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: «Проєкт підсистеми автентифікації онлайн-магазину на базі протоколу Open ID Connect та різних провайдерів ідентифікації»

Виконав студент 4-го курсу групи КІ-20-1  
спеціальності 123 «Комп’ютерна інженерія»

Владислав Мурсалов

Керівник посада, науковий ступінь, вчене звання,

Олена Головня

Рецензент посада, науковий ступінь, вчене звання

Імʼя Прізвище

Житомир – 2023

Державний університет «Житомирська політехніка»

Факультет інформаційно-комп’ютерних технологій

Кафедра комп’ютерної інженерії та кібербезпеки

Спеціальність 123 «Комп’ютерна інженерія»

Освітня програма «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач   
кафедри компʼютерної   
інженерії та кібербезпеки

\_\_\_\_\_\_\_\_\_ Андрій Єфіменко

1 березня 2022 р.

**ЗАВДАННЯ**

на випускну кваліфікаційну роботу бакалавра

Студент: Мурсалов Владислав Радікович

Тема роботи: «Проєкт підсистеми автентифікації онлайн-магазину на базі протоколу Open ID Connect та різних провайдерів ідентифікації»

затверджена Наказом університету від 1 березня 2023 р. № \_\_\_\_

Термін здачі студентом закінченої роботи 10 червня 2023 р.

Вихідні дані роботи (зазначається предмет і об’єкт дослідження):   
вихідні дані для роботи;

Обладнання та програмне забезпечення:

1. … (Обладнання, операційні системи, програмне забезпечення тощо)

2. …

3. …

4. …

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

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Консультант | Підпис, дата | |
| Завдання видав | Завдання прийняв |
| 1 | Керівник чи інші |  |  |
| 2 | викладачі, які |  |  |
| 3 | вас консультували |  |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Імʼя Прізвище

Керівник \_\_\_\_\_\_\_\_\_\_\_ Імʼя Прізвище

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назви етапів випускної кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
| 1 | Постановка завдання |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Владислав Мурсалов

Керівник \_\_\_\_\_\_\_\_\_\_\_ Олена Головня

АНОТАЦІЯ

Завданням кваліфікаційної роботи була розробка проекту підсистеми автентифікації онлайн-магазину на базі протоколу Open ID Connect та різних провайдерів ідентифікації.

Пояснювальна записка до курсового розробка Проекту інфраструктури для онлайн-магазину складається з вступу, трьох розділів, висновків, списку використаної літератури та додатків.

Текстова частина викладена на 27 сторінках друкованого тексту. Список використаних джерел містить 12 найменувань і займає 1 сторінку. В роботі наведено 31 рисунки. Загальний обсяг роботи – 30 сторінок.

Метою кваліфікаційної роботи є прозробка проєкт підсистеми автентифікації онлайн-магазину на базі протоколу Open ID Connect та різних провайдерів ідентифікації

В результаті було розроблено проект, який за допомогою протоколу OIDC здатний ідентифікувати користувачів інтернет магазину за допомогою різних провайдерів ідентифікації.

Ключові слова: OIDC, SAML, ASP.NET MVC, OKTA, AUTH0.

ANNOTATION

The task of the qualification work was to develop a project of an online store authentication system based on the Open ID Connect protocol and various identity providers.

The explanatory note to the course development of the Infrastructure Project for an online store consists of an introduction, three chapters, conclusions, a list of references and appendices.

The text part is set out on 27 pages of printed text. The list of references includes 12 titles and occupies 1 page. The paper contains 31 figures. The total volume of the work is 30 pages.

The purpose of the qualification work is to design an online store authentication subsystem based on the Open ID Connect protocol and various identification providers.

As a result, a project was developed that is able to identify users of an online store using the OIDC protocol with the help of various identification providers.

Keywords: OIDC, SAML, ASP.NET MVC, OKTA, AUTH0.

ЗМІСТ

[ВСТУП 9](#_Toc168857420)

[**РОЗДІЛ 1 АНАЛІЗ ТЕОРІЇ ОСНОВ АВТЕНТИФІКАЦІЇ ЗА ДОПОМОГОЮ SINGLE SIGN-ON** 10](#_Toc168857421)

[**1.1 Автентифікація, авторизація та ідентифікація в сучасних реаліях індустрії** 10](#_Toc168857422)

[**1.2 Недоліки традиційної системи автентифікації та користь Single Sign-on** 12](#_Toc168857423)

[1.2.1. Зручність використання 12](#_Toc168857424)

[1.2.2. Адміністрування та підтримка 14](#_Toc168857425)

[**1.3 Протоколи для реалізації SSO. SAML vs OIDC** 15](#_Toc168857426)

[1.3.1 SAML 16](#_Toc168857427)

[1.3.2 OAuth 2.0 та OIDC 21](#_Toc168857428)

[1.3.3 Порівняння 25](#_Toc168857429)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ РІШЕННЯ 29](#_Toc168857430)

