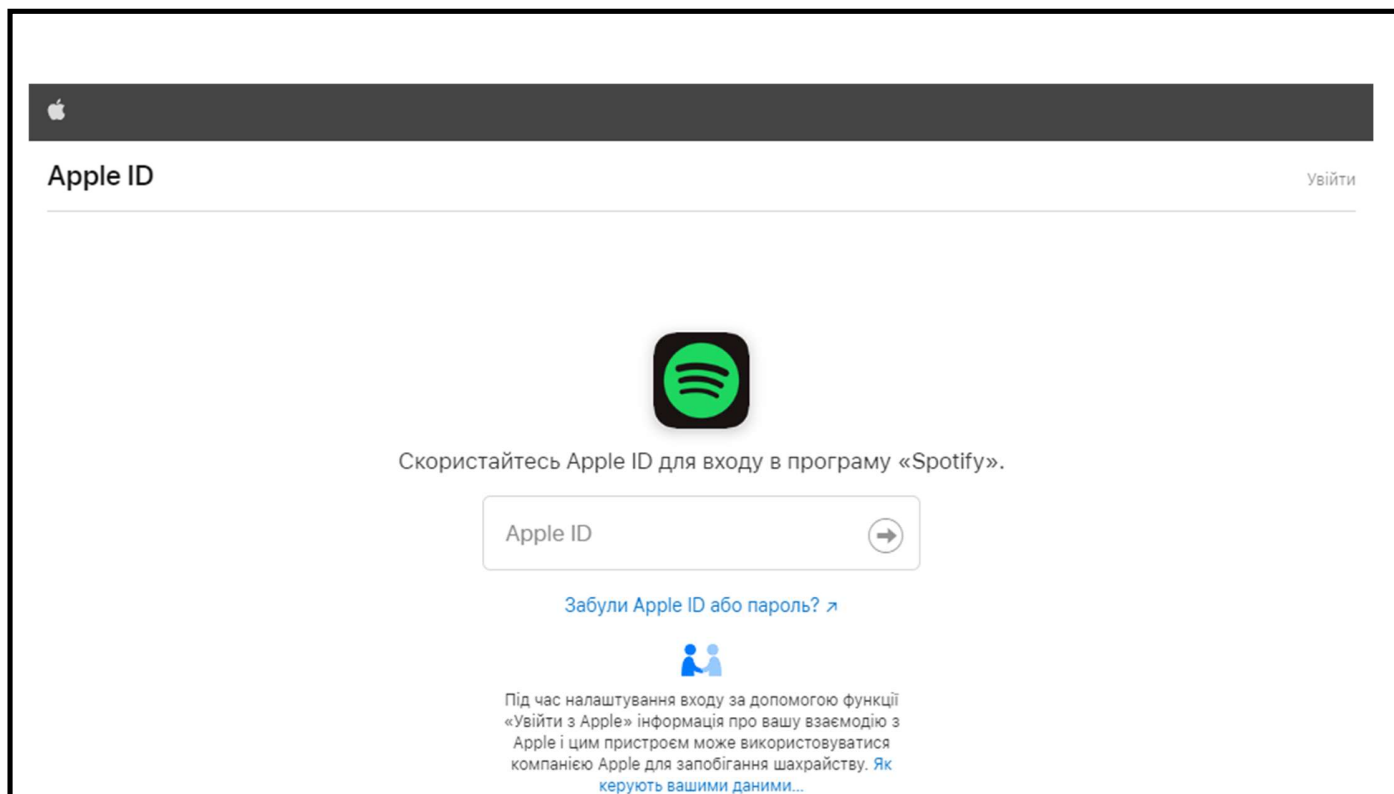


Рисунок 2.2 – Екран входу за допомогою Apple ID

Під час входу додаток отримує маркери та інформацію про користувача, яку можна перевірити на сервері відповідно до наступної діаграми:

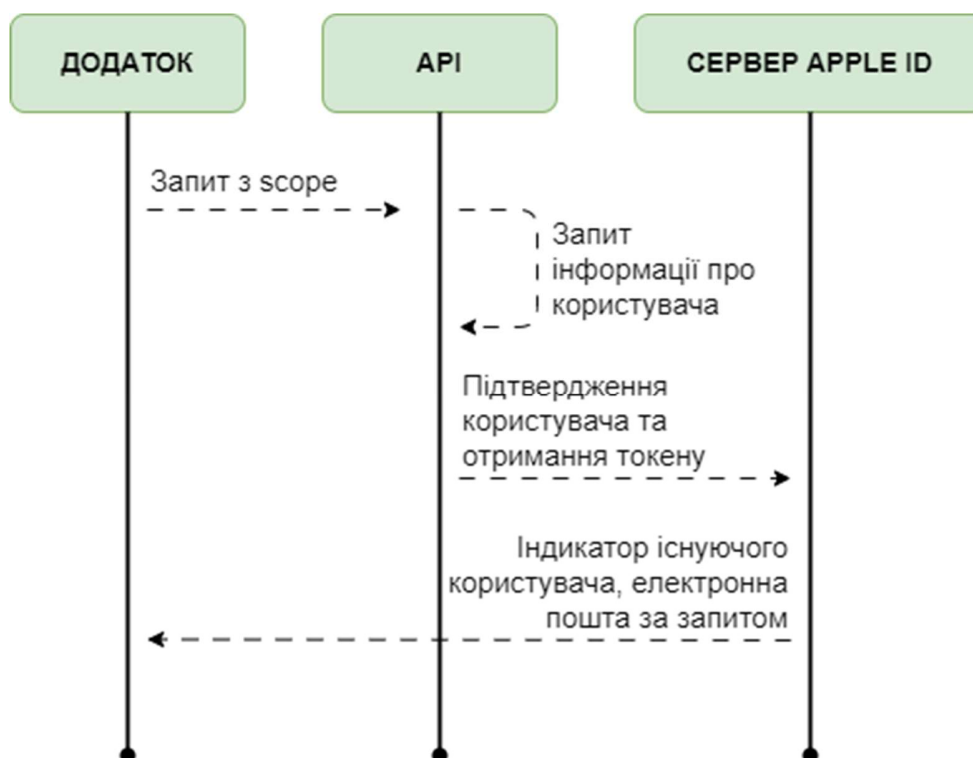


Рисунок 2.3 – Діаграма роботи аутентифікації за допомогою Apple ID

		Москвін Д. В.			KIK.KP.B – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				33
Змн.	Арк.	№ докум.	Підпис	Дата		

Вхід за допомогою Apple захищає облікові записи користувачів за допомогою двофакторної аутентифікації. Користувачі, які входять в систему на пристрої Apple, можуть швидко увійти у відповідну програму такими способами:

- з Face ID або Touch ID на пристроях, захищених паролем;
- за допомогою пароля, якщо Touch ID або Face ID недоступні;
- з паролем Apple ID, якщо пароль не встановлено.

Нативні програми дозволяють лише авторизованому користувачеві iCloud використовувати Вхід за допомогою Apple. Веб-сервіси дозволяють входити за допомогою будь-якого Apple ID.

Варто зауважити, що на пристроях, що не належать Apple, користувачі повинні входити за допомогою свого Apple ID, пароля та коду двофакторної аутентифікації.

Apple визначає, чи є користувач реальною людиною, поєднуючи машинне навчання на пристрої, історію облікових записів і атестацію обладнання за допомогою механізмів збереження конфіденційності. Для визначення того, чи є користувач реальною людиною, є три можливі значення:

- користувач виглядає реальною особою і можна вважати такий обліковий запис дійсним користувачем. В такому випадку можна пропустити будь-які додаткові перевірки на шахрайство або CAPTCHA, які зазвичай використовують додатки;
- система не може визначити, чи є користувач реальною людиною. Сервер може повернути це значення, якщо визначення статусу займає занадто багато часу. В такому випадку рекомендовано поводитися з цим користувачем як з будь-яким іншим обліковим записом з обмеженою інформацією, яка вимагає додаткових кроків перевірки, не варто блокувати сервіс, оскільки користувач все ж може бути реальною людиною.

Дана система визначення реальності користувача доступна лише на операційних системах iOS 14, macOS 11, watchOS 7, tvOS 14 і новіших версіях та налаштована на високу точність і помірний час відкликання. Дану функцію можна використовувати для виявлення шахрайства з обліковим записом.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				34
Змн.	Арк.	№ докум.	Підпис	Дата		

Після того, як користувач увійде у програму за допомогою входу через Apple на одному зі своїх пристроїв, він також зможе увійти на всіх інших своїх пристроях. Видалення програми з пристрою не впливає на цю можливість. Якщо користувач повторно інсталує програму, він може продовжувати використовувати функцію «Увійти за допомогою Apple» на будь-якому зі своїх пристроїв, щоб увійти за допомогою наявного облікового запису.

2.1.3. Єдиний вхід в Azure Active Directory

Вибір методу SSO залежить від того, як програму налаштовано для автентифікації. Хмарні додатки можуть використовувати параметри на основі об'єднання, такі як OpenID Connect, OAuth і SAML. Програма також може використовувати систему єдиного входу на основі пароля, на основі зв'язків або систему єдиного входу, яку можна вимкнути.

Налаштування SSO для роботи між кількома постачальниками ідентифікаційної інформації називається федерацією. Реалізація системи єдиного входу на основі протоколів об'єднання покращує безпеку, надійність, взаємодію з кінцевими користувачами та впровадження. За допомогою об'єднаного єдиного входу Azure AD аутентифікує користувача в програмі за допомогою його облікового запису Microsoft. Цей метод підтримується для програм SAML 2.0, WS-Federation або OpenID Connect. Федеративне SSO є найбагатшим режимом, варто використовувати саме його замість SSO на основі пароля та служб об'єднання Active Directory (AD FS).

Є деякі сценарії, коли опція SSO не доступна для корпоративної програми. Якщо програму було зареєстровано за допомогою реєстрацій програм на порталі, тоді можливість єдиного входу налаштовано на використання OpenID Connect і OAuth за замовчуванням. У цьому випадку параметр єдиного входу не відображатиметься в навігації під корпоративними програмами. Єдиний вхід недоступний, якщо програму розміщено в іншому клієнті або якщо обліковий запис не має необхідних дозволів (глобального адміністратора, адміністратора хмарної програми, адміністратора програми або власника принципала служби).

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				35
Змн.	Арк.	№ докум.	Підпис	Дата		

Веб-програми розміщуються різними компаніями та надаються як послуга. Популярними прикладами веб-додатків є Microsoft 365, GitHub і Salesforce та тисячі інших. Люди отримують доступ до веб-програм за допомогою веб-браузера на своєму комп'ютері. Єдиний вхід дає змогу людям переходити між різними веб-додатками без необхідності багаторазового входу.

Спосіб реалізації SSO залежить від того, де розміщено програму. Хостинг має значення через спосіб маршрутизації мережевого трафіку для доступу до програми. Користувачам не потрібно використовувати Інтернет для доступу до локальних програм (розміщених у локальній мережі). Якщо додаток розміщено в хмарі, користувачам для його використання потрібен Інтернет. Програми, розміщені в хмарі, також називаються програмами «Програмне забезпечення як послуга» (SaaS).

Безпроблемний єдиний вхід Azure Active Directory (Azure AD Seamless SSO) автоматично вводить користувачів, коли вони перебувають на своїх корпоративних пристроях, підключених до корпоративної мережі[19]. Якщо ввімкнено, користувачам не потрібно вводити свої паролі, щоб увійти в Azure AD, і зазвичай навіть вводити свої імена користувачів. Ця функція надає користувачам легкий доступ до хмарних програм без потреби в додаткових локальних компонентах.

Для Windows 10, Windows Server 2016 і пізніших версій рекомендується використовувати SSO через первинний маркер оновлення (PRT). Для Windows 7 і Windows 8.1 рекомендується використовувати Seamless SSO. Безперервна система єдиного входу вимагає, щоб пристрій користувача було приєднано до домену, але вона не використовується на приєднаних пристроях Windows 10 Azure AD або гібридних пристроях, приєднаних до Azure AD. Система єдиного входу на приєднаних Azure AD, приєднаних гібридних Azure AD і зареєстрованих в Azure AD пристроях працює на основі основного маркера оновлення (PRT)

SSO через PRT працює, коли пристрої зареєстровано в Azure AD для гібридного приєднання до Azure AD, приєднаних до Azure AD або особистих зареєстрованих пристроїв через Додати робочий або навчальний обліковий запис.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				36
Змн.	Арк.	№ докум.	Підпис	Дата		

Серед основних переваг аутентифікації через Azure Active Directory можна виділити наступні:

- чудовий досвід користувача: користувачі автоматично входять у локальні та хмарні програми, не потрібно повторно вводити паролі;
- простий у розгортанні та адмініструванні: не потребує додаткових локальних компонентів, працює з будь-яким методом хмарної автентифікації та є можливість розгорнути для деяких або всіх користувачів за допомогою групової політики.

2.1.4. Аутентифікація за допомогою Google

Даний провайдер надає можливість налаштувати обліковий запис Cloud Identity або Google Workspace на використання єдиного входу (SSO)[3]. Якщо ввімкнути систему єдиного входу, користувачам не буде запропоновано ввести пароль, коли вони намагатимуться отримати доступ до служб Google. Натомість вони перенаправляються до зовнішнього постачальника ідентифікаційної інформації (IdP) для аутентифікації.

Використання SSO від Google може надати кілька переваг:

- кращий досвід для користувачів, оскільки вони можуть використовувати свої наявні облікові дані для автентифікації та їм не потрібно вводити облікові дані так часто;
- гарантія, що існуючий IdP залишається системою запису для аутентифікації користувачів;
- не потрібно синхронізувати паролі з Cloud Identity або Google Workspace.

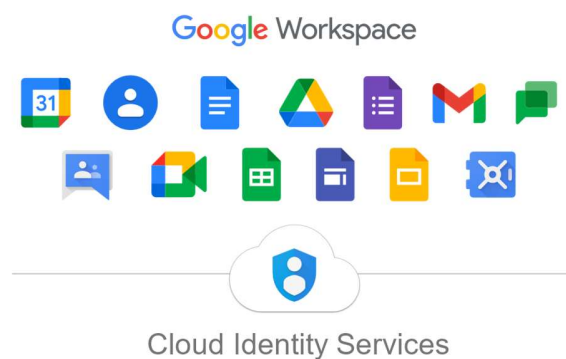


Рисунок 2.4 – Продукти Google Workspace

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				37
Змн.	Арк.	№ докум.	Підпис	Дата		

Щоб використовувати Google SSO, користувач повинен мати обліковий запис користувача в Cloud Identity або Google Workspace і відповідний ідентифікатор у зовнішньому IdP. Тому SSO зазвичай використовується в поєднанні із зовнішнім авторитетним джерелом, яке автоматично надає користувачам доступ до Cloud Identity або Google Workspace.

Варто зауважити, що користувачі з правами суперадміністратора можуть обійти єдиний вхід. Це гарантує, що суперадміністратори можуть отримати доступ до облікового запису, навіть якщо конфігурація SSO неправильна або зовнішній IdP недоступний.

2.1.5. Порівняння провайдерів аутентифікації

Отже, коротко розглянувши дані три провайдера аутентифікації, можна зробити висновки відповідно до визначених раніше критеріїв підбору.

Перш за все, варто сказати, що усі розглянуті методи мають приблизно рівний рівень безпеки, сертифіковані та відповідають всім необхідним вимогам. Кожен з них має реалізацію протоколу OpenID Connect, може запропонувати двофакторну аутентифікацію та різні методи біометричної аутентифікації відповідно до апаратного забезпечення. Тому наш вибір буде залежати від сценаріїв використання, поширеності та зручності для користувача.

Apple ID найкраще інтегрована в операційні системи Apple та демонструє найвищий рівень як зручності користування так і різноманітності та продуманості в плані методів розблокування. Але проблема даного методу в тому, що він орієнтований для користування лише власниками продуктів Apple, а отже, інші користувачі можуть зіштовхнутися з серйозними проблемами та незручностями.

Microsoft Active Directory більш орієнтована на корпоративне користування та велика кількість його функціоналу тісно пов'язана з операційною системою Windows, хоча це не велика проблема, адже за статистикою дана ОС встановлена на близько 75% комп'ютерів. Але даний метод також може використовуватись за межами організацій звичайними особистими обліковими записами Microsoft, відповідно до налаштувань.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

Аутентифікація за допомогою Google виявилась найбільш поширеною та зручною для користувача. Даний метод та продукти Google не мають тісної прив'язки до операційних систем та виробника техніки, хоч і можна знайти деякі незначні переваги в сторону користувачів Android. Майже кожен потенційний користувач нашим майбутнім продуктом буде мати обліковий запис Google, адже навіть власники пристроїв на iOS часто користуються продуктами Google Workspace завдяки їх кросплатформеності, що не притаманно продуктам Apple. Отже, даний метод аутентифікації є найбільш зручним та доступним завдяки мультиплатформенності та поширеності продукції Google.