[**2.1. Формування вимог до проекту відповідно до бізнес логіки інтернет-магазину** 29](#_Toc168857431)

[2.1.1 Стратегічні вимоги 29](#_Toc168857432)

[2.1.2 Функціональні вимоги 32](#_Toc168857433)

[2.1.3 Сформовані вимоги 34](#_Toc168857434)

[**2.2 Проектування рішення** 35](#_Toc168857435)

[2.2.1 Огляд технологічного стеку інтернет магазину 35](#_Toc168857436)

[2.2.2 Проектування рішення під існуючу кодову базу 37](#_Toc168857437)

[2.2.3 Проектування взаємодії між основними компонентами алгоритму 39](#_Toc168857438)

[**2.4 Імплементація рішення** 42](#_Toc168857439)

[2.4.1. Динамічна реєстрація IdP 42](#_Toc168857440)

[2.4.2. Автентифікація через IdP 46](#_Toc168857441)

[2.4.3. Вихід з IdP 50](#_Toc168857442)

[2.4.4. Розмежування ролей автентифікованого користувача 50](#_Toc168857443)

[РОЗДІЛ 3 ТЕСТУВАННЯ ІНФРАСТРУКТУРИ 52](#_Toc168857444)

[**3.1 Тестування роботоздатності інтернет магазину** 52](#_Toc168857445)

[3.2. Налаштування Okta як серверу автентифіації 54](#_Toc168857446)

[**3.3. Налаштування Auth0 як серверу автентифіації** 60](#_Toc168857447)

[**3.4 Тестування автентифікації за допомогою Okta** 64](#_Toc168857448)

[**3.5 Тестування автентифікації за допомогою Auth0** 66](#_Toc168857449)

[**ВИСНОВКИ** 69](#_Toc168857450)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 70](#_Toc168857451)

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SSO – Single sign-on

IDaaS – Identity as a Service

SaaS –Software as a Service

OIDC – Open ID Connect

IdP – Identity Provider

RP – Relying Party

JWT – JavaScript Web Token

SP - Service Provider

# ВСТУП

**Актуальність теми.** У сучасному світі інтернет-торгівлі питання безпеки та зручності користування стають все більш важливими для користувачів та власників онлайн-магазинів. Одним з ключових аспектів забезпечення безпеки є автентифікація користувачів, яка дозволяє переконатися у достовірності особи, що здійснює вхід на сайт. Одним з найсучасніших та найбільш ефективних способів реалізації автентифікації є використання технології Single sign-on, який забезпечує безпечний та стандартизований спосіб аутентифікації через різні провайдери ідентифікації.

**Метою кваліфікаційної роботи** є прозробка проєкт підсистеми автентифікації онлайн-магазину на базі протоколу Open ID Connect та різних провайдерів ідентифікації.

Визначена мета обумовлює наступні завдання:

* + Провести аналіз існуючих методів автентифікації та їх переваг і недоліків
  + Дослідити протокол OpenID Connect та його можливості для автентифікації
  + Розробити архітектуру підсистеми автентифікації для онлайн-магазину з урахуванням обраних провайдерів ідентифікації.
  + Обрати провайдерів ідентифікації, які підтримують OpenID Connect, та провести їх порівняльний аналіз.

**Об’єктом дослідження** є Об'єктом дослідження є системи автентифікації для онлайн-магазинів. Предметом дослідження виступає підсистема автентифікації на базі протоколу OpenID Connect з використанням різних провайдерів ідентифікації.

**Предметом дослідження** є протокол OIDC, його інтеграція з .NET та сервісами авторизації Okta та Auth0.

У процесі виконання кваліфікаційної роботи було розроблено проект інтеграції інтернет магазину з різними сервісами авторизації по протоколу OIDC.

# **РОЗДІЛ 1 АНАЛІЗ ТЕОРІЇ ОСНОВ АВТЕНТИФІКАЦІЇ ЗА ДОПОМОГОЮ SINGLE SIGN-ON**

## **1.1 Автентифікація, авторизація та ідентифікація в сучасних реаліях індустрії**

У сучасному світі, де інформаційні технології грають центральну роль у функціонуванні бізнесу, урядових установ і особистого життя, питання забезпечення безпеки даних стають надзвичайно актуальними. Автентифікація, авторизація та ідентифікація є ключовими процесами у цій сфері, що забезпечують надійний контроль доступу до інформаційних ресурсів і захист від несанкціонованого доступу. Ці процеси, кожен з яких виконує специфічну роль, взаємодіють між собою, створюючи багатошарову систему безпеки. В умовах швидкого розвитку технологій та зростання кіберзагроз їх значення стає ще більш очевидним.

**Автентифікація** є фундаментальним процесом, який забезпечує підтвердження особи користувача або системи, що звертається до ресурсу. Від правильності автентифікації залежить безпека всієї системи, оскільки саме цей процес встановлює, чи є користувач тим, за кого він себе видає. Традиційні методи автентифікації, такі як паролі, вже не забезпечують достатнього рівня захисту через їх уразливість до атак типу брутфорс, фішинг і соціальної інженерії. У зв'язку з цим, значну популярність набули багатофакторна автентифікація (MFA) та біометричні методи, які поєднують кілька факторів для підтвердження особи. Використання одноразових паролів (One Time Password, OTP), які генеруються на основі часу або подій, значно ускладнює завдання для зловмисників. Застосування біометричних даних, таких як відбитки пальців, розпізнавання обличчя або сканування райдужної оболонки ока, підвищує рівень безпеки завдяки унікальності цих характеристик.

**Авторизація** є наступним кроком після автентифікації і відповідає на питання: "Що користувач має право робити?". Це процес надання або обмеження доступу до ресурсів або даних на основі прав та привілеїв, встановлених для користувача. Авторизація може бути реалізована на різних рівнях, включаючи рольову авторизацію, де користувачам надаються певні ролі з відповідними правами доступу, та дискреційний контроль доступу, де власник ресурсу вирішує, хто має до нього доступ. У корпоративних середовищах часто застосовується мандатний контроль доступу, де політики безпеки визначають, хто може отримати доступ до інформації залежно від рівня секретності та інших критеріїв. Це забезпечує додатковий рівень захисту і дозволяє гнучко налаштовувати доступ до ресурсів.

**Ідентифікація** є початковим етапом процесу управління доступом і передбачає визначення особи користувача перед автентифікацією. Користувач пред'являє унікальну інформацію, таку як ім'я користувача або ідентифікаційний номер, яка дозволяє системі розпізнати його як конкретну особу. Ідентифікація є критично важливим етапом, оскільки від неї залежить правильність подальшої автентифікації та авторизації. Без надійної ідентифікації система не зможе точно визначити, хто саме намагається отримати доступ до ресурсів, що може призвести до серйозних порушень безпеки.

У сучасних умовах індустрії питання автентифікації, авторизації та ідентифікації набувають нового значення у зв'язку з розвитком хмарних технологій та мобільних пристроїв. Хмарні сервіси, такі як Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform, пропонують зручні і масштабовані рішення, але вони також вимагають надійних механізмів управління доступом. Забезпечення безпеки в хмарних середовищах потребує особливого підходу, оскільки дані зберігаються на віддалених серверах, доступ до яких може бути здійснений з будь-якої точки світу. Це вимагає використання передових методів автентифікації та авторизації, таких як OpenID Connect та SAML, які забезпечують безпечний обмін інформацією між користувачами і сервісами.

Мобільні пристрої також додають нових викликів у сфері автентифікації та авторизації. Зростання популярності мобільних додатків вимагає розробки методів, які забезпечують безпечний доступ до даних з мобільних пристроїв. Використання біометричних методів, таких як відбитки пальців та розпізнавання обличчя, стало стандартом для багатьох сучасних смартфонів. Однак, це також вимагає додаткових заходів для захисту біометричних даних від компрометації.

Конфіденційність даних є ще одним важливим аспектом сучасних реалій індустрії. Використання біометричних методів автентифікації та зберігання особистих даних вимагає високого рівня захисту та дотримання нормативних вимог, таких як Загальний регламент захисту даних (GDPR) в Європейському Союзі. Забезпечення конфіденційності даних потребує комплексного підходу, включаючи шифрування даних, регулярні аудити безпеки та навчання користувачів щодо безпечного поводження з інформацією.

Автентифікація, авторизація та ідентифікація є ключовими складовими забезпечення безпеки в сучасних інформаційних системах. Вони забезпечують надійний контроль доступу до ресурсів, допомагають захистити конфіденційну інформацію та знизити ризики несанкціонованого доступу. Використання сучасних технологій та підходів до управління доступом дозволяє підвищити рівень безпеки та зручності для користувачів, знижуючи ризики несанкціонованого доступу до ресурсів та даних. Їх правильна реалізація та інтеграція в інформаційні системи є ключем до забезпечення надійного захисту даних та безперебійного функціонування бізнесу і урядових установ. Сучасні виклики вимагають постійного вдосконалення методів та технологій, що дозволяє зберігати високий рівень безпеки в умовах швидкого розвитку інформаційних технологій та зростання кіберзагроз.

## **1.2 Недоліки традиційної системи автентифікації та користь Single Sign-on**

Традиційна система автентифікації має кілька недоліків порівняно з технологією Single Sign-On (SSO). Нижче розглянемо основні з них:

### 1.2.1. Зручність використання

Зручність використання є одним із ключових аспектів, що відрізняє технологію SSO від традиційної системи автентифікації. Для користувача це означає суттєве полегшення в процесі взаємодії з різними системами та додатками.