2.2. Деталі процесу аутентифікації на основі технології єдиного входу

Варто почати з того, що аутентифікація Google, як і інші аналогічні продукти, реалізовує загальний протокол OpenID Connect та з технічної точки зору принцип її роботи буде схожим на інші з деякими відмінностями у функціоналі, наприклад, як ми вже дізналися, Azure Active Directory орієнтована на корпоративне використання та має деякі додаткові налаштування та можливості, але все одно протокол визначає послідовність дій та мінімальний об'єм даних, який нам має надати провайдер.

Серед найбільш часто використовуваних підходів для автентифікації користувача та отримання ідентифікаційного маркера виділяють серверний і неявний (implicit) потоки. Потік сервера дозволяє внутрішньому серверу програми перевірити особу особи за допомогою браузера або мобільного пристрою. Неявний потік використовується у випадку коли програмі на стороні клієнта (як правило, програмі JavaScript, що працює в браузері) потрібно отримати доступ до API безпосередньо, а не через внутрішній сервер. Відповідно до документації Google, рекомендовано реалізовувати серверний потік, адже неявний потік значно складніший через ризики безпеки під час обробки та використання токенів на стороні клієнта.

2.2.1. Аутентифікація користувача

Ми повинні гарантувати безпеку своїх користувачів, запобігаючи атакам підробки запитів. Першим кроком є створення унікального маркера сеансу, який утримує стан між нашою програмою та клієнтом користувача. Пізніше цей унікальний

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

маркер сеансу буде порівняно з відповіддю аутентифікації, яку повертає служба Google OAuth Login, щоб переконатися, що запит робить користувач, а не злоумисник. Ці маркери часто називають маркерами підробки міжсайтових запитів (CSRF).

Відповідно до документації, хорошим вибором для маркера стану є рядок із 30 або близько того символів, створений за допомогою високоякісного генератора випадкових чисел або ж хеш, створений шляхом підписання деяких змінних стану сеансу за допомогою ключа, який зберігається в таємниці на сервері[2].

Наступним кроком є формування HTTPS-GET запиту з відповідними параметрами URI. Варто звернути окрему увагу на використання HTTPS, а не HTTP на всіх етапах цього процесу; HTTP-з'єднання не підтримується з міркувань безпеки. Слід отримати базовий URI з документа Discovery за допомогою `authorization_endpoint` значення метаданих. Розглянемо приклад повного URI аутентифікації OpenID Connect із розривами рядків і пробілами для зручності читання:

```
https://accounts.google.com/o/oauth2/v2/auth?
response_type=code&
client_id=424911365001.apps.googleusercontent.com&
scope=openid%20email&
redirect_uri=https%3A//oauth2.example.com/code&
state=security_token%3D138r5719ru3e1%26url%3Dhttps%3A%2F%2Foauth2-login-demo.example.com%2FmyHome&
login_hint=jsmith@example.com&
nonce=0394852-3190485-2490358&
hd=example.com
```

Рисунок 2.5 – Приклад повного URI аутентифікації

Перейшовши за даним посиланням, користувачі повинні дати згоду, якщо програма запитує будь-яку нову інформацію про них або доступ до облікового запису, який вони раніше не схвалили.

Далі надсилається відповідь на вказаний нами в запиті `redirect_uri`. На сервері ми повинні підтвердити, що `state` отриманий від Google збігається з маркером сеансу, який ми створили на першому кроці. Ця двостороння перевірка допомагає переконатися, що запит робить користувач, а не злоумисник.

Відповідь містить `code` параметр, одноразовий код авторизації, який сервер може обміняти на маркер доступу та ідентифікатор. Сервер здійснює цей обмін, надсилаючи POST запит HTTPS. Запит POST надсилається кінцевій точці маркера, яку ми повинні

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

отримати з документа Discovery за допомогою token_endpoint значення метаданих. Тіло запиту має виглядати наступним чином:

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7&
client_id=your-client-id&
client_secret=your-client-secret&
redirect_uri=https%3A//oauth2.example.com/code&
grant_type=authorization_code
```

Рисунок 2.6 – Приклад запиту обміну code

З відповіді ми отримаємо ряд токенів, серед яких є ідентифікаційний маркер – це JWT (JSON Web Token), тобто об’єкт JSON із криптографічним підписом і кодуванням Base64. Зазвичай важливо підтвердити маркер ідентифікатора перед його використанням, але оскільки ми комунікуємо безпосередньо з Google через канал HTTPS без посередників і використовуємо секрет клієнта для автентифікації в Google, ми можемо бути впевнені, що маркер дійсно надходить від Google і є дійсним.

Отримавши інформацію про користувача з маркера ідентифікатора, ми повинні зробити запит до бази даних користувачів програми. Якщо користувач уже існує у нашій базі даних, ми можемо почати сеанс програми для цього користувача, якщо відповідь Google API відповідає всім вимогам входу.

Якщо користувача немає у нашій базі даних користувачів, його слід перенаправити до процесу реєстрації нового користувача. Можна автоматично зареєструвати користувача на основі отриманої від Google інформації, або, принаймні, попередньо заповнити багато полів, які вимагає реєстраційна форма. Окрім інформації в токени ідентифікатора, є можливість отримати додаткову інформацію профілю користувача в кінцевих точках профілю користувача.

2.2.2. Оновлення токенів

У своєму запиті на доступ до API можна подати запит на повернення маркера оновлення під час обміну code[2]. Термін дії токенів доступу періодично закінчується,

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				41
Змн.	Арк.	№ докум.	Підпис	Дата		

і облікові дані стають недійсними для відповідного запиту API. Маркер оновлення надає додатку безперервний доступ до API Google, поки користувач не присутній у додатку. Щоб надіслати запит на маркер оновлення, потрібно додати значення offline параметра access_type у запит автентифікації.

Варто зберігати токен оновлення безпечно та постійно, оскільки отримати токен оновлення можливо лише під час першого виконання потоку обміну кодом.

Існують обмеження на кількість виданих токенів оновлення: один ліміт на комбінацію клієнт/користувач, а інший на користувача для всіх клієнтів. Якщо програма запитує забагато маркерів оновлення, вона може перевищити ці обмеження, і в цьому випадку старіші маркери оновлення перестануть працювати.

Запит офлайн-доступу є вимогою для будь-якої програми, яка потребує доступу до Google API, коли користувач відсутній. Наприклад, програма, яка виконує служби резервного копіювання або виконує дії в заздалегідь визначений час, повинна мати можливість оновлювати свій маркер доступу, коли користувач відсутній. Стандартний стиль доступу називається online.

Серверні веб-програми, встановлені програми та пристрої отримують маркери оновлення під час процесу авторизації. Маркери оновлення зазвичай не використовуються у веб-додатках на стороні клієнта (JavaScript).

2.2.3. Відкликання токенів

У деяких випадках користувач може скасувати доступ до програми, відвідавши налаштування облікового запису. Програма також може програмно скасувати наданий їй доступ. Програмне скасування є важливим у випадках, коли користувач скасовує підписку, видаляє програму або ресурси API, необхідні програмі, суттєво змінилися. Іншими словами, частина процесу видалення може включати запит API, щоб переконатися, що дозволи, надані програмі, видалено.

Щоб програмно відкликати маркер, програма надсилає запит <https://oauth2.googleapis.com/revoke> і включає маркер як параметр[2]. Маркер може бути маркером доступу або маркером оновлення. Якщо маркер є маркером доступу та має відповідний маркер оновлення, маркер оновлення також буде відкликано.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

2.2.4. Документ Discovery

Протокол OpenID Connect вимагає використання кількох кінцевих точок для аутентифікації користувачів і для запиту ресурсів, зокрема маркерів, інформації про користувача та відкритих ключів[20].

Щоб спростити впровадження та підвищити гнучкість, OpenID Connect дозволяє використовувати «документ виявлення», документ JSON, знайдений у добре відомому місці, що містить пари ключ-значення, які надають детальну інформацію про конфігурацію постачальника OpenID Connect, включаючи URI авторизації, маркер, відкликання, інформація про користувача та кінцеві точки відкритих ключів. Документ Discovery для служби Google OpenID Connect можна отримати з <https://accounts.google.com/.well-known/openid-configuration>.

Щоб використовувати служби Google OpenID Connect, слід сконфігурувати URI документа Discovery у нашій програмі, таким чином програма отримує документ, застосовує правила кешування у відповіді, а потім за потреби отримує з нього URI кінцевої точки. Наприклад, щоб аутентифікувати користувача, ми отримаємо `authorization_endpoint` значення метаданих як базовий URI для запитів автентифікації, які надсилаються в Google.

2.2.5. Валідація ID токenu

Потрібно перевірити всі ідентифікаційні маркери на сервері, якщо немає впевненості, що вони надійшли безпосередньо від Google. Наприклад, сервер повинен підтвердити автентичність будь-яких маркерів ідентифікатора, які він отримує від клієнтських програм.

Варто надсилати ідентифікаційні маркери на сервер із запитом, які потребують автентифікації. Ідентифікаційні токени повідомляють конкретного користувача, який робить запит, і для якого клієнта було надано цей ідентифікаційний токен.

Ідентифікаційні маркери є чутливими і можуть бути використані зловмисниками у разі перехоплення. Варто переконатися, що ці маркери обробляються безпечно, передаючи їх лише через HTTPS і лише через дані POST або в заголовках запиту.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Одна річ, яка робить ідентифікаційні маркери корисними, полягає в тому, що ми можемо передавати їх між різними компонентами програми. Ці компоненти можуть використовувати ідентифікаційний маркер як легкий механізм автентифікації, що аутентифікує програму та користувача. Але перш ніж використовувати інформацію в маркері ID або покладатися на неї як на твердження про те, що користувач пройшов автентифікацію, варто перевірити її.

Валідний ID маркер має відповідати наступним вимогам[2]:

1. Правильно підписаний емітентом. Випущені Google маркери підписуються за допомогою одного із сертифікатів, знайдених за URI, указаним у `jwks_uri` значенні метаданих документа Discovery.
2. Значення клейму `iss` в маркері дорівнює `https://accounts.google.com` або `accounts.google.com`.
3. Значення клейму `aud` в маркері дорівнює ідентифікатору клієнта нашої програми.
4. Термін дії (`exp` клейму) маркера не минув.
5. Якщо було вказано значення параметра `hd` у запиті, маркер має клейм `hd`, який відповідає прийнятному домену, пов'язаному з організацією Google Cloud.

Перший пункт передбачає перевірку криптографічного підпису. З метою налагодження можна використовувати кінцеву точку `Google tokeninfo` для порівняння з локальною обробкою, реалізованою на сервері чи пристрої. Припустимо, що значення токена ID дорівнює `XYZ123`. Якщо підпис маркера дійсний, відповіддю буде корисне навантаження JWT у формі декодованого об'єкта JSON.
`https://oauth2.googleapis.com/tokeninfo?id_token=XYZ123`

Кінцева `tokeninfo` точка корисна для налагодження, але у `production` режимі варто отримувати відкриті ключі Google із кінцевої точки ключів і виконувати перевірку локально. Слід отримати URI ключів із документа Discovery за допомогою `jwks_uri` значення метаданих. Запити до кінцевої точки налагодження можуть бути обмежені або іншим чином піддаватися періодичним помилкам.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Оскільки Google рідко змінює свої відкриті ключі, можна кешувати їх за допомогою директив кешу відповіді HTTP і, у переважній більшості випадків, виконувати локальну перевірку набагато ефективніше, ніж за допомогою кінцевої точки tokeninfo. Ця перевірка вимагає отримання та аналізу сертифікатів, а також виконання відповідних криптографічних викликів для перевірки підпису. На щастя, для цього існують добре налагоджені бібліотеки, доступні багатьма мовами.

2.2.6. Механізм Proof-of-Procession

За замовчуванням маркери доступу OAuth є так званими маркерами-носіями, тобто вони не прив'язані до клієнта, і кожен, хто володіє токеном, може ним користуватися.

OAuth 2.0 Demonstrating Proof-of-Possession (DPoP), допомагає запобігти використанню несанкціонованими чи нелегітимними сторонами витікших чи викрадених маркерів доступу[10]. Даний механізм працює на рівні програми для обмеження відправника маркерів доступу та оновлення, що допомагає запобігти повторному відтворенню маркерів на різних кінцевих точках.

DPoP дозволяє клієнту підтвердити володіння парою відкритий/приватний ключ, включивши заголовок DPoP у /token.

Даний механізм є доповненням до вищерозглянутого принципу аутентифікації з моменту отримання одноразового коду авторизації та складається з наступних кроків:

1. Клієнт генерує пару відкритий/приватний ключ для використання з DPoP.
2. Клієнт додає відкритий ключ у заголовок JWT і підписує JWT закритим ключем.
3. Клієнт додає JWT до DPoP заголовка запиту та надсилає запит до /token
4. Сервер авторизації перевіряє DPoP заголовок і надсилає назад повідомлення про помилку «Сервер авторизації вимагає nonce у підтвердження DPoP» і включає заголовок dpop-nonce у відповідь.
5. Клієнт додає значення nonce та jti до корисного навантаження JWT, оновлює заголовок запиту новим значенням JWT і знову надсилає запит маркера доступу.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

6. Сервер авторизації прив'язує відкритий ключ до маркера доступу та надсилає відповідь.

7. Клієнт надсилає запит на доступ до ресурсу та включає в заголовок маркер доступу, прив'язаний до DPoP, і доказ JWT DPoP.

8. Ресурс перевіряє прив'язаний до DPoP маркер доступу, перевіряючи, що відкритий ключ перевірки DPoP JWT у заголовку DPoP збігається з відкритим ключем, до якого прив'язаний маркер доступу. Після успішної перевірки ресурс надає доступ.

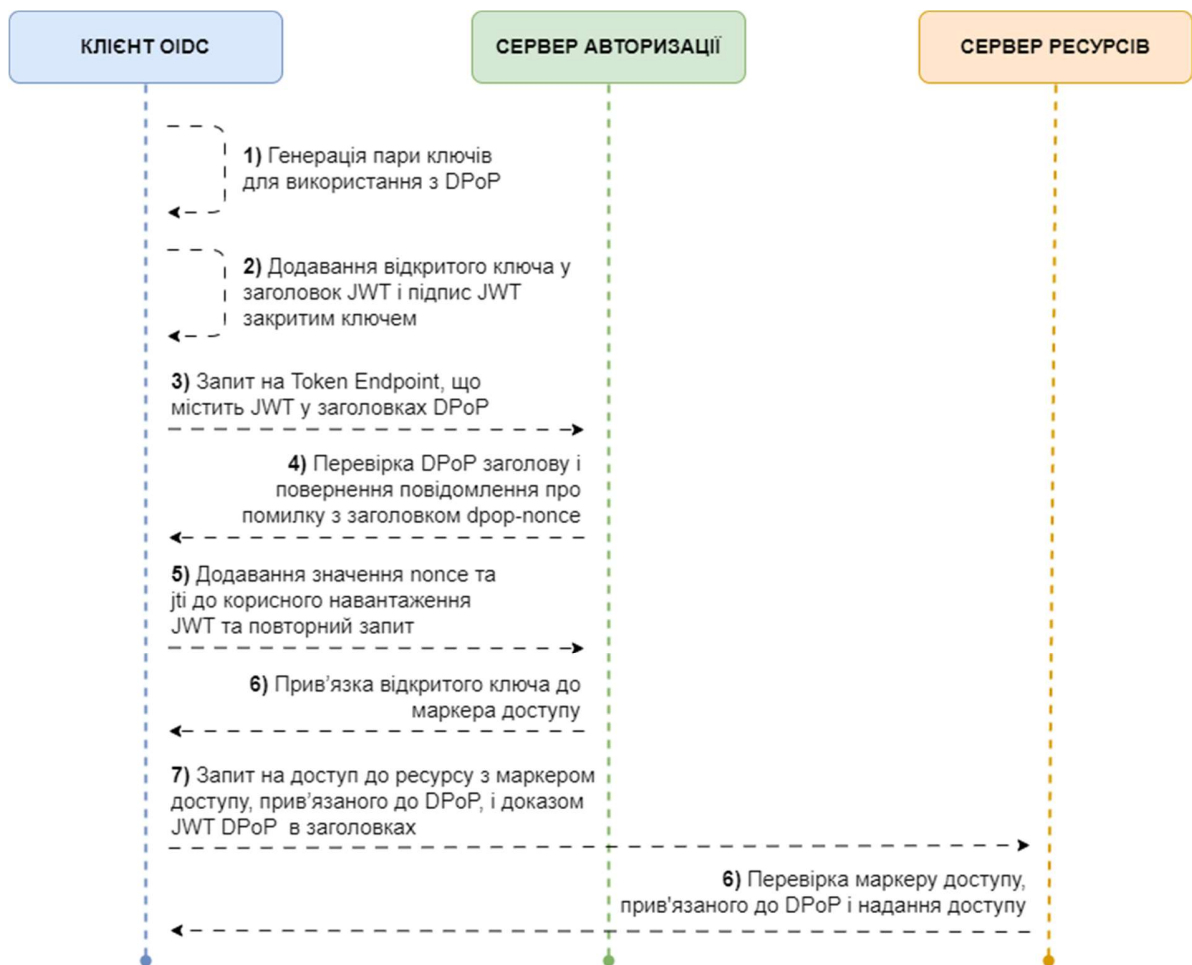


Рисунок 2.7 – Візуалізація принципу роботи DPoP

На жаль, даний захист поки не є загальноприйнятим та поки не підтримується Google, хоча в Okta вже запровадили таку систему.

2.3. Проектування архітектури веб-додатку

Отже, ми визначили, що нам варто реалізувати серверний flow, тому наш додаток буде мати серверну частину, на якій буде здійснюватися робота з даними. Для

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

гнучкості та масштабованості варто реалізувати сервер як API, який на відповідні запити буде давати відповідь у вигляді JSON. Такий підхід надасть можливість використовувати наш сервер як веб-додатку, так і мобільному або десктопному додатку у майбутньому.

2.3.1. Чотирирівнева архітектура

У рамках даної роботи нам потрібно реалізувати потужну основу, яка дасть можливість функціоналу соціальної мережі працювати з сесіями користувачів. У нашому додатку ми буде запроваджено чотирирівневу архітектуру, що дозволить без проблем розширювати можливості додатку у майбутньому. Програма буде складатися з чотирьох рівнів, які незалежні один від одного та які у будь-який момент можуть бути замінені на аналоги:

- рівень представлення (PL);
- рівень обслуговування даних (DSL);
- рівень бізнес-логіки (BLL);
- рівень доступу до даних (DAL).

Розглянемо дану архітектуру на діаграмі:

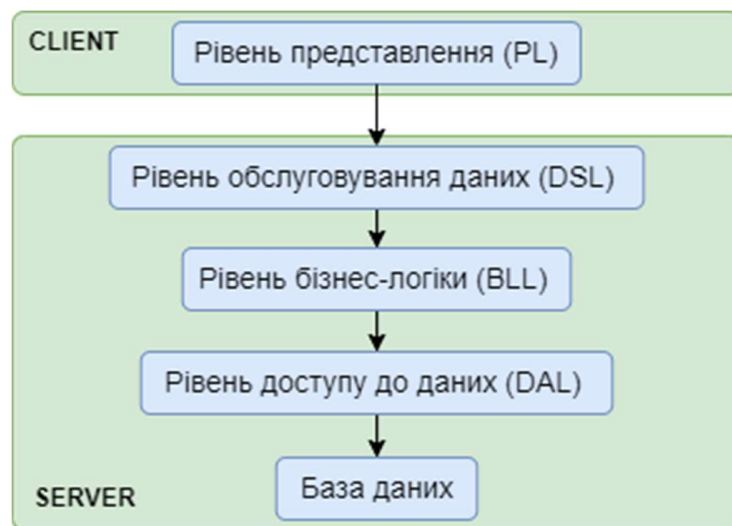


Рисунок 2.8 – Діаграма чотирирівневої архітектури

Як видно з діаграми, відповідно до географічного положення, шари можуть бути поділяються на два види. Одним з них є рівень доступу до даних, рівень бізнес-логіки та рівень служб даних яка розташована на стороні сервера, а також модель бізнес-

сутності та загальну бібліотеку класів. Інший – це рівень презентації, розташований у клієнті.

2.3.2. Рівень представлення

Рівень представлення (PL) він розташований у зовнішньому шарі та представляє собою інтерфейс, з яким працюють користувач. Його функції включають отримання введених даних, інтерпретацію інструкцій користувачів, надсилання запитів на рівень служб даних і відображення даних, отриманих із рівня служб даних користувачам у спосіб, який вони можуть зрозуміти. Він максимально наближений до користувачів і забезпечує інтерактивний інтерфейс роботи[21].

У контексті веб-додатку соціальної мережі, наш клієнт має забезпечувати швидкість та зручність. Для цього варто реалізувати асинхронну роботу з даними, тобто ментально реагувати на дії користувача, а дані показувати по мірі їх готовності. Одним із сучасних підходів є відображення placeholder-компонентів:

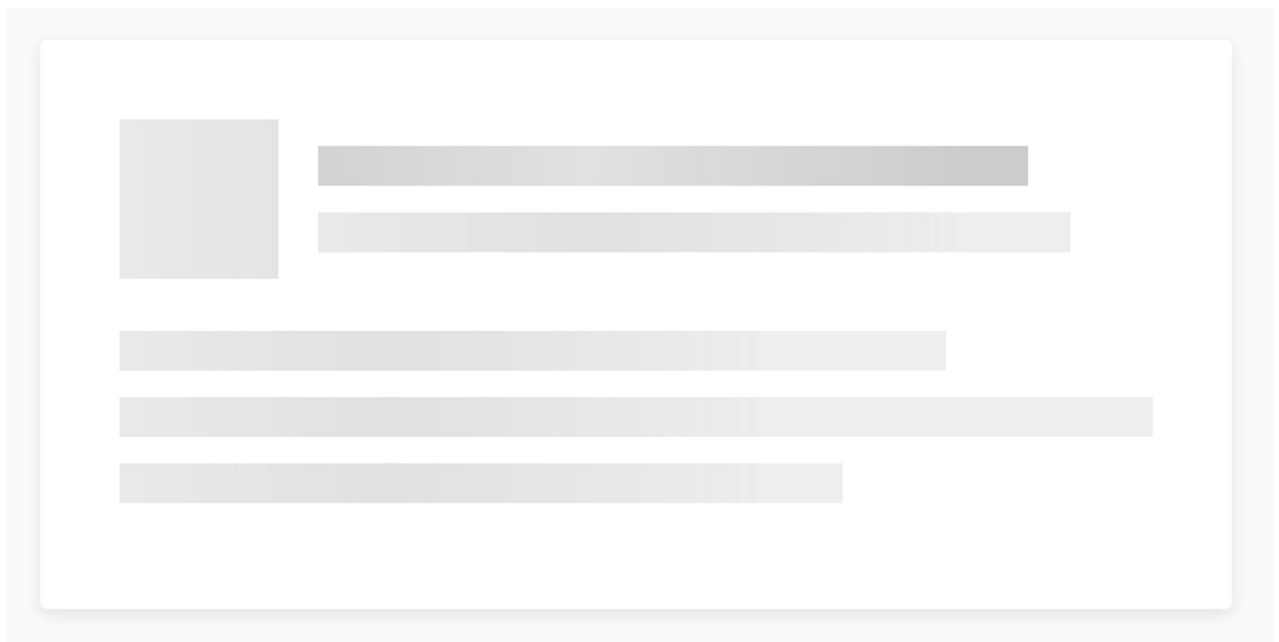


Рисунок 2.9 – Відображення placeholder-компонентів під час завантаження даних

Також буде доречно реалізувати можливість роботи з умовними вкладками. Іншими словами, якщо користувач знаходиться в чаті та вирішив тимчасово переключитися на свій профіль, ми маємо зберегти стан вкладки чату. Такий підхід є дуже зручним, адже користувачу не потрібно буде знову заходити в чат, в якому він

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

був, внаслідок чого не потрібно буде повторно завантажувати дані з серверу, вони вже будуть збережені на клієнтській стороні.

З вищевказаними вимогами чудово впорається клієнтський веб-додаток на основі JavaScript бібліотек React, Redux та Redux-Saga.

React – це відкрита JavaScript-бібліотека, яка використовується для створення інтерфейсів користувача та вирішує проблему часткового оновлення вмісту веб-сторінок, з якими стикаються в розробці односторінкових застосунків (SPA). Розробляється Meta (раніше Facebook) і спільнотою індивідуальних розробників.

React спрощує створення інтерактивних інтерфейсів. Розробнику потрібно лише описати яким чином різні частини інтерфейсу мають виглядати у кожному стані додатку і React ефективно оновить та відобразить лише потрібні компоненти, коли дані зміняться і головне – зробить це без перезавантаження сторінки. Декларативні інтерфейси роблять код більш передбачуваним, а отже, його набагато простіше налагоджувати та підтримувати.

Інтерфейси з використанням React засновані на компонентах, тобто можна створювати інкапсульовані компоненти, які керують власним станом, а з них будувати складні інтерфейси. Логіка компонентів написана на JavaScript, що дає широкий спектр можливостей, за допомогою яких можна легко складні дані у додатку і зберігати стан окремо від DOM.

Redux є відкритою JavaScript бібліотекою, яка призначена для керування станом JavaScript-додатків. Зазвичай використовується у комбінації з React або Angular для побудови інтерфейсів користувача.

Redux побудований на трьох основних концепціях:

- єдине джерело істини;
- стан призначений тільки для читання;
- зміни за допомогою чистих функцій.

Redux зберігає всю інформацію про стан застосунку в єдиному місці – у дереві об'єктів в одному сховищі. Такий підхід спрощує налагодження та перевірку програми,

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

оскільки весь стан доступний з одного джерела. Крім того, це дозволяє зберігати стан застосунку під час розробки, щоб прискорити процес розробки.

Стор (Store) – це об'єкт, який об'єднує екшени, які сигналізують про факт події, і редюсери, які оновлюють стан (state) додатку відповідно цих подій.

Запуск екшену є єдиним способом змінити стан, що гарантує його цілісність та неможливість модифікації під час перегляду. Всі зміни централізовані і відбуваються одна за одною у чітко заданій послідовності. Екшени являються єдиними джерелами інформації для стану.

Редюсери (reducers) являють собою лише чисті функції, які приймають попередній state додатку і action та повертають наступний стан. Іншими словами, вони визначають яким чином state додатка змінюється у відповідь на відповідний екшен. Варто зауважити, що екшени можуть лише описувати подію, а не як само змінюється стан додатку.

Redux Saga – це проміжне ПЗ (middleware), яке відповідає за керування екшенами до того, як безпосередньо досягне reducers. Дана бібліотека буде у нагоді для керування запитами на сервер та в обробці результату дії або підвантаження потрібних даних під час переходу між сторінками. Все це буде відбуватися асинхронно, що дасть нам можливість відреагувати на дію користувача запитом на сервер, дочекатися його завершення та, відповідно до результату, відобразити отримані дані або ж видати помилку та у окремих випадках зробити додаткову дію.

2.3.3. Рівень обслуговування даних

Рівень обслуговування даних (DSL) розташований між рівнем представлення та рівнем бізнес-логіки (BLL). Як ізоляційний рівень він відокремить бізнес-логіку від клієнта, щоб гарантувати безпеку інформації. Відповідно до потреб кожного модуля рівень служб даних інкапсулює бізнес-логіку на високому рівні. Оперативна діяльність відігравала роль конфіденційності. Для великих програмних систем важливе значення мають міжплатформні розподілені обчислення та ферми серверів між комунікаціями, що є функцією встановлення рівня обслуговування. Основною функцією DSL є передача даних, оброблених BLL, на його безпосередній верхній рівень (рівень

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

представлення) або передача даних, надісланих PL, на його безпосередньо нижчий рівень, тобто BLL, відповідно до вказаних визначень моделі[21].

В якості DSL реалізуємо API на основі технології GraphQL, що забезпечить неймовірну гнучкість, можливість керувати правами для доступу до окремих полів та оптимальну роботу з даними.