У традиційній системі автентифікації кожен додаток або система має свої власні облікові дані. Користувачу необхідно щоразу вводити ім'я користувача та пароль для кожного ресурсу, до якого він бажає отримати доступ. Це створює ряд незручностей.

Наприклад, користувачеві доводиться запам'ятовувати численні комбінації імен користувачів і паролів, що може призвести до використання однакових паролів для різних систем або до збереження паролів у ненадійних місцях. Наприклад, користувач може мати окремі облікові записи для корпоративної електронної пошти, внутрішніх порталів, систем управління проектами, різних SaaS-додатків тощо.

До тогож, постійна необхідність вводити паролі призводить до втоми від паролів (password fatigue), що може спричинити використання простих або повторних паролів, що знижує загальну безпеку.

SSO кардинально змінює цей процес. Завдяки використанню єдиного облікового запису для доступу до всіх необхідних систем та додатків, користувачі мають змогу значно спростити собі життя.

Користувач входить в систему лише один раз, використовуючи єдиний набір облікових даних. Після цього він автоматично отримує доступ до всіх інтегрованих додатків без необхідності повторного введення логіну та паролю.

SSO дає можливість імплементувати безшовну інтеграцію, Після автентифікації користувач може переміщуватися між різними системами і додатками без переривання робочого процесу. Наприклад, після входу в корпоративний портал користувач може миттєво перейти до системи управління проектами, CRM-системи або електронної пошти, не вводячи знову свої облікові дані.

Це також вирішує проблему password fatigue, адже зманшується кількість запитів на відновлення паролів, оскільки користувачам легше запам'ятати один надійний пароль замість десятків різних.

### 1.2.2. Адміністрування та підтримка

Адміністрування та підтримка користувачів є важливими аспектами управління інформаційними системами, і вони суттєво відрізняються у традиційній системі автентифікації та у системах.

У традиційній системі автентифікації кожна система або додаток має свої власні облікові дані та механізми автентифікації. Це означає, що адміністратори повинні окремо керувати обліковими записами для кожної системи, що викликає ряд проблем.

Найголовніше проблема – проблема децентралізованих даних. Адміністраторам доводиться створювати, змінювати та видаляти облікові записи для кожної системи окремо. Наприклад, при прийнятті нового співробітника на роботу адміністратору потрібно вручну створювати облікові записи в системі електронної пошти, в CRM, в системі управління проектами тощо. Це трудомісткий процес, який збільшує навантаження на ІТ-відділ. До того ж підтримка актуальності облікових даних користувачів у різних системах може бути складною задачею. Наприклад, якщо користувач змінює своє ім’я або посаду, ці зміни потрібно вручну оновити у всіх системах. Відсутність автоматичної синхронізації може призвести до розбіжностей в даних та потенційних проблем з доступом.

Оскільки процеси створення та керування обліковими записами виконуються вручну, зростає ймовірність недостовірності даних. Наприклад, адміністратор може забути створити обліковий запис в одній з систем або неправильно налаштувати права доступу, що призведе до затримок у роботі користувача або до проблем з безпекою, займе багато часу і буте джерелом потенційних помилок, особливо в великих організаціях з численними користувачами і системами.

Використання технології SSO значно спрощує адміністрування та підтримку користувачів, забезпечуючи централізоване управління обліковими записами та доступами. Ось як це виглядає:

SSO вирішує проблема децентралізованих даних, дозволяючи адміністраторам керувати обліковими записами користувачів з однієї централізованої системи. Наприклад, при прийнятті нового співробітника на роботу адміністратор створює один обліковий запис на сервері адентичності, який автоматично надає доступ до всіх необхідних систем та додатків. Це значно скорочує час та зусилля, необхідні для налаштування доступів.

До того ж зміни цього облікового записа автоматично синхронізуються у всіх системах, інтегрованих з SSO. При зміні ролей або обов’язків співробітника адміністратор вносить зміни лише в одному місці, і ці зміни автоматично застосовуються до всіх відповідних систем. Це зменшує ймовірність помилок та підвищує ефективність управління доступами. Крім того, при звільненні співробітника адміністратор може миттєво відкликати всі доступи, просто деактивувавши один обліковий запис.

Завдяки спрощенню процесу автентифікації та централізованому керуванню обліковими записами, значно зменшується кількість запитів на відновлення паролів та інших запитів до ІТ-відділу. Користувачам не потрібно запам'ятовувати кілька паролів, що зменшує кількість інцидентів, пов’язаних з забутими паролями.

Також комерційні рішення для реалізації Single Sign-On (SSO) часто відзначаються детально розробленими системами аудиту та моніторингу. Вони забезпечують ефективний аналіз подій, виявлення незвичайної активності та оперативну реакцію на потенційні загрози.Протоколи для реалізації Single sign-on. SAML vs OIDC

## **1.3 Протоколи для реалізації SSO. SAML vs OIDC**

Порівняння протоколів SAML (Security Assertion Markup Language) та OIDC (OpenID Connect) є важливим аспектом при виборі систем автентифікації та авторизації для сучасних веб-додатків. Обидва протоколи мають свої унікальні особливості, переваги та недоліки, що робить їх придатними для різних сценаріїв використання.

**Основи та призначення**

SAML - це стандарт, який базується на XML для обміну даними автентифікації та авторизації між різними доменами безпеки. Він був розроблений для великих організацій та підприємств, де потрібна взаємодія між різними системами і службами, що використовують єдиний набір облікових даних користувачів. SAML використовується головним чином у випадках, коли потрібна міждоменна автентифікація, наприклад, між постачальниками послуг (Service Providers, SP) та постачальниками ідентичності (Identity Providers, IdP).

OIDC - це протокол автентифікації, побудований поверх OAuth 2.0, який використовує JSON замість XML. Він був розроблений для полегшення інтеграції автентифікації з сучасними веб-додатками, мобільними додатками та API. OIDC підтримує більш легку та гнучку модель автентифікації, яка краще підходить для динамічних та розподілених систем.

### 1.3.1 SAML

Протокол SAML є складним та багатокомпонентним стандартом для обміну даними автентифікації та авторизації між різними доменами безпеки. Для того, щоб зрозуміти роботу цього протоколу, необхідно детально розглянути три основні компоненти: Assertions, Protocols та Bindings.

**Assertions (Підтвердження)** у SAML є основним носієм інформації, яку обмінюють між IdP та SP. Підтвердження містять дані про автентифікацію користувача, його атрибути та рішення щодо авторизації. Підтвердження у SAML зазвичай структуровані у вигляді XML-документів.Розглянемо детально три основні типи підтвердженнь у SAML: Authentication Assertions, Attribute Assertions та Authorization Decision Assertions.

**Authentication Assertions** (Підтвредження автентифікації) містять інформацію про автентифікацію суб’єкта (користувача) у певний момент часу за допомогою певного методу автентифікації. Це підтвердження використовується для підтвердження того, що користувач був успішно автентифікований системою постачальника ідентичностей (IdP).

Структура Authentication Assertion включає такі елементи, як момент часу автентифікації (AuthnInstant), який вказує точний час проведення автентифікації, та метод автентифікації (AuthnContext), який описує спосіб автентифікації, наприклад, використання паролю або сертифіката. Це підтвердження також може містити інформацію про місцезнаходження суб'єкта під час автентифікації, наприклад, IP-адресу (SubjectLocality).

**Attribute Assertions (Атрибутивні підтвердження)** пов’язані з атрибутами суб’єкта, такими як ім'я, електронна адреса, роль у системі або будь-яка інша інформація, яка може бути використана для ідентифікації та авторизації користувача. Атрибутивні підтвердження дозволяють передавати додаткові дані про користувача, які можуть бути використані для надання персоналізованих послуг або для прийняття рішень щодо доступу до ресурсів.

Ці підтвердження включають елементи, що описують конкретні атрибути (Attribute), кожен з яких має назву, тип та значення. Атрибути можуть бути закодовані у різних форматах, залежно від специфічних вимог та використовуваних схем даних. Атрибутивні підтвердження забезпечують гнучкість у передачі детальної інформації про користувача, що є корисним у складних системах з багатьма різнорідними ресурсами.

**Authorization Decision Assertions** (підвердження про рішення авторизації) містять інформацію про рішення щодо надання або відмови в доступі до певного ресурсу для суб'єкта. Це підтвердження використовується для передачі результатів перевірки прав доступу до ресурсів, на підставі політик безпеки, визначених SP.

Підтвердження про рішення авторизації включають елементи, що описують дію (Action), яку користувач хоче виконати (наприклад, читання, запис, видалення), ресурс (Resource), до якого користувач намагається отримати доступ, та рішення (Decision), яке може бути дозволом (Permit), забороною (Deny) або невизначеністю (Indeterminate). Це підтвердження також може містити додаткову інформацію (Evidence), що підтверджує підстави для прийняття рішення.

Узагальнюючи, кожен з трьох типів тверджень у SAML відіграє важливу роль у забезпеченні безпечного та ефективного обміну інформацією між різними системами. Authentication Assertions підтверджують факт автентифікації користувача, Attribute Assertions передають додаткову інформацію про користувача, а Authorization Decision Assertions визначають права доступу користувача до ресурсів. Разом ці твердження забезпечують повний спектр функцій, необхідних для комплексного управління ідентичностями та доступом у сучасних інформаційних системах.

**Protocols (Протоколи)** у SAML визначають правила обміну твердженнями між IdP та SP. Вони описують формати запитів і відповідей, які використовуються для отримання тверджень, а також інші дії, такі як запити атрибутів або авторизації. Протоколи SAML також реалізовані у вигляді XML-документів.