GraphQL – це мова запитів і маніпуляції даними з відкритим кодом для API і середовище виконання для обслуговування запитів з наявних даних, яка була розроблена Facebook у 2012 році, а публічний реліз відбувся у 2015 році[22].

Факт того, що дана мова була створена саме у Facebook для проекту з великим обсягом різноманітних даних, сигналізує про те, що при роботі над подібним продуктом можуть виникати ситуації з обмеженнями REST-архітектури. Наприклад, отримання профілю користувача, його постів та коментарів з першого погляду виглядає досить простим завданням. Але якщо врахувати обсяг даних у системі та припустити, що всі ці дані зберігаються у різних базах даних, для цього знадобиться створити кілька REST-endpoint'ів, а якщо обсяг даних дійсно великий і має декілька джерел даних, стає зрозуміло мету розробки нового підходу до роботи з API. В основі такої реалізації лежить принцип запровадження одного розумного endpoint, який буде здатний працювати зі складними запитами і повертати дані саме в тій формі і в тому обсязі, які необхідні клієнту.

У центрі будь-якої реалізації GraphQL API лежить схема даних, тобто опис того, з якими типами даних він може працювати і які типи даних може повернути у відповідь запит. Іншими словами, для роботи з будь-яким API користувачеві необхідно знати, які типи об'єктів можна отримати, які поля вибрати, які доступні у внутрішніх об'єктах тощо.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

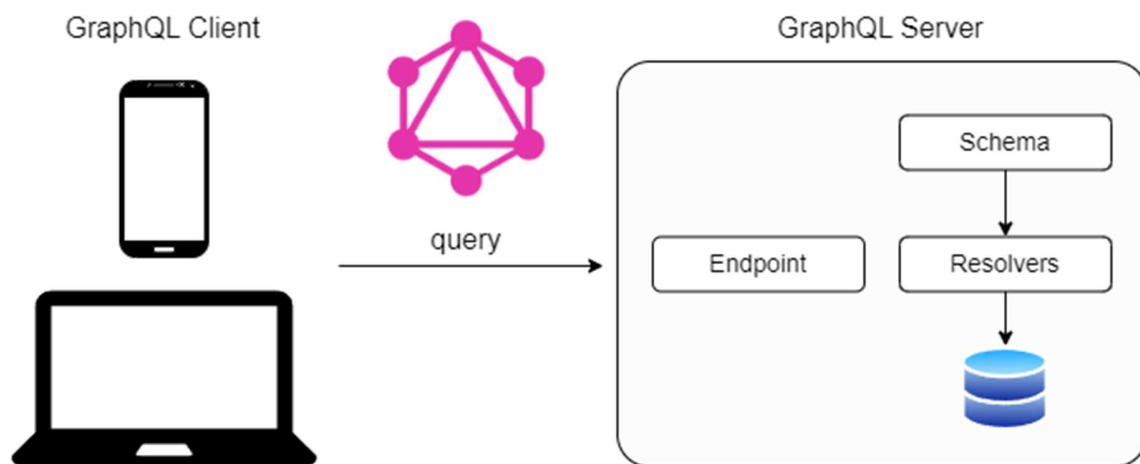


Рисунок 2.10 – Схема взаємодії клієнта та сервера

Зараз GraphQL підтримують різноманітні платформи такі як веб, Android, iOS та інші. GraphQL-клієнт надсилає запит на отримання даних (query) або їх зміну (mutation), складений відповідно до схеми, на GraphQL-сервер. GraphQL-сервер, в свою чергу, є HTTP-сервером, з яким пов'язана схема GraphQL, тобто через цю схему проходять всі запити, отримані від клієнта, та відповіді, що повертаються.

Сервер GraphQL не може знати, що робити із запитом, якщо йому не «пояснити» це за допомогою спеціальних функцій, завдяки яким GraphQL розуміє, як отримати дані для полів, що запитуються. Ці функції пов'язані з відповідними полями і називаються розпізнавачами або резолверами (resolvers). Після цього клієнту повертається відповідь, яка відображає структуру даних, що запитуються з клієнта, зазвичай у форматі JSON.

Отже, у GraphQL є ряд переваг у порівнянні з REST:

- клієнтоорієнтованість, тобто можливість клієнту отримати лише ту інформацію і саме в тому обсязі, який йому потрібний, не більше і не менше;
- лише один endpoint;
- строга типізація, що дозволяє заздалегідь оцінити правильність запиту до виконання програми;
- можливість поєднувати запити;
- запити завжди повертають очікуваний результат, відповідно до схеми даних;

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

- менша кількість даних, що передаються клієнту, оскільки клієнт запитує лише необхідні дані. Тобто, якщо при використанні REST кінцева точка визначає те, які дані і в якій формі отримає клієнт, то при використанні GraphQL клієнт не повинен запитувати зайві дані, тим самим зменшуючи обсяг відповіді сервера і покращуючи безпеку даних.

Підсумовуючи, хотілося б відзначити, що GraphQL – це концепція створення API, яка забезпечує слабку зв'язність клієнта та сервера, що чудово вписується у наш підхід з реалізацією чотирирівневої архітектури.

2.3.4. Рівень бізнес-логіки

Рівень бізнес-логіки (BLL) він розташований між PL і рівнем доступу до даних (DSL), відіграючи сполучну роль в обміні даними. Рівень бізнес-логіки відповідає за різні типи бізнес-операцій системи, виконання відповідних функцій, які є операціями, пов'язаними з задачею, обробку бізнес-логіки даних. Задачі рівня зосереджені в основному на розробці бізнес-правил, бізнес-процесів і системи, пов'язаної з бізнес-потребами, тобто це пов'язано з областю систем, до яких звертається логіка. Дуже часто він також відомий як доменний рівень[21].

Наш сервер буде побудований на основі .NET 6 – модульної платформи для розробки програмного забезпечення з відкритим кодом. Дана платформа сумісна з операційними системами Windows, Linux та macOS, випущена та підтримується компанією Microsoft, що надає багато переваг під час розробки, адже все досить якісно документовано та підтримується на усіх популярних ОС, та надає обширні можливості розгортання за допомогою сервісу Microsoft Azure.

Шоста версія була випущена у кінці 2021 року та є LTS (Long Term Support), що означає, що стабільний реліз підтримуватиметься довгий час і вважатиметься стандартною версією до поширення, тобто гарантує, що ми будемо досить довгий час використовувати актуальну версію платформи і нам не потрібно буде витрачати додаткові ресурси на оновлення до новішої версії.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

Програми .NET забезпечують більш швидкий час відгуку та вимагають менше обчислювальної потужності. Вони мають надійні вбудовані заходи безпеки та ефективно виконують серверні завдання, такі як доступ до бази даних.

2.3.5. Рівень доступу до даних

Рівень доступу до даних (DAL) розташований на внутрішньому рівні, який реалізує логіку збереження. Функція цього рівня відповідає забезпечення доступу до системи баз даних, двійкових файлів, текстових документів або документа XML на інші. Операції над даними включають пошук, додавання, видалення, зміну тощо. Цей рівень працює незалежно, не покладаючись на інші рівні. Відповідно до запитів верхнього рівня, DAL витягує відповідні дані з бази даних і передає дані верхньому. DAL також виконує операції CRUD над даними в базі даних відповідно до інструкцій вище[21].

У нашому проєкті за рівень доступу буде відповідати ORM-система Entity Framework Core.

Перш за все, ORM (Object-relational mapping) є технологією, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних»[24].

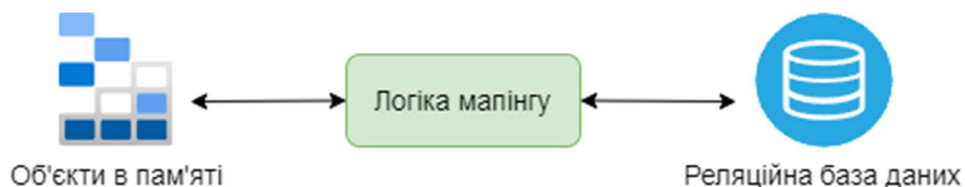


Рисунок 2.11 – Візуалізація принципу роботи ORM

Entity Framework (EF) Core – це легка, розширювана кросплатформна версія популярної технології доступу до даних Entity Framework із відкритим кодом[23]. EF Core являє собою ORM систему, яка:

- дозволяє розробникам .NET працювати з базою даних за допомогою об'єктів;
- усуває потребу в більшості коду доступу до даних, який зазвичай потрібно писати.

У EF Core доступ до даних здійснюється за допомогою моделі. Модель складається з класів сутностей і об'єкта контексту, який представляє сеанс із базою даних. Об'єкт контексту дозволяє запитувати та зберігати дані.

EF підтримує наступні підходи до розробки моделей:

- створення моделі із існуючої бази даних;
- ручне кодування моделі відповідно до бази даних;
- створення бази даних з моделі за допомогою EF Migrations. Міграції

дозволяють розвивати базу даних у міру зміни моделі.

В якості бази даних будемо використовувати Microsoft SQL Server, що дасть нам можливість комфортно та продуктивно працювати з вищевказаними технологіями. Також перевагою є можливість розгортання бази даних в хмарі сервісу Microsoft Azure, що знову ж таки буде зручно та оптимально з точки зору підтримки та адміністрування, адже всі наші сервіси будуть розгорнуті в одній системі.

Висновки до другого розділу

В даному розділі було розглянуто реалізації провайдерів аутентифікації від відомих організацій, таких як Apple, Google та Microsoft, було визначено критерії вибору та, відповідно до них, обрано варіант від Google, адже він є найбільш зручним та доступним завдяки мультиплатформенності та поширеності продукції Google. Було покроково розглянуто алгоритм аутентифікації користувачів за допомогою Google SSO та вимоги до валідації токенів. Відповідно до технічних вимог підсистеми аутентифікації було спроектовано проект веб-додатку соціальної мережі з використанням чотирирівневої архітектури, що забезпечить перспективу розширення функціоналу у майбутньому, а також можливість замінити один із рівнів на будь-який аналог. Визначено стек технологій, які будуть використані для реалізації даної підсистеми, а саме: React, Redux, Redux-Saga на клієнтській частині та .NET 6, GraphQL, Entity Framework Core і Microsoft SQL Server на стороні сервера.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ АУТЕНТИФІКАЦІЇ НА ОСНОВІ ТЕХНОЛОГІЇ ЄДИНОГО ВХОДУ

3.1. Налаштування Google API Console

Перш ніж наш веб-додаток зможе використовувати для входу користувачів систему аутентифікації Google OAuth 2.0, необхідно налаштувати проект у Google API Console, щоб отримати облікові дані OAuth 2.0, установити URI перенаправлення та (необов'язково) налаштувати фірмову інформацію, яку користувачі будуть бачити на екрані згоди. Також можна використовувати консоль API для створення облікового запису служби, увімкнення виставлення рахунків, налаштування фільтрації та виконання інших завдань.

Першим кроком налаштування є створення проекту у Google API Console:

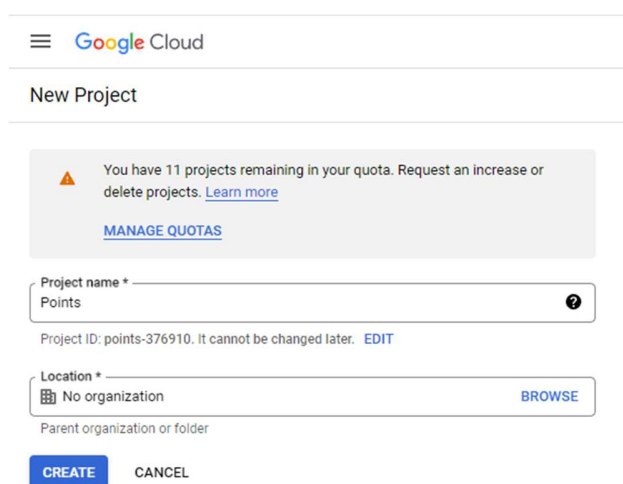


Рисунок 3.1 – Створення проекту в Google API Console

Далі потрібно перейти на сторінку облікових даних:

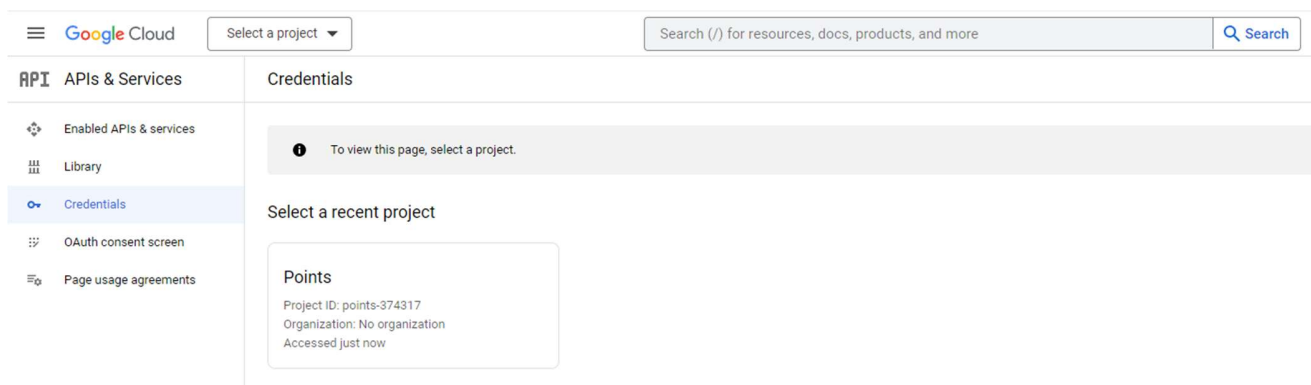


Рисунок 3.2 – Сторінка облікових даних

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

Наступним кроком створимо облікові дані OAuth 2.0 client ID попередньо обравши створений на першому кроці проект у випадяючому списку:

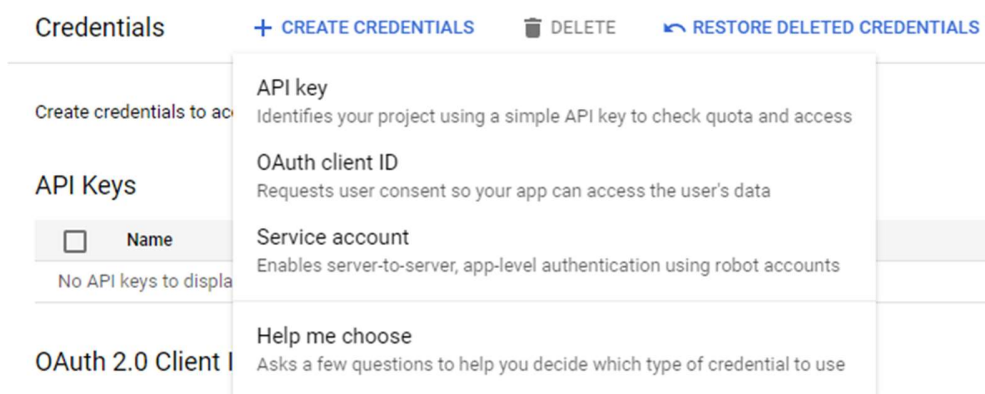


Рисунок 3.3 – Створення облікових даних

Під час створення облікових даних необхідно вказати URI адресу, на якій розміщується наш додаток та URI адресу на яку може бути перенаправлено користувача після проходження аутентифікації. Варто зазначити, що значень може бути декілька. Тобто додаток у запиті може запросити перенаправлення на різні адреси, але важливо, щоб сам додаток розташовувався за адресою, яка міститься у списку, та адреса перенаправлення була із списку дозволених. Це може бути корисним, наприклад, під час процесу розробки. Додамо в URI значення `https://localhost:5000`, оскільки саме на цьому порті ми будемо локально запускати наш проект. Другим значенням додамо URI опублікованого проекту у Microsoft Azure.

Authorized JavaScript origins ?

For use with requests from a browser

URIs 1 *

URIs 2 *

[+ ADD URI](#)

Рисунок 3.4 – Налаштування списку дозволених джерел

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

Authorized redirect URIs ?

For use with requests from a web server

URIs 1 *

https://localhost:5000/sso

URIs 2 *

https://points-webapp.azurewebsites.net/sign-in

URIs 3 *

https://points-webapp.azurewebsites.net/sign-up

+ ADD URI

Рисунок 3.5 – Налаштування списку дозволених перенаправлень

Також надається можливість налагодити екран згоди користувача, який буде відображатися під час першого входу у додаток:

Sign in with Google

1

[Display Name] wants access to your Google Account

2

Select what [Display Name] can access

3

Make sure you trust [Display Name]

Cancel

Allow

Рисунок 3.6 – Структура екрану згоди користувача

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Отже, даний екран містить наступну структуру:

1. Логотип і назва додатку

2. Адреса електронної пошти, яку користувачі можуть використовувати для зв'язку з розробником додатку, і список областей, тобто конкретних запитів додатка на доступ до облікового запису Google користувача та дані, які будуть надані.

3. Посилання на Політику конфіденційності й Умови використання програми. Цей розділ також містить:

- речення, яке описує, як користувач може керувати доступом до свого облікового запису Google;

- посилання на статтю Google, яка описує, як Google допомагає користувачам безпечно ділитися своїми даними.

Виконаємо відповідні налагодження:

App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *
Points

The name of the app asking for consent

User support email *
d.moskvin2002@gmail.com

For users to contact you with questions about their consent

App logo

This is your logo. It helps people recognize your app and is displayed on the OAuth consent screen.

After you upload a logo, you will need to submit your app for verification unless the app is configured for internal use only or has a publishing status of "Testing". [Learn more](#)



Change logo

Рисунок 3.7 – Налаштування екрану згоди користувача

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

Наступним кроком є налагодження безпосередньо областей, тобто списку даних та дій, які будуть надані та дозволені додатку:

Your non-sensitive scopes

API ↑	Scope	User-facing description	
	.. ./auth/userinfo .email	See your primary Google Account email address	
	.. ./auth/userinfo .profile	See your personal info, including any personal info you've made publicly available	
	openid	Associate you with your personal info on Google	

Your sensitive scopes

Sensitive scopes are scopes that request access to private user data.

API ↑	Scope	User-facing description
No rows to display		

Your restricted scopes

Restricted scopes are scopes that request access to highly sensitive user data.

API ↑	Scope	User-facing description
No rows to display		

Рисунок 3.8 – Налагодження областей

В scopes можна додати безліч областей від звичайних імені та фотографії до можливості відредагувати опис YouTube каналу користувача та інших дій з сервісами Google. В рамках даного проекту додамо лише базові пункти, щоб забезпечити можливість доступу до імені, пошти та іншої персональної інформації, яка нам буде необхідна для реалізації функціоналу.

		Москвін Д. В.			KIK.KP.B – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

Для повноцінного функціонування провайдеру аутентифікації потрібно опублікувати проєкт та пройти відповідні перевірки безпеки від Google. Оскільки наш проєкт є навчальним, доцільно обмежитися тестовим режимом та списком дозволених користувачів:

Test users

While publishing status is set to "Testing", only test users are able to access the app. Allowed user cap prior to app verification is 100, and is counted over the entire lifetime of the app. [Learn more](#)

[+ ADD USERS](#)

 **Filter** Enter property name or value 

User information	
d.moskvina2002@gmail.com	
kb192_mdva@student.ztu.edu.ua	

[SAVE AND CONTINUE](#)

[CANCEL](#)

Рисунок 3.9 – Конфігурація списку тестових користувачів

Тепер проєкт в Google API Console повністю готовий для роботи у тестовому режимі. У вкладці Credentials можна отримати значення, необхідні для конфігурації додатку. Серед них є Client ID, який не є секретним та Client secret, який повинен зберігатися в таємниці, адже отримавши це значення злоумисник може фальсифікувати аутентифікацію з метою отримання облікових даних або здійснення дій з сервісами Google під виглядом нашого додатку. Секретів може бути декілька (але це може спричинити ризики безпеки) та вони можуть бути відключені або видалені у разі витоку значення:

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

Additional information

Client ID	356593616449-uvn6c77356ouh2dfndtr74s2fi8uceru.apps.googleusercontent.com
Creation date	January 10, 2023 at 7:35:02 PM GMT+2

Client secrets

If you are in the process of changing client secrets, you can manually rotate them without downtime. [Learn more](#)



Having more than one secret increases security risks. Disable and delete the old secret once you have verified that your app is using the new secret instead of the old secret.

Client secret	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	
Creation date	January 10, 2023 at 7:35:02 PM GMT+2	
Status	Enabled	DISABLE

Client secret	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	
Creation date	June 3, 2023 at 2:06:17 PM GMT+3	
Status	Disabled	ENABLE

+ ADD SECRET

Рисунок 3.10 – Інформація про проект

3.2. Аутентифікація користувачів, оновлення та відклик токенів

Аутентифікація користувача передбачає отримання ідентифікаційного маркера та його перевірку. Ідентифікаційні маркери – це стандартизована функція OpenID Connect, розроблена для використання в обміні підтвердженнями ідентифікації в Інтернеті.

Найбільш часто використовувані підходи для автентифікації користувача та отримання ідентифікаційного маркера називаються «серверним» потоком і «неявним» потоком. Потік сервера дозволяє внутрішньому серверу програми перевірити особу особи за допомогою браузера або мобільного пристрою. Неявний потік використовується, коли програмі на стороні клієнта (як правило, програмі JavaScript, що працює в браузері) потрібно отримати доступ до API безпосередньо, а не через внутрішній сервер.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

В рамках даного проекту реалізуємо серверний потік для автентифікації користувача, оскільки неявний потік значно складніший через ризики безпеки під час обробки та використання токенів на стороні клієнта.

Проект нашого веб-додатку буде складатися з серверу .NET 6.0 та клієнту на React.js. Також використовується такі технології:

- GraphQL для керування ролями та дозволами, отримання та маніпуляції запитів на стороні сервера;
- Redux для керування та доступу до глобального стану додатку;
- Redux-Saga для керування сайд-ефектами та запитами на сервер зі сторони клієнта.

Відповідно до документації OpenID Connect автентифікації від Google, додаток має реалізовувати наступні кроки для автентифікації користувача:

1. Створити маркер стану для захисту від підробки.
2. Надіслати запит на автентифікацію в Google.
3. Підтвердити маркер стану захисту від підробки.
4. Обміняти code на маркер доступу та ідентифікатор.
5. Отримати інформацію про користувача з ідентифікаційного токена.
6. Аутентифікувати користувача.

Отже, реалізуємо створення маркеру для захисту від підробки. Це має бути унікальний рядок довжиною не менше 30 символів:

```
private const string PkceSessionKey = "codeVerifier";

public string GenerateCodeVerifier()
{
    var codeVerifier = GenerateRandomString();

    SetCodeVerifier(codeVerifier);

    return hashHelper.ComputeHash(codeVerifier);
}
```

Також реалізуємо методи для збереження даного маркеру у сесію, щоб мати можливість отримати його значення під час наступних запитів:

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public string? GetCodeVerifier()
    => httpContextAccessor.HttpContext?.Session.GetString(PkceSessionKey);

public void SetCodeVerifier(string codeVerifier)
    => httpContextAccessor.HttpContext?.Session.SetString(PkceSessionKey,
codeVerifier);

```

Реалізуємо метод, який створить URI автентифікації OpenID Connect, на який потрібно перенаправити користувача для автентифікації в Google:

```

public string CreateAuthorizationRequestUrl(bool isRegistration = false)
{
    var redirectUri = googleOAuthConfiguration.RedirectUri;

    if (string.IsNullOrEmpty(redirectUri))
        throw new ArgumentNullException("Google OAuth provider redirect URLs are not
configured.");

    var clientId = googleOAuthConfiguration.ClientId;
    if (string.IsNullOrEmpty(clientId))
        throw new ArgumentNullException("The ClientId credential of Google OAuth
server are not configured.");

    var codeChallenge = codeVerificationHelper.GenerateCodeVerifier();
    var queryString = new Dictionary<string, string?>
    {
        { "client_id", googleOAuthConfiguration.ClientId },
        { "redirect_uri", redirectUri },
        { "response_type", "code" },
        { "scope", "openid profile email" },
        { "code_challenge", codeChallenge },
        { "code_challenge_method", "S256" },
        { "access_type", "offline" }
    };

    return QueryHelpers.AddQueryString(AuthorizationEndpoint, queryString);
}

```

У відповідь ми отримаємо code, одноразовий код авторизації, який сервер може обміняти на маркер доступу та ідентифікатор за допомогою POST запиту на провайдер аутентифікації, тобто Google. Даний запит має містити створений раніше маркер для захисту від підробки, щоб провайдер міг перевірити його валідність:

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public async Task<TokenInfo?> ExchangeAuthorizationCode(string code, bool isRegistration
= false)
{
    var redirectUri = googleOAuthConfiguration.RedirectUri;
    var clientId = googleOAuthConfiguration.ClientId;
    var clientSecret = googleOAuthConfiguration.ClientSecret;

    if (string.IsNullOrEmpty(redirectUri) || string.IsNullOrEmpty(clientId) ||
string.IsNullOrEmpty(clientSecret))
        throw new ArgumentNullException("One or more Google OAuth provider
credentials are not configured.");

    var codeVerifier = codeVerificationHelper.GetCodeVerifier();
    if (string.IsNullOrEmpty(codeVerifier))
        return null;

    var bodyParams = new Dictionary<string, string>
    {
        { "client_id", clientId },
        { "client_secret", clientSecret },
        { "code", code },
        { "grant_type", "authorization_code" },
        { "redirect_uri", redirectUri },
        { "code_verifier", codeVerifier }
    };

    return await httpRequestHelper.SendPostRequest<TokenInfo>(TokenEndpoint,
bodyParams);
}

```

У відповідь на запит заміни коду ми отримаємо JSON зі значеннями id_token, access_token, expires_in, refresh_token, scope та token_type. Значення access_token нам потрібне для роботи з Google API, наприклад, змінити опис каналу на YouTube, отримати інформацію профілю та безліч інших дій з продуктами Google, на які погодився користувач на екрані згоди. Ми налаштували лише отримання даних профіля, тому реалізуємо саме цю дію:

```

public async Task<UserInfo?> GetUserInfo(string accessToken)
{
    var queryParams = new Dictionary<string, string?>
    {
        { "access_token", accessToken }
    };
}

```

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

```
return await httpRequestHelper.SendGetRequest<UserInfo>(UserInfoEndpoint,
queryParams);
}
```

Значення `id_token` ми будемо використовувати для аутентифікації, але він має обмежений термін дії (як і `access_token`) і для його оновлення потрібно відправити запит з відповідними параметрами:

```
public async Task<TokenInfo?> RefreshToken(string refreshToken)
{
    if (string.IsNullOrEmpty(googleOAuthConfiguration.ClientId) ||
        string.IsNullOrEmpty(googleOAuthConfiguration.ClientSecret))
        throw new ArgumentNullException("Client credentials of Google OAuth server
are not configured.");

    var bodyParams = new Dictionary<string, string>
    {
        { "client_id", googleOAuthConfiguration.ClientId },
        { "client_secret", googleOAuthConfiguration.ClientSecret },
        { "grant_type", "refresh_token" },
        { "refresh_token", refreshToken }
    };

    return await httpRequestHelper.SendPostRequest<TokenInfo>(TokenEndpoint,
bodyParams);
}
```

У відповідь ми отримаємо нову пару `id_token` та `access_token`. З `access_token` ми не можемо отримати жодної інформації, він підписаний секретним ключем і використовується лише Google API, а `id_token` має публічну частину, з якої ми можемо отримати ідентифікатор та інформацію користувача, термін дії унікальний ідентифікатор токена на інші поля, які використовуються для валідації.

На даному етапі можна вважати процес аутентифікації завершеним. У нас є все, щоб, отримавши інформацію про користувача з токена, зробити запит до бази даних користувачів програми. Якщо користувач уже існує у нашій базі даних, ми повинні почати сеанс програми для цього користувача, якщо відповідь Google API відповідає всім вимогам входу.

Може виникнути ситуація, коли користувач припинить взаємодію з нашим додатком та захоче припинити доступ до його профілю. Для цього він може зробити

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

це вручну у налаштуваннях свого профіля Google, але розробникам додатку варто автоматизувати даний процес:

```
public async Task<bool> RevokeToken(string token)
{
    var queryParams = new Dictionary<string, string?>
    {
        { "token", token }
    };