Протокол SAML має кілька важливих аспектів, серед яких виділяються протоколи Single Logout, Authentication Request та Assertion Query and Request. Кожен з цих протоколів виконує специфічні функції в межах SAML та забезпечує різні аспекти безпеки та управління сесіями.

**Single Logout Protocol** (SLO) у SAML дозволяє одночасно завершувати всі сесії, надані певним центром сесій, майже одночасно. Цей протокол використовується, коли користувач виходить із системи на одній з учасниць сесії або безпосередньо у центрі сесій. Протокол також може бути використаний для завершення сесії у випадку тайм-ауту. Причина виходу може бути вказана через атрибут Reason.

Коли користувач ініціює процес виходу на одній з учасниць сесії, ця учасниця повинна надіслати повідомлення “LogoutRequest” до центру сесій, який надав твердження, що містить автентифікаційний запис, пов'язаний з цією сесією. Якщо користувач ініціює вихід у центрі сесій, або ж учасниця сесії надсилає запит на вихід до центру сесій з вказаним користувачем, центр сесій повинен надіслати повідомлення LogoutRequest до кожної учасниці сесії, якій він надав твердження з автентифікаційними записами в межах поточної сесії з користувачем. Центр сесій повинен спробувати зв'язатися з якомога більшою кількістю цих учасниць, завершити власну сесію з користувачем та, нарешті, повернути повідомлення LogoutResponse до запитуючої учасниці сесії, якщо така є.

**Authentication Request Protocol,** протокол запиту автентифікації дозволяє учасниці сесії надіслати запит на автентифікацію користувача до центру автентифікації. У відповідь центр автентифікації повертає відповідне твердження, що містить результат автентифікації. Цей процес включає кілька ключових етапів:

1. Учасниця сесії надсилає запит на автентифікацію користувача.

2. Центр автентифікації обробляє запит і проводить автентифікацію користувача.

3. У разі успішної автентифікації центр автентифікації генерує твердження, яке містить інформацію про автентифікацію, і надсилає його назад до учасниці сесії.

Цей протокол забезпечує надійну та захищену передачу автентифікаційної інформації між учасницями сесії та центром автентифікації.

**Assertion Query and Request Protocol** - Протокол запиту та отримання тверджень (Assertion Query and Request Protocol) дозволяє учасниці сесії надіслати запит на отримання певних тверджень від центру автентифікації. Цей протокол використовується для отримання інформації, що відповідає певним критеріям, визначеним у запиті.

Процес складається з таких етапів:

1. Учасниця сесії формує запит на отримання тверджень, що відповідають певним критеріям.

2. Центр автентифікації отримує та обробляє запит.

3. Центр автентифікації формує відповідь, що містить запитувані твердження, і надсилає її назад до учасниці сесії.

Цей протокол дозволяє гнучко та ефективно отримувати необхідну автентифікаційну інформацію, що сприяє підвищенню загальної безпеки та управління сесіями в межах SAML.

**Bindings**

Протокол SAML використовує механізми прив’язки (Binding), щоб інтегрувати свої протоколи з різними транспортними механізмами, такими як HTTP та SOAP. Ці прив’язки забезпечують стандартизований спосіб передачі SAML-повідомлень між IdP та SP. Основними видами зв’язувань у SAML є HTTP Redirect Binding, HTTP POST Binding та SOAP Binding.

**HTTP Redirect Binding** - використовується для передачі SAML-повідомлень через URL-параметри у HTTP-запитах методом GET. Цей метод прив’язки часто застосовується для ініціації автентифікації з браузера користувача. У цьому випадку запит на автентифікацію формується IdP та перенаправляється на SP, де він обробляється. Основною перевагою HTTP Redirect Binding є те, що він дозволяє передавати запити без потреби в криптографічному захисті самої передачі, оскільки запити та відповіді передаються через захищене з’єднання HTTPS.

При використанні цього методу, SAML-запит кодується у формі Base64 та передається у вигляді параметра URL. Потім браузер користувача перенаправляється на IdP, де користувач проходить автентифікацію. Після успішної автентифікації IdP формує відповідь SAML, кодує її та передає назад SP через браузер користувача.

**HTTP POST Binding** - використовується для передачі SAML-повідомлень через тіла HTTP-запитів методом POST. Цей метод забезпечує можливість передачі великих обсягів даних, таких як SAML Assertions, які перевищують обмеження на довжину URL. HTTP POST Binding часто використовується для передачі SAML-повідомлень з серверу на сервер або для передачі даних, які потребують більшого рівня захисту та цілісності.

При використанні HTTP POST Binding, SAML-повідомлення кодується у формі Base64 та включається у тіло HTTP-запиту. Потім цей запит надсилається на IdP для обробки. Після успішної автентифікації IdP формує відповідь SAML, кодує її та передає назад SP через тіло HTTP-відповіді.

**SOAP Binding** - використовується для передачі SAML-повідомлень через протокол SOAP, що забезпечує надійну та захищену передачу даних між серверами. SOAP є протоколом обміну структурованими повідомленнями у розподілених системах, і він підтримує різні транспортні протоколи, включаючи HTTP та SMTP. SOAP Binding забезпечує високий рівень захисту та гнучкості, що робить його придатним для складних корпоративних середовищ.

При використанні SOAP Binding, SAML-повідомлення інкапсулюється у SOAP-повідомлення, яке передається через транспортний протокол, наприклад, HTTP. Це повідомлення може бути підписано та зашифровано, що забезпечує автентифікацію та цілісність даних. SOAP Binding часто використовується у випадках, коли необхідна взаємодія між різними системами у рамках корпоративної мережі або між організаціями.

### 1.3.2 OAuth 2.0 та OIDC

Протокол OAuth 2.0 є авторизаційним фреймворком, який дозволяє третім сторонам отримувати обмежений доступ до HTTP-сервісів, діючи від імені користувача. Основна мета OAuth 2.0 полягає в розділенні ролей клієнта та власника ресурсу, що дозволяє уникнути необхідності передачі облікових даних користувача третім сторонам.

У протоколі OAuth 2.0 визначено чотири основні ролі:

1. **Resource Owner**, або власник ресурсу, є особою або системою, яка володіє або контролює доступ до певних ресурсів. Це може бути користувач веб-додатка, який надає дозвіл на доступ до своїх даних, таких як фотографії, контакти або інша особиста інформація. Власник ресурсу є основним суб'єктом, який може делегувати обмежені права доступу до своїх ресурсів іншим додаткам або сервісам.

2. **Resource Server**, або сервер ресурсу, є сервером, який хостить захищені ресурси і здатний обробляти запити на доступ до них. Він відповідає за перевірку дійсності токенів доступу, які надаються клієнтами, і забезпечує доступ до ресурсів у відповідності до отриманих дозволів. Наприклад, якщо клієнт намагається отримати доступ до фотографій користувача, сервер ресурсу перевіряє токен доступу, щоб переконатися, що він дійсний і має відповідні права доступу.

3. **Клієнт (Client)** є додатком, який бажає отримати доступ до ресурсів, що належать власнику ресурсу і розміщені на сервері ресурсу. Це може бути веб-додаток, мобільний додаток або будь-яка інша програма, яка виконує дії від імені користувача. Клієнт спочатку взаємодіє з сервером авторизації, щоб отримати дозвіл на доступ, а потім використовує отриманий токен доступу для взаємодії з сервером ресурсу. Наприклад, мобільний додаток, який хоче отримати доступ до контактів користувача, повинен спочатку отримати токен доступу від сервера авторизації.

4. **Authorization Server**, або сервер авторизації, є сервером, який відповідає за видачу токенів доступу клієнтам після успішного проходження процесу авторизації. Він взаємодіє з власником ресурсу, щоб отримати дозвіл на видачу токена доступу клієнту. Сервер авторизації аутентифікує клієнта та власника ресурсу, а потім видає токен доступу, який клієнт може використовувати для доступу до ресурсів на сервері ресурсу. Сервер авторизації також може видавати токени оновлення, які дозволяють клієнтам отримувати нові токени доступу після закінчення строку дії попередніх токенів.

**OAuth 2.0 Grants**

OAuth 2.0 Authorization Framework описує декілька грантів (grant types), які дозволяють різним типам клієнтів отримувати токени доступу для доступу до захищених ресурсів.

**1. Authorization Code Grant** є найпоширенішим і найбезпечнішим методом отримання токенів доступу. Цей метод передбачає, що клієнт (наприклад, веб-додаток) спочатку переспрямовує користувача на сервер авторизації, де користувач аутентифікується і підтверджує дозвіл на доступ до ресурсів. Після успішної аутентифікації та авторизації сервер видає авторизаційний код, який клієнт отримує через переспрямування користувача назад до клієнта. Потім клієнт використовує цей код для запиту токена доступу від сервера авторизації. Цей процес зменшує ризик викрадення токена, оскільки сам токен не передається через браузер.

2. **Implicit Grant** схожий на Authorization Code Grant, але спрощує процес, оскільки токен доступу видається безпосередньо в результаті авторизаційного запиту, без потреби в обміні на авторизаційний код. Цей метод часто використовується для клієнтів, які є веб-додатками або односторінковими додатками (SPA), де зберігання секретів клієнта небезпечно. Однак цей підхід менш безпечний, оскільки токен доступу передається через браузер і може бути викрадений.

3. **Resource Owner Password Credentials Grant** дозволяє клієнту безпосередньо отримати токен доступу, використовуючи ім’я користувача і пароль ресурсу власника. Цей метод передбачає, що користувач передає свої облікові дані клієнту, який потім відправляє їх на сервер авторизації для отримання токена доступу. Хоча цей підхід може бути корисним у випадках, коли клієнт і сервер авторизації належать одній організації або для додатків, які довіряють клієнту, він менш безпечний і не рекомендований для загального використання, оскільки вимагає передачі облікових даних через мережу.