    var response = await
    httpRequestHelper.SendHttpRequestAsync<object>(HttpMethod.Post, RevocationEndpoint,
    queryParams: queryParams);
    return response != null;
}
```

Після виконання такого запиту, всі id_token, access_token та refresh_token стають недійсними.

3.3. Перевірка працездатності підсистеми аутентифікації

Перевіримо роботу нашого функціоналу на практиці. Отримаємо URI аутентифікації:

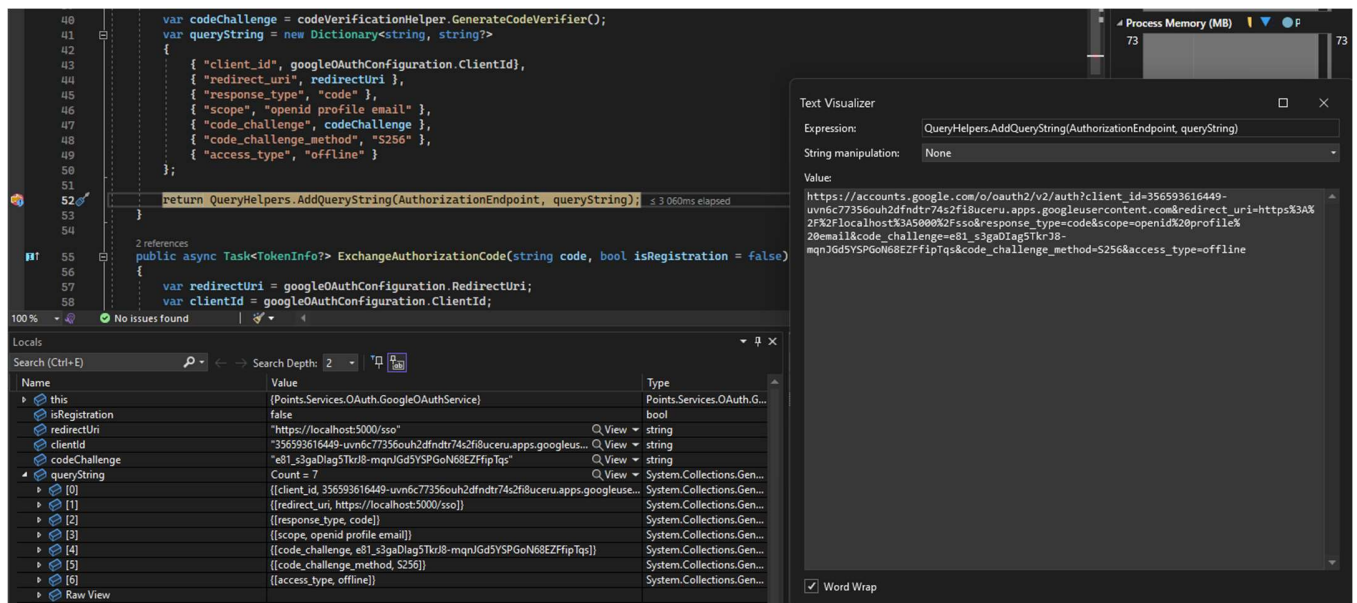


Рисунок 3.11 – Отриманий URI аутентифікації

Отже, був сформований URI з необхідними параметрами та маркер для захисту від підробки yJWzqbU7hTlyJ9I0c0Esn9VQ-9n1wegSpjgjfWyBOjA.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

Оберемо наш обліковий запис:

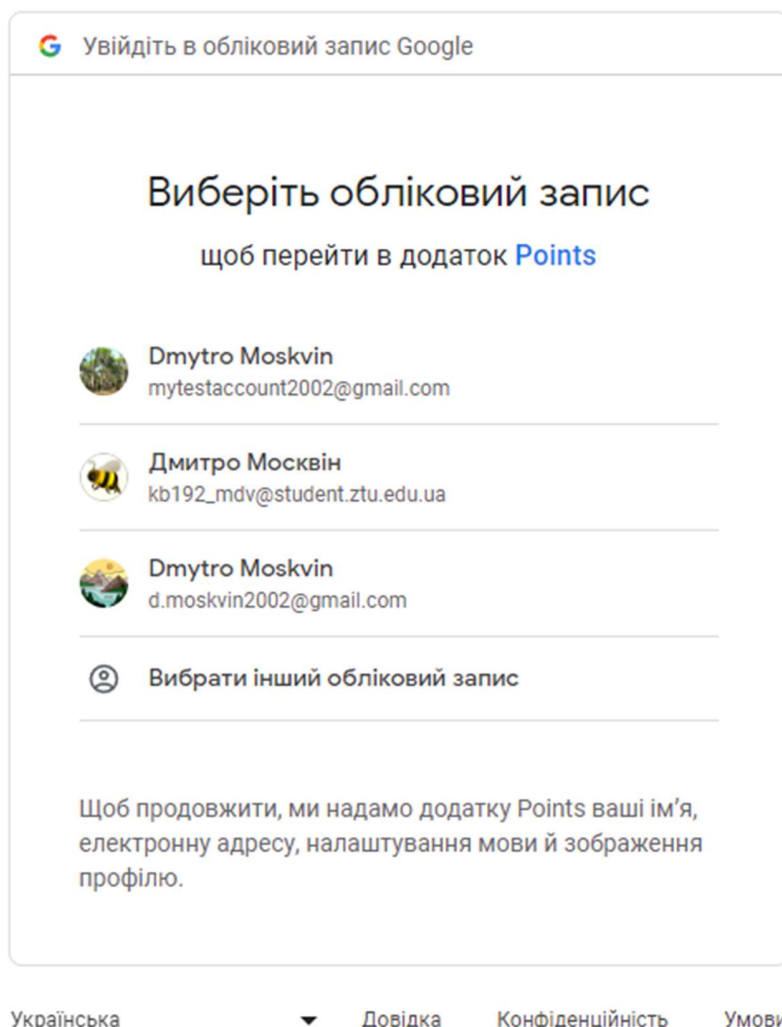


Рисунок 3.12 – Екран згоди користувача

Після підтвердження нас буде перенаправлено на redirect URI, який ми вказали у параметрах запиту, з кодом та іншою інформацією у параметрах посилання:

<https://localhost:5000/sign-in?code=4/0AWtgzh4e7-bgycyTetNaExcMzQMDSaldAG161ktfbBoZQYRYXALZ-BaYXMOMraHKtXaR2g&scope=email%20profile%20openid%20https://www.googleapis.com/auth/userinfo.profile%20https://www.googleapis.com/auth/userinfo.email&authuser=1&hd=student.ztu.edu.ua&prompt=none>.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

Обмінємо отриманий код на токени:

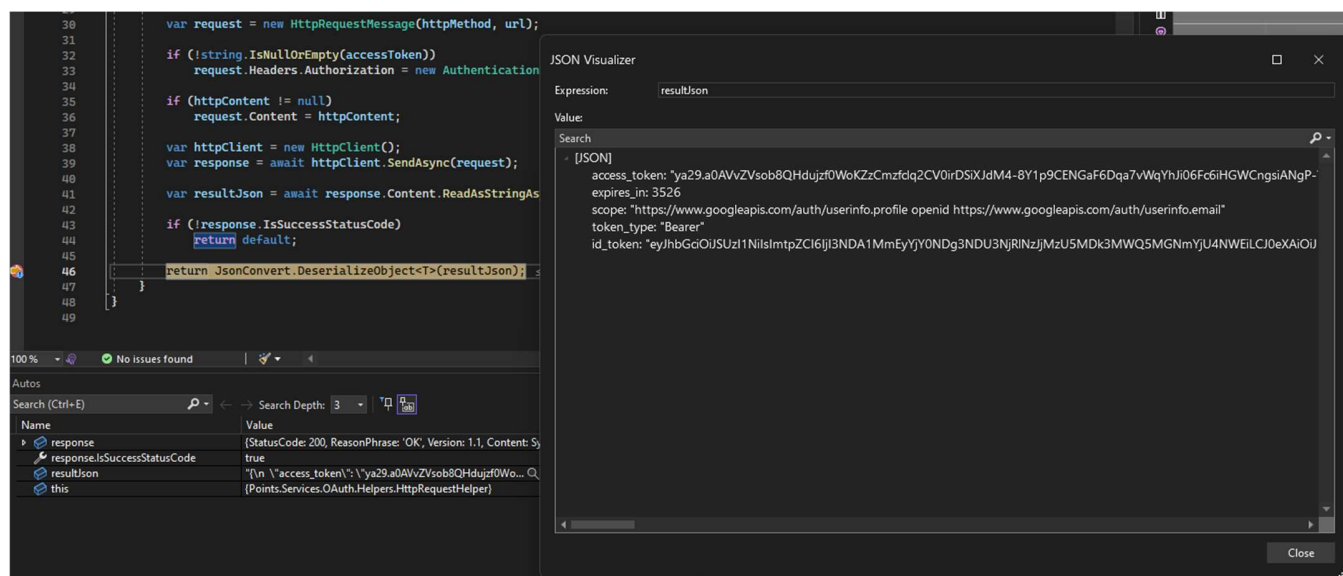


Рисунок 3.13 – Отримані токени в обмін на код

Тепер спробуємо отримати дані з Google API за допомогою токену доступу:

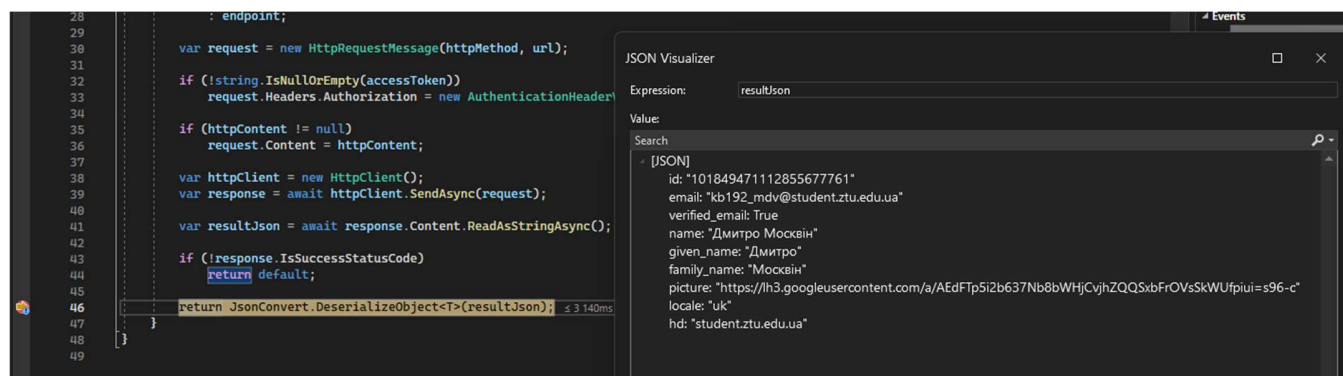


Рисунок 3.14 – Отримані дані профіля з Google API

Оновимо отримані токени:

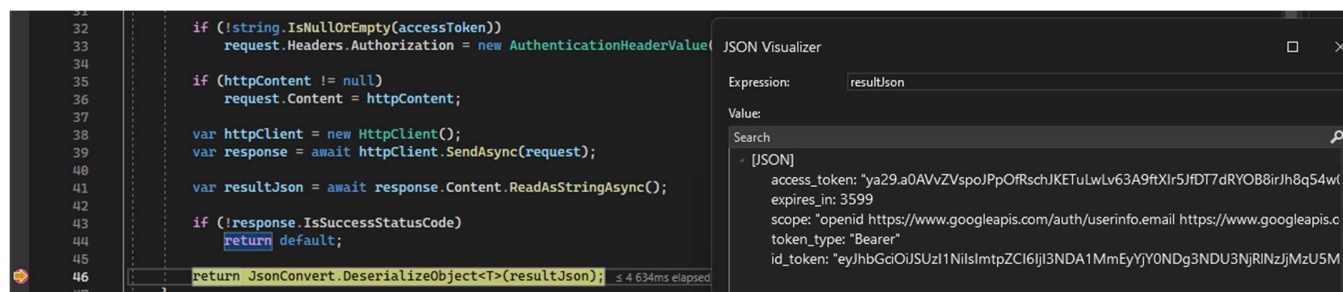


Рисунок 3.15 – Результат оновлення токенів

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

Отже, ми отримали новий токен доступу та ідентифікації, а також нове значення терміну їх дії. Варто зазначити, що у даному списку немає refresh токена, оскільки він видається лише від час аутентифікації та має вічний термін дії.

Перейдемо у налаштування облікового запису Google та переглянемо сервіси, яким ми надали доступ:

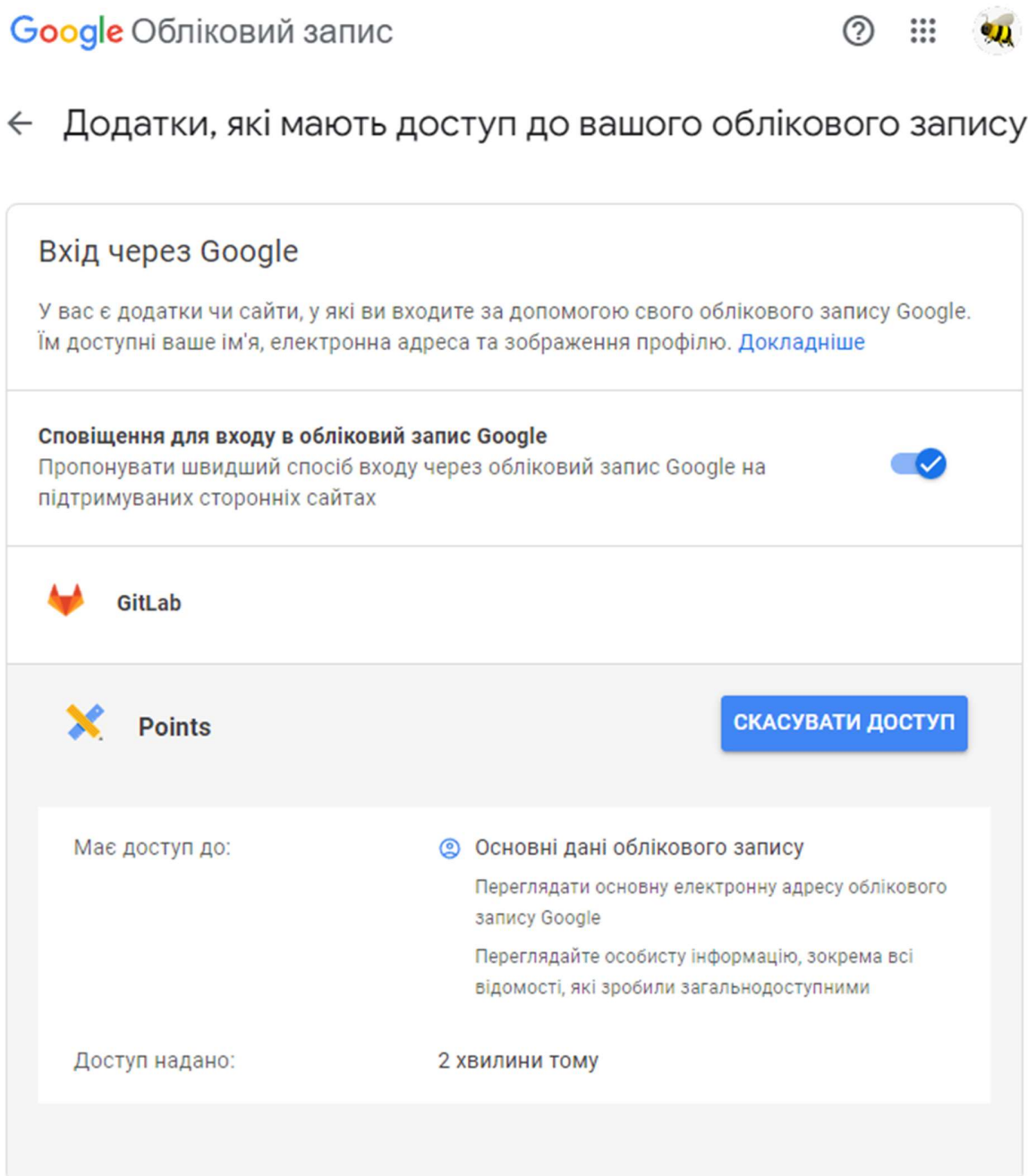


Рисунок 3.17 – Список додатків, які мають доступ до облікового запису

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

Бачимо у списку наш додаток. Надішлемо запит на відміну доступу та перевіримо зміни:

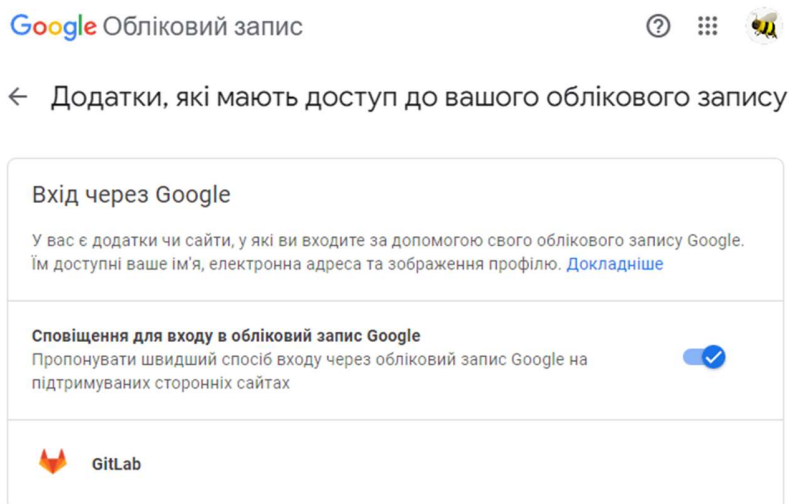


Рисунок 3.18 – Список додатків, які мають доступ до облікового запису

Отже, ми успішно відмінили доступ нашого додатку до облікового запису користувача.

Спробуємо відмінити доступ вручну та оновити токени:

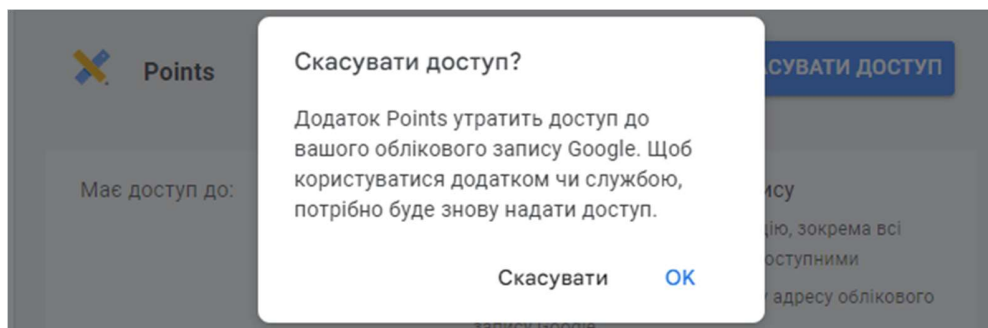


Рисунок 3.19 – Скасування доступу



Рисунок 3.20 – Результат спроби оновити токени

Отже, ми успішно пройшли повний шлях, описаний в документації Google[2] та можемо зробити висновок, що поставлене завдання виконано.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

3.4. Проведення атак та запровадження захисту

3.4.1. HTML Injection атака

Дана атака полягає в впровадженні HTML-коду через уразливі частини веб-сайту. Зловмисник надсилає HTML-код через будь-яке вразливе поле з метою змінити дизайн веб-сайту або будь-яку інформацію, яка відображається користувачеві.

Ін'єкційну атаку можна виконати з двома різними цілями:

- змінити зовнішній вигляд відображеного веб-сайту;
- викрасти особисті дані.

Крім того, цю ін'єкційну атаку можна здійснити через різні частини веб-сайту, тобто поля введення даних і посилання на веб-сайт.

Оскільки наш додаток є SPA та побудований на описаному раніше стеку технологій, виконання атаки через посилання на веб-сайт не можуть бути успішними, адже параметри з URL адреси використовується лише для побудови запиту на сервер, де даний код не може бути виконаний. Відображення значень параметрів на сторінці є практикою додатків, які побудовані на технологіях подібним MVC, наш підхід передбачає отримання даних окремими запитами.

Спробуємо провести HTML Injection атаку шляхом введення HTML коду в текстове поле імені.

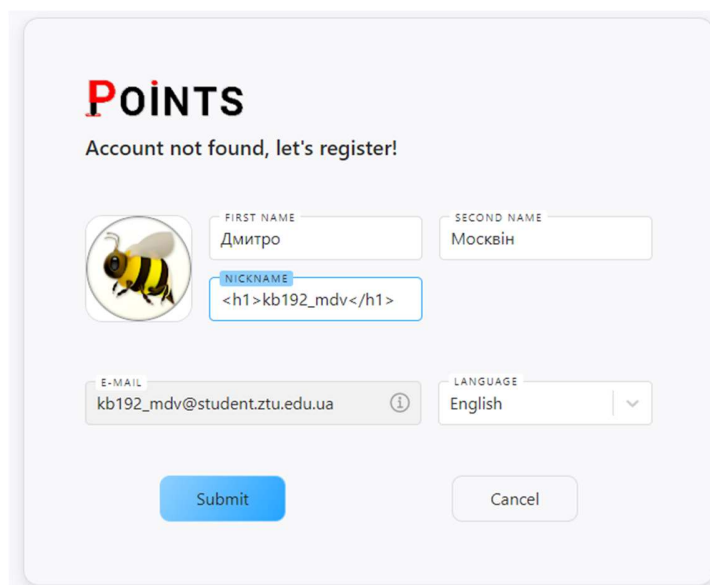


Рисунок 3.21 – Введення тексту з використання тегів HTML

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

Після підтвердження форми, наш тег буде відправлено на сервер у вигляді рядку «<h1>kb192_mdv</h1>» та збережено в базі даних

ID	NickName	FirstName	SecondName
101849471112855677761	kb192_mdv	Дмитро	Москвін

Рисунок 3.22 – Збережене в базі даних значення

Але таке збереження є допустимим, адже HTML код може бути задіяно та відображено лише на сторінці браузера, в усіх інших випадках він буде звичайним рядком, навіть під час запитів від клієнту:

```
firstName: "Дмитро"
id: "101849471112855677761"
nickName: "<h1>kb192_mdv</h1>"
```

Рисунок 3.23 – Отримане значення у відповіді від сервера

Розглянемо будь-який елемент на сторінці, що містить наш HTML код:



Рисунок 3.24 – Відображення значення на сторінці

Отже, дані відображається у вигляді звичайного тексту, а не HTML коду, ми можемо це перевірити за допомогою детального перегляду розмітки сторінки:

```
<div class="profile-info"> flex
  <h2> ... </h2>
  "<h1>kb192_mdv</h1>" == $0
</div>
```

Рисунок 3.25 – Отримане значення у розмітці сторінки

```
<div class="profile-info"> flex
  <h2> ... </h2>
  &lt;h1&gt;kb192_mdv&lt;/h1&gt;
</div>
```

Рисунок 3.26 – Отримане значення у відповіді від сервера (редагування)

Отже, бачимо, що HTML теги було закодовано та перетворено у звичайні символи, які відображаються виключно як текст. Така поведінка пояснюється особливостями React: за замовчуванням всі значення конвертуються на рендеряться як звичайний текст. Для можливості відобразити такий текст з HTML тегами передбачено

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

окрему властивість «dangerouslySetInnerHTML», у яку потрібно передати таке значення:

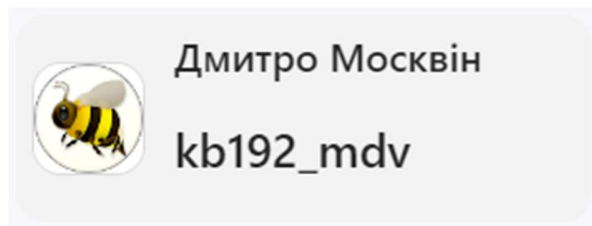


Рисунок 3.27 – Відображення значення на сторінці

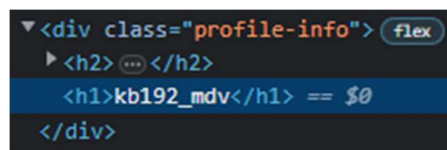


Рисунок 3.28 – Отримане значення у розмітці сторінки

Отже, на цей раз значення було відображено як DOM елемент. Отже, в даному випадку нам не потрібно реалізовувати додатковий захист від HTML-атак, адже він вбудований в стек технологій, які ми використовуємо. Варто зауважити, що вищерозглянуту властивість варто використовувати лише в окремих випадках коли ми впевнені в відсутності загроз, адже це може стати вразливістю для HTML Injection атак. Також у такому значенні можуть бути не лише теги, які змінюють стиль відображення на сторінці, а і повноцінний JS скрипт, який може виконати несанкціоновані дії з особистими даними користувача. Така атака вже буде мати набагато серйозніші наслідки.

Безсумнівно, основною причиною даної атаки є неуважність і необізнаність розробника. Цей тип ін'єкційної атаки відбувається, коли вхід і вихід не перевірені належним чином. Тому основним правилом запобігання атаці HTML є відповідна перевірка даних.

У випадку відсутності відповідного захисту на рівні вбудованого інструментарію, кожен вхід слід перевіряти, чи містить він будь-який скрипт або будь-який HTML-код. Зазвичай перевіряється, чи містить код якийсь спеціальний скрипт або HTML-дужки – `<script></script>`, `<html></html>`.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

3.4.2. JavaScript Injection атака

Окрім полів форм та адресного рядку існує консоль розробника, яка часто стає причиною несправностей внаслідок цікавості чи довірливості користувачів. Часто шахраї просять користувачів зробити якісь дії в консолі розробника, найпростіше – вставити в консоль скрипт та виконати його.

Змоделюємо подібну атаку, яка буде спрямована на отримання ідентифікаційного токена, отримання якого надасть нам змогу на деякий час отримати повний доступ до облікового запису жертви та виконувати дії від її особи. Токен зберігається у localStorage користувача задля забезпечення ідентифікації користувача та підтримки його сесії після оновлення сторінки або повернення після тривалої перерви. Доступ до цього сховища може отримати лише безпосередньо клієнт додатку. Переглянемо значення за допомогою консолі розробника:

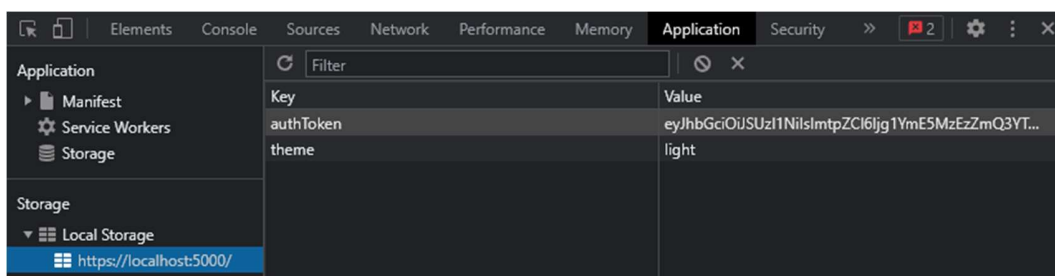


Рисунок 3.29 –Значення auth токена в localStorage

Для проведення атаки знадобиться будь-який ресурс, який зможе прийняти запит від клієнта жертви, нехай це буде простий веб-додаток. Запустимо в консолі розробника скрипт, який отримує з localStorage значення «authToken» та відправляє його на додаток-приймач, який знаходиться за іншою адресою «https://localhost:7168»:

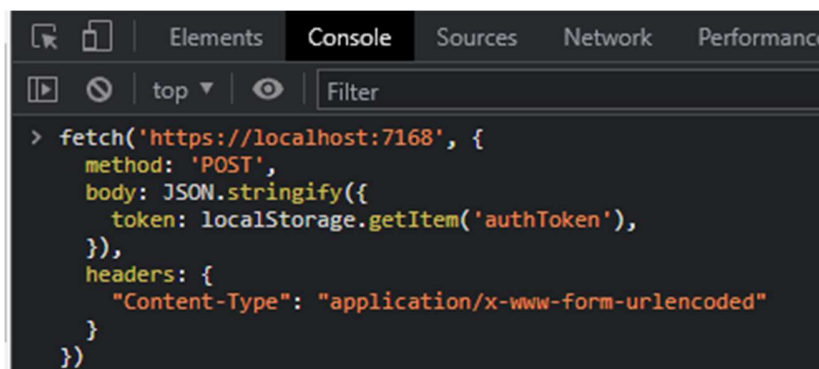


Рисунок 3.30 – Запуск JS-скрипту в консолі розробника

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

Переглянемо запит, який прийшов на наш додаток-приймач:

```

19 [HttpPost]
20 public IActionResult Index()
21 {
22     var bodyValue = Request.Form.Keys.FirstOrDefault();
23     if (string.IsNullOrEmpty(bodyValue))
24         return View();
25
26     var data = JsonConvert.DeserializeObject<TokenData>(bodyValue);
27     return View();
28 }
29
30
31 private class TokenData
32 {
33     [JsonProperty("token")]
34     public string? Token { get; set; }
35 }
36

```

Autos

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
bodyValue	"{"token": "eyJhbGciOiJIUzI1NiIsImtpZCI6Ijg1YmE5MzEzZmQ3YT..."	string
data	{Getter.Controllers.HomeController.TokenData}	Getter.Controllers.Hom...
Token	"eyJhbGciOiJIUzI1NiIsImtpZCI6Ijg1YmE5MzEzZmQ3YTdkNGFmYTg..."	string
this	{Getter.Controllers.HomeController}	Getter.Controllers.Hom...

Рисунок 3.31 – Отримане значення auth токена

Отже, ми успішно отримали значення токена. Таким чином зломисник може отримати будь-які дані жертви включаючи особисту інформацію та конфіденційні дані.

Якщо у випадку з полями форм ми могли уникати ризиків безпеки шляхом перетворення HTML розмітки, яка також може містити у собі подібний скрипт, у звичайний текст, то в даному випадку ми не зможемо заблокувати консоль розробника, адже це досить потужний інструмент для діагностики, подальшого розвитку та підтримки програмного забезпечення.

Вирішенням даної проблеми є додавання спеціального сповіщення в консоль, яке буде попереджувати користувача про ризики будь-яких дій. Такий підхід можна зустріти на Facebook та інших веб-додатках:

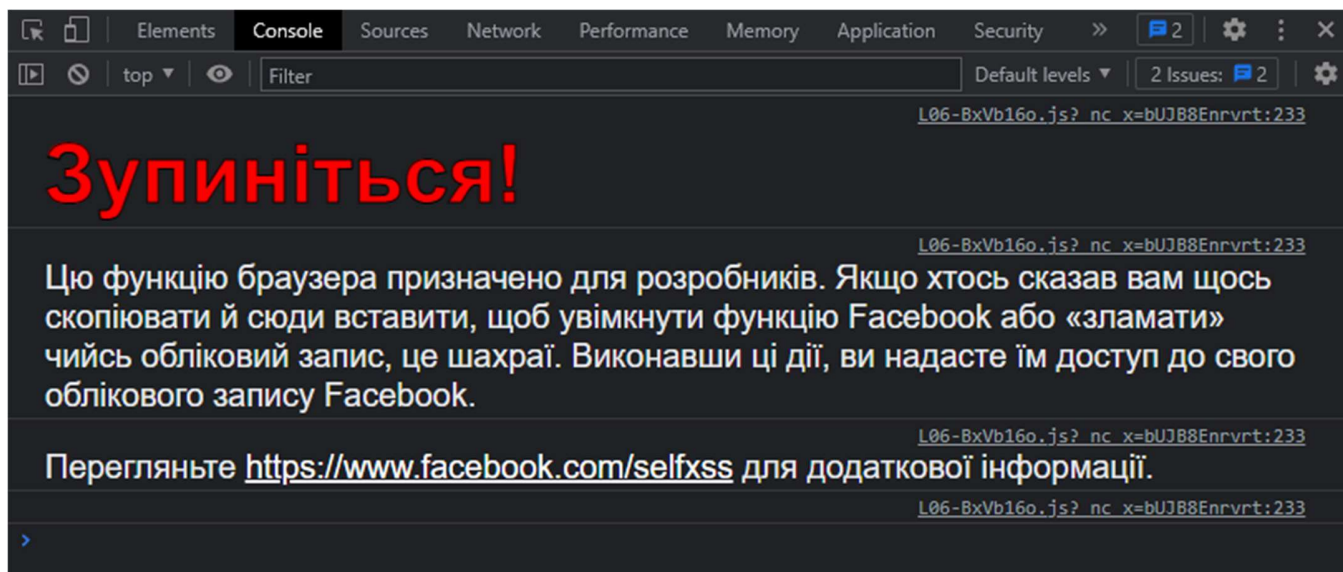


Рисунок 3.32 – Застереження під час відкриття консолі розробника на Facebook

Реалізуємо подібне сповіщення у нашому веб-додатку:

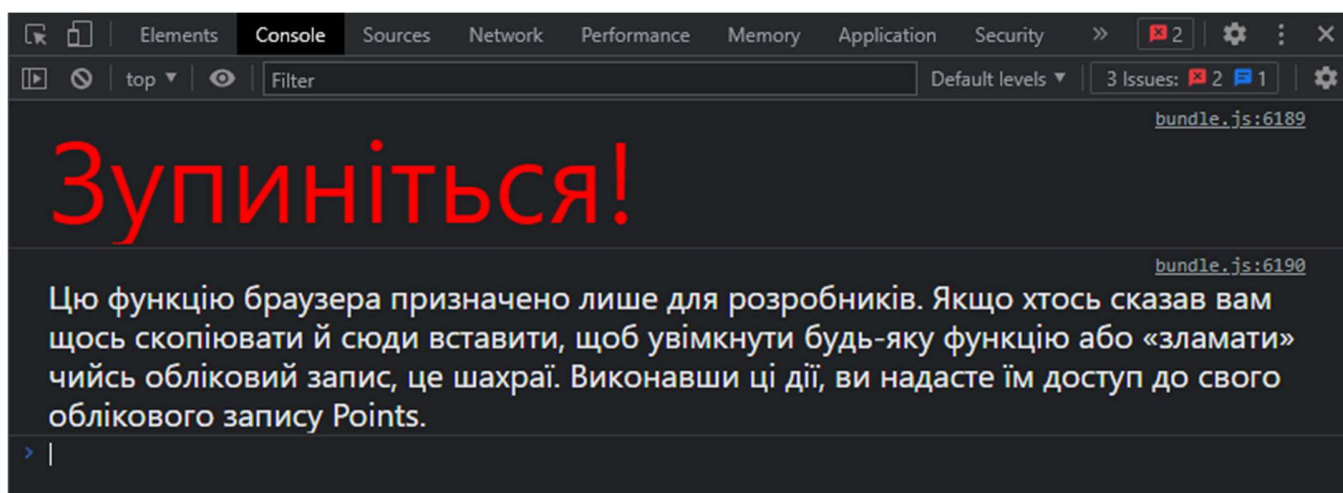


Рисунок 3.33 – Реалізоване застереження під час відкриття консолі розробника

3.4.3. Уособлення облікового запису

Отже, завдяки попередній атаці ми отримали auth токен іншого користувача, спробуємо використати цей токен та обійти систему аутентифікації.

Перейдемо в інший браузер або анонімну вкладку та для зручності підміни значень увійдемо в інший обліковий запис:

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

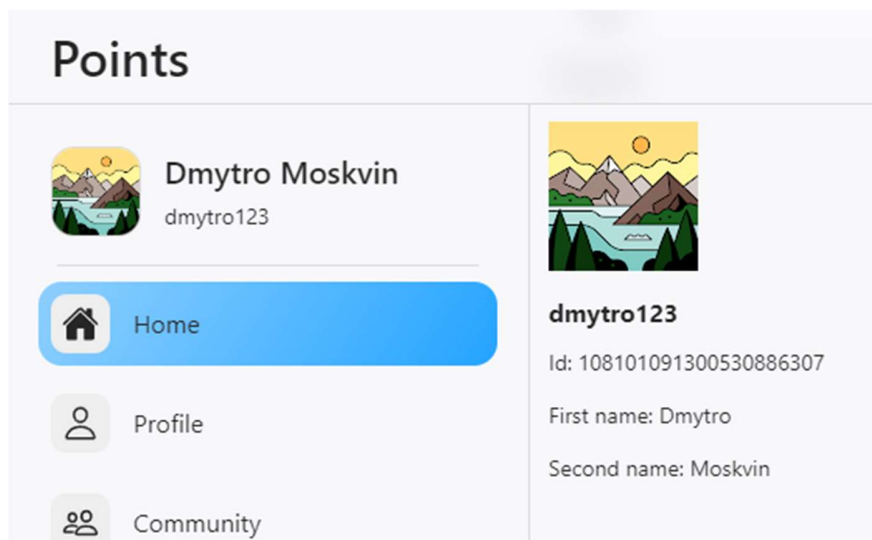


Рисунок 3.34 – Обліковий запис умовного зловмисника

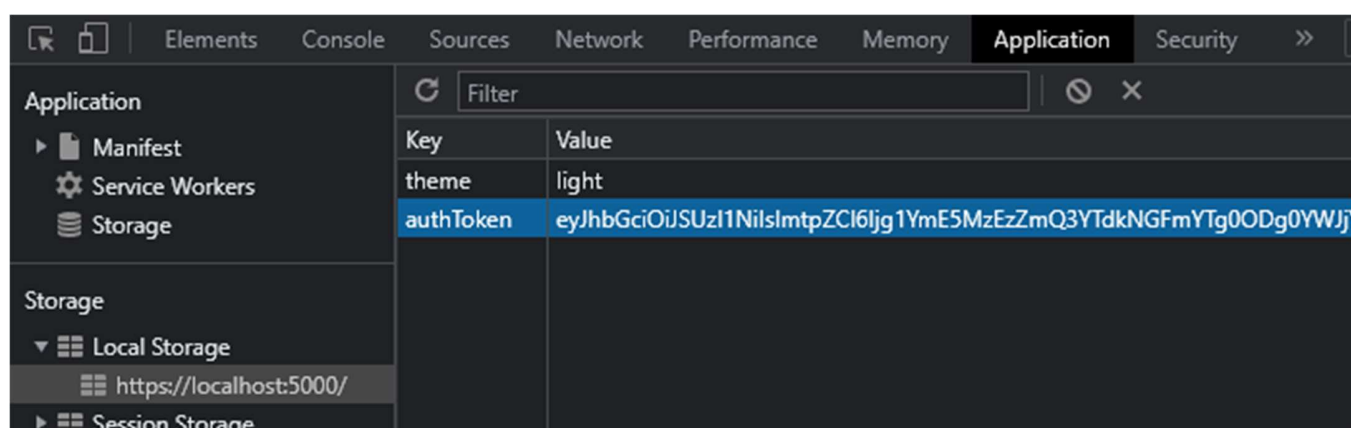


Рисунок 3.35 – Значення auth токена

Підмінімо значення токеном облікового запису жертви та оновимо сторінку:

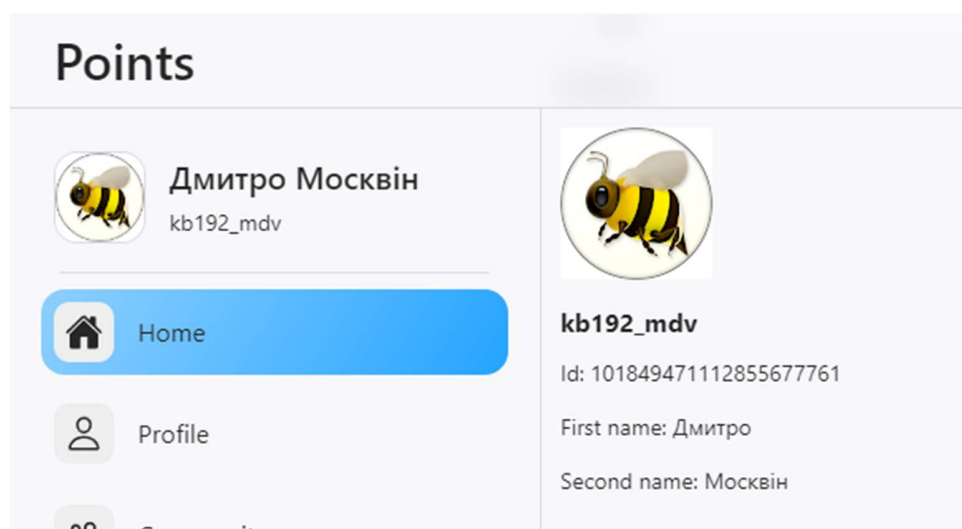


Рисунок 3.36 – Успішне уособлення жертви

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

Отже, ми успішно отримали доступ до облікового запису жертви і можемо повноцінно їм користуватися допоки не застаріє вкрадений токен (зазвичай токен дійсний близько години).

За замовчуванням маркери доступу OAuth є так званими маркерами-носіями, тобто вони не прив'язані до клієнта, і кожен, хто володіє токеном, може ним користуватися.

Токени підтвердження володіння (Proof-of-Possession) прив'язані до клієнта, який запитав токен. Якщо цей маркер витікає, його не зможе використати ніхто інший. На жаль, даний захист поки не є загальноприйнятим та поки не підтримується Google, хоча в Oka вже запровадили таку систему. Саме тому на даний момент так важливо перекрити всі шляхи за допомогою яких токен може бути вкрадений.

Висновки до третього розділу

В даному розділі виконано налагодження проекту в Google API Console, отримано credentials, які необхідні для встановлення підключення між додатком та identity провайдером. Відповідно до рекомендацій та вимог документації Google, було реалізовано процес аутентифікації, оновлення та відзиву токенів. Протестовано роботу реалізованої логіки та детально проаналізовано кожен крок алгоритму та відповідно отримані дані. Розглянуто та проведено найпоширеніші атаки, які спрямовані на крадіжку облікових даних та конфіденційної інформації, зокрема auth токену, та, врешті-решт, розглянуто наслідки та можливості, які може отримати зловмисник у разі успіху. В результаті аналізу було внесено програмні правки, які націлені на зниження ризику безпеки.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

ВИСНОВКИ

Було розглянуто теоретичні відомості та визначено популярні методи аутентифікації, зокрема за допомогою технології єдиного входу. Було визначено відмінності між усіма методами та їх особливості. Докладно розглянуто принцип роботи OAuth 2.0 та поняття JWT маркерів. В результаті проведеного аналізу можна зробити висновок, що в рамках даного проекту варто реалізувати систему аутентифікації на основі протоколу OpenID Connect та потоку аутентифікації, який потребує серверну частину.

Розглянуто реалізації провайдерів аутентифікації від відомих організацій, таких як Apple, Google та Microsoft, було визначено критерії вибору та, відповідно до них, обрано варіант від Google, адже він є найбільш зручним та доступним завдяки мультиплатформенності та поширеності продукції Google. Було покроково розглянуто алгоритм аутентифікації користувачів за допомогою Google SSO та вимоги до валідації токенів. Відповідно до технічних вимог підсистеми аутентифікації було спроектовано проект веб-додатку соціальної мережі з використанням чотирирівневої архітектури, що забезпечить перспективу розширення функціоналу у майбутньому, а також можливість замінити один із рівнів на будь-який аналог. Визначено стек технологій, які будуть використані для реалізації даної підсистеми, а саме: React, Redux, Redux-Saga на клієнтській частині та .NET 6, GraphQL, Entity Framework Core і Microsoft SQL Server на стороні сервера.

Виконано налагодження проекту в Google API Console, отримано credentials, які необхідні для встановлення підключення між додатком та identity провайдером. Відповідно до рекомендацій та вимог документації Google, було реалізовано процес аутентифікації, оновлення та відзиву токенів. Протестовано роботу реалізованої логіки та детально проаналізовано кожен крок алгоритму та відповідно отримані дані. Розглянуто та проведено найпоширеніші атаки, які спрямовані на крадіжку облікових даних та конфіденційної інформації, зокрема auth токену, та, врешті-решт, розглянуто наслідки та можливості, які може отримати зловмисник у разі успіху. В результаті аналізу було внесено програмні правки, які націлені на зниження ризику безпеки.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

Реалізована система закликає бути потужною основою для подальшого розвитку соціальної мережі, адже запроваджує високий рівень безпеки та можливостей в плані розробки завдяки сучасному стеку технологій, гнучкій архітектурі та ряду можливостей, які закладені в основу проекту, серед яких система збереження стану вкладок, асинхронне завантаження даних та відображення розмітки сторінки, динамічне підвантаження текстів інтерфейсу відповідно до обраної мови, особливості проектування бази даних, що дають можливість запровадження контролю сесій з боку та багато інших функцій.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What is Authentication? Definition and uses - Auth0 [Електронний ресурс] – Режим доступу до ресурсу: <https://auth0.com/intro-to-iam/what-is-authentication>.
2. OpenID Connect [Електронний ресурс] / Google – Режим доступу до ресурсу: <https://developers.google.com/identity/openid-connect/openid-connect>.
3. Single sign-on | Identity and access management [Електронний ресурс] – Режим доступу до ресурсу: <https://cloud.google.com/architecture/identity/single-sign-on>.
4. What Is Single Sign-On Authentication (SSO) [Електронний ресурс] – Режим доступу до ресурсу: <https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/>.
5. How Does Single Sign-On (SSO) Work? [Електронний ресурс] – Режим доступу до ресурсу: <https://frontegg.com/guides/single-sign-on-ssso>.
6. OpenID Connect Core 1.0 incorporating errata set 1 [Електронний ресурс] – Режим доступу до ресурсу: https://openid.net/specs/openid-connect-core-1_0.html.
7. Single Sign-On - Auth0 [Електронний ресурс] – Режим доступу до ресурсу: <https://auth0.com/docs/authenticate/single-sign-on>.
8. Intro to single sign-on with Apple devices [Електронний ресурс] – Режим доступу до ресурсу: <https://support.apple.com/guide/deployment/intro-to-single-sign-on-depfdbf18f55/web>.
9. Identity - Google for Developers [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.google.com/identity>.
10. OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ietf.org/archive/id/draft-ietf-oauth-dpop-04.html>.
11. OpenID Connect flows - IBM Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ibm.com/docs/en/sva/9.0?topic=federations-openid-connect-flows>.
12. What is Authentication? - TechTarget [Електронний ресурс] – Режим доступу до ресурсу: <https://www.techtarget.com/searchsecurity/definition/authentication>.

		Москвін Д. В.			<i>КІК.КР.Б – 125 – 23 – ПЗ</i>	Арк.
		Байлюк Є.М.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

13. How Does Single Sign-On (SSO) Work? | OneLogin [Електронний ресурс] – Режим доступу до ресурсу: <https://www.onelogin.com/learn/how-single-sign-on-works>.
14. What is a JWT? Understanding JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://supertokens.com/blog/what-is-jwt>.
15. OAuth 2.0 [Електронний ресурс] – Режим доступу до ресурсу: <https://oauth.net/2/>.
16. OAuth 2.0 Grant Types [Електронний ресурс] – Режим доступу до ресурсу: <https://oauth.net/2/grant-types/>.
17. OpenID Connect Protocol - Auth0 [Електронний ресурс] – Режим доступу до ресурсу: <https://auth0.com/docs/authenticate/protocols/openid-connect-protocol>.
18. What is Okta and What Does Okta Do? [Електронний ресурс] – Режим доступу до ресурсу: https://support.okta.com/help/s/article/what-is-okta?language=en_US.
19. Azure AD Connect: Seamless single sign-on - Microsoft Entra [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/azure/active-directory/hybrid/connect/how-to-connect-sso>.
20. OpenID Connect Discovery 1.0 incorporating errata set 1 [Електронний ресурс] – Режим доступу до ресурсу: https://openid.net/specs/openid-connect-discovery-1_0.html.
21. Research and Application of the Four-tier Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://www.atlantispress.com/article/8022.pdf>.
22. GraphQL | A query language for your API [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql.org/>.
23. Overview of Entity Framework Core - EF Core - Microsoft Learn [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/>.
24. What Is an ORM? How Does It Work? How Should We Use One? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.baeldung.com/cs/object-relational-mapping>.
25. OAuth 2.0 Roles - Jenkov.com [Електронний ресурс] – Режим доступу до ресурсу: <https://jenkov.com/tutorials/oauth2/roles.html>.

		Москвін Д. В.			КІК.КР.Б – 125 – 23 – ПЗ	Арк.
		Байлюк Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

ДОДАТКИ

Програмний код основних сервісів додатку

Лістинг класу GoogleOAuthService:

```
namespace Points.Services.OAuth;

public class GoogleOAuthService : IOAuthService
{
    public readonly GoogleOAuthConfiguration;
    public readonly CodeVerificationHelper;
    public readonly HttpRequestHelper;

    public GoogleOAuthService(
        IOptions<GoogleOAuthConfiguration> googleOAuthConfiguration,
        CodeVerificationHelper,
        HttpRequestHelper httpRequestHelper)
    {
        this.googleOAuthConfiguration = googleOAuthConfiguration.Value;
        this.codeVerificationHelper = codeVerificationHelper;
        this.httpRequestHelper = httpRequestHelper;
    }

    public string CreateAuthorizationRequestUrl(bool isRegistration = false)
    {
        var redirectUri = googleOAuthConfiguration.RedirectUri;

        if (string.IsNullOrEmpty(redirectUri))
            throw new ArgumentNullException("Google OAuth provider redirect URLs are not configured.");

        var clientId = googleOAuthConfiguration.ClientId;
        if (string.IsNullOrEmpty(clientId))
            throw new ArgumentNullException("The ClientId credential of Google OAuth server are not configured.");

        var codeChallenge = codeVerificationHelper.GenerateCodeVerifier();
        var queryString = new Dictionary<string, string?>
        {
            { "client_id", googleOAuthConfiguration.ClientId },
            { "redirect_uri", redirectUri },
            { "response_type", "code" },
            { "scope", "openid profile email" },
            { "code_challenge", codeChallenge },
            { "code_challenge_method", "S256" },
            { "access_type", "offline" }
        };

        return QueryHelpers.AddQueryString(AuthorizationEndpoint, queryString);
    }
}
```

```

public async Task<TokenInfo?> ExchangeAuthorizationCode(string code, bool isRegistration = false)
{
    var redirectUri = googleOAuthConfiguration.RedirectUri;
    var clientId = googleOAuthConfiguration.ClientId;
    var clientSecret = googleOAuthConfiguration.ClientSecret;

    if (string.IsNullOrEmpty(redirectUri) || string.IsNullOrEmpty(clientId) ||
string.IsNullOrEmpty(clientSecret))
        throw new ArgumentNullException("One or more Google OAuth provider credentials are not configured.");

    var codeVerifier = codeVerificationHelper.GetCodeVerifier();
    if (string.IsNullOrEmpty(codeVerifier))
        return null;

    var bodyParams = new Dictionary<string, string>
    {
        { "client_id", clientId },
        { "client_secret", clientSecret },
        { "code", code },
        { "grant_type", "authorization_code" },
        { "redirect_uri", redirectUri },
        { "code_verifier", codeVerifier }
    };

    return await httpRequestHelper.SendPostRequest<TokenInfo>(TokenEndpoint, bodyParams);
}

public async Task<TokenInfo?> RefreshToken(string refreshToken)
{
    if (string.IsNullOrEmpty(googleOAuthConfiguration.ClientId) ||
string.IsNullOrEmpty(googleOAuthConfiguration.ClientSecret))
        throw new ArgumentNullException("Client credentials of Google OAuth server are not configured.");

    var bodyParams = new Dictionary<string, string>
    {
        { "client_id", googleOAuthConfiguration.ClientId },
        { "client_secret", googleOAuthConfiguration.ClientSecret },
        { "grant_type", "refresh_token" },
        { "refresh_token", refreshToken }
    };

    return await httpRequestHelper.SendPostRequest<TokenInfo>(TokenEndpoint, bodyParams);
}

public async Task<UserInfo?> GetUserInfo(string accessToken)
{
    var queryParams = new Dictionary<string, string?>
    {
        { "access_token", accessToken }
    }
}

```

```

    };

    return await httpRequestHelper.SendGetRequest<UserInfo>(UserInfoEndpoint, queryParams);
}

public async Task<bool> RevokeToken(string token)
{
    var queryParams = new Dictionary<string, string?>
    {
        { "token", token }
    };

    var response = await httpRequestHelper.SendHttpRequestAsync<object>(HttpMethod.Post, RevocationEndpoint,
queryParams: queryParams);
    return response != null;
}
}

```

Лістинг класу GoogleTokenValidator:

```

namespace Points.Services.OAuth;

public class GoogleTokenValidator : ISecurityTokenValidator
{
    private readonly JwtSecurityTokenHandler tokenHandler;
    private readonly string defaultScheme;
    private readonly string? audience;

    public bool CanValidateToken => true;

    public int MaximumTokenSizeInBytes { get; set; } = TokenValidationParameters.DefaultMaximumTokenSizeInBytes;

    public GoogleTokenValidator(string defaultScheme, string? audience)
    {
        tokenHandler = new JwtSecurityTokenHandler();
        this.defaultScheme = defaultScheme;
        this.audience = audience;
    }

    public bool CanReadToken(string securityToken) => tokenHandler.CanReadToken(securityToken);

    public ClaimsPrincipal? ValidateToken(string securityToken, TokenValidationParameters validationParameters, out
SecurityToken? validatedToken)
    {
        var validationSettings = new ValidationSettings
        {
            Audience = new[] { audience },
        };
    }
}

```

```

try
{
    var payload = ValidateAsync(securityToken, validationSettings).Result;
    var tokenInfo = tokenHandler.ReadJwtToken(securityToken);

    var claims = new Claim[]
    {
        new Claim(JwtRegisteredClaimNames.Email, payload.Email),
        new Claim(JwtRegisteredClaimNames.Sub, payload.Subject),
        new Claim(JwtRegisteredClaimNames.Iss, payload.Issuer),
        tokenInfo.Claims.First(claim => claim.Type == JwtRegisteredClaimNames.AtHash),
    };

    validatedToken = tokenInfo;
    return new ClaimsPrincipal(new ClaimsIdentity[]
    {
        new ClaimsIdentity(claims, defaultScheme)
    });
}
catch (Exception e)
{
    Console.WriteLine(e);
    validatedToken = new JwtSecurityToken();
    return new();
}
}
}

```

Лістинг класу HttpRequestHelper:

```

namespace Points.Services.OAuth.Helpers;

public class HttpRequestHelper
{
    public async Task<T> SendPostRequest<T>(string endpoint, Dictionary<string, string> bodyParams)
    {
        var httpContent = new FormUrlEncodedContent(bodyParams);

        return await SendHttpRequestAsync<T>(HttpMethod.Post, endpoint, httpContent: httpContent);
    }

    public async Task<T> SendGetRequest<T>(string endpoint, Dictionary<string, string?>? queryParams)
        => await SendHttpRequestAsync<T>(HttpMethod.Get, endpoint, queryParams: queryParams);

    public async Task<T> SendHttpRequestAsync<T>(
        HttpMethod,
        string endpoint,
        string? accessToken = null,
        Dictionary<string, string?>? queryParams = null,
    )
    {
        // Implementation details
    }
}

```

```

    HttpContent? httpContent = null)
{
    var url = queryParams != null
        ? QueryHelpers.AddQueryString(endpoint, queryParams)
        : endpoint;

    var request = new HttpRequestMessage(httpMethod, url);

    if (!string.IsNullOrEmpty(accessToken))
        request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

    if (httpContent != null)
        request.Content = httpContent;

    var httpClient = new HttpClient();
    var response = await httpClient.SendAsync(request);

    var resultJson = await response.Content.ReadAsStringAsync();

    if (!response.IsSuccessStatusCode)
        return default;

    return JsonConvert.DeserializeObject<T>(resultJson);
}
}

```

Лістинг модуля graphApi:

```

import type { LocalStorage } from './localStorage';

type ErrorResponse = {
    errors: {
        extensions: {
            code: string;
            codes: string[];
            details: string;
            number: string;
        };
        locations: {
            column: number;
            line: number;
        }[];
        message: string;
    }[];
};

type SuccessResponse<T> = {
    data: T;
};

```

```

const isErrorResponse = <T>(response: T | ErrorResponse): response is ErrorResponse => {
  return (response as ErrorResponse).errors !== undefined;
};

const createGraphApi = (tokenStorage: LocalStorage) => {
  const graphApi = async <T>(query: string, variables?: object, headers?: HeadersInit) => {
    const authToken = tokenStorage.get();

    const response = await fetch('/graphql', {
      method: 'POST',
      headers: {
        'content-type': 'application/json',
        'authorization': `bearer ${authToken}`,
        ...headers
      },
      body: JSON.stringify({
        query,
        variables,
      }),
    });

    const responseJson = await response.json() as SuccessResponse<T> | ErrorResponse;
    if (isErrorResponse(responseJson))
      return null;

    return responseJson.data as T;
  };

  return graphApi;
};

export default createGraphApi;

```

Лістинг модуля navigation/saga:

```

import type { AppState } from 'store/types';
import { call, put, select, takeLatest } from 'redux-saga/effects';
import { locationChanged, ROUTER_LOCATION_CHANGED } from 'store/router/actions';
import { PageNames, pagePaths, getPageHandlers, TabNames } from './constants';
import { navigateTo } from './actions';
import { saveSettings } from 'store/settings';

function* handlePage(action: ReturnType<typeof locationChanged>) {
  const { pathname, search, state } = action.payload.location;
  const { shouldBeUpdated: updatePage, ...restState } = state || {};

  const page = pagePaths.find(p => p.path === pathname) || pagePaths.find(p => p.pageName === PageNames.NotFound!);
  const hideLayout = page.layout?.hideLayout;

```

```
const hideLayoutSetting = yield select(state => (state as AppState).settings.layout.hideLayout);

if (hideLayout !== hideLayoutSetting)
  yield put(saveSettings({ layout: { hideLayout } }));

const currentTab = yield select(state => (state as AppState).page.tabs[page.tab || TabNames.Other]);
const shouldBeUpdated = !(updatePage || !currentTab);
const searchString = search.slice(1);
const params = Object.fromEntries(searchString.split('&').map(p => p.split('=')));
const path = page.path + search;

const pageRoute = {
  ...page,
  path,
};

const handler = getPageHandlers()[page.pageName];
if (!handler) {
  yield put(navigateTo(pageRoute));
  return;
}

const routeData = {
  page: pageRoute,
  ...params,
  ...restState,
};

yield call(handler, routeData, { shouldBeUpdated });
}

export default function* navigationSaga() {
  yield takeLatest(ROUTER_LOCATION_CHANGED, handlePage);
}
```