4. **Client Credentials Grant** призначений для отримання токенів доступу додатками, які не діють від імені конкретного користувача, а скоріше як автономні суб'єкти. У цьому випадку клієнт запитує токен доступу, використовуючи свої власні облікові дані (ідентифікатор клієнта і секрет клієнта). Цей підхід часто використовується для сервер-серверних інтеграцій, де клієнт потребує доступу до ресурсів, які не залежать від конкретного користувача.

**Access Token**

Access Token є ключовим елементом в OAuth 2.0, який надає клієнту право доступу до захищених ресурсів на сервері ресурсів. Ці токени можуть мати різні формати, включаючи JWT, і містять інформацію про права доступу, термін дії та інші атрибути. Сервер ресурсів перевіряє дійсність токена і, якщо він є дійсним, надає доступ до запитуваних ресурсів.

**Безпека в OAuth 2.0**

OAuth 2.0 забезпечує високий рівень безпеки, оскільки Access Token можуть бути підписані та зашифровані, що захищає їх від підробки та несанкціонованого доступу. Протокол також рекомендує використовувати захищені з'єднання HTTPS для передачі токенів, щоб запобігти їх перехопленню.

«Протокол OpenID Connect (OIDC) розширює OAuth 2.0, додаючи шар автентифікації, який дозволяє клієнтам перевіряти особу кінцевого користувача, використовуючи дані, отримані від сервера авторизації.» [1]. Для досягнення цієї мети OIDC використовує кілька ключових компонентів, таких як ID Token, UserInfo Endpoint, Authorization Endpoint та Token Endpoint.

**ID Token**

ID Token – «Основним розширенням, яке OpenID Connect вносить до OAuth 2.0, щоб забезпечити автентифікацію кінцевих користувачів, є структура даних ID Token. ID Token - це маркер безпеки, який містить твердження про автентифікацію кінцевого користувача сервером авторизації при використанні клієнта, а також потенційно інші запитувані твердження. ID Token представлений у вигляді веб-токену у форматі JSON Web Token» [1].

Цей токен надається клієнту під час процесу автентифікації та містить інформацію, таку як ідентифікатор видавця токену (issuer), ідентифікатор користувача (subject), аудиторію токену (audience), час випуску та закінчення терміну дії токену. ID Token також може містити інші клейми, які запитує клієнт, що дозволяє отримувати додаткову інформацію про користувача, таку як його електронну пошту або ім'я.

JWT — це стандартний спосіб передачі даних між двома сторонами у вигляді JSON-об'єктів. Ці дані можуть бути підписані за допомогою секретного ключа (HMAC) або асиметричного ключа (RSA). Підписані JWT гарантують цілісність даних та захист від несанкціонованого доступу.

Структура JWT складається з трьох частин, розділених крапками:

1. Header - секція яка містить інформацію про тип токена та алгоритм підпису або шифрування.

2. Payload – секція з корисними даними, що містить клейми про користувача та інші дані. Це основна частина, де зберігається інформація.

3. Signature - підпис створюється за допомогою алгоритму, зазначеного в заголовку, та секретного ключа або приватного ключа.

Header і Payload навантаження кодуються у форматі Base64Url, після чого ці частини об'єднуються разом з підписом, що формує повний JWT.

JWT має кілька ключових переваг:

1. Завдяки використанню формату JSON та кодуванню Base64Url, JWT має невеликий розмір, що робить його зручним та компактним для передачі через URI та заголовки HTTP.

2. Вся необхідна інформація для перевірки автентичності та авторизації міститься всередині самого токена, що зменшує кількість запитів до серверів автентифікації та робить токен самодостатнім.

3. JWT може бути підписаний та зашифрований, що забезпечує захист від підробки та несанкціонованого доступу.

**UserInfo Endpoint**

«UserInfo Endpoint - є захищеним ресурсом OAuth 2.0, який повертає клейми про аутентифікованого кінцевого користувача» [1]

є захищеним ресурсом, який, отримуючи токен доступу від клієнта, повертає авторизовану інформацію про кінцевого користувача. Клієнт надсилає запит до UserInfo Endpoint, додаючи токен доступу у заголовок запиту. У відповідь сервер надає JSON-об'єкт, що містить затверджені клейми про користувача, наприклад, його ім'я, електронну адресу та інші профільні дані. Це дозволяє клієнту отримувати оновлену інформацію про користувача після початкової автентифікації.

**Authorization Endpoint** використовується для ініціації процесу автентифікації та отримання коду авторизації. Клієнт надсилає запит на авторизацію до цього ендпоінту, вказуючи необхідні параметри, такі як тип відповіді (response type), ідентифікатор клієнта (client ID), перенаправлення URI (redirect URI) та scope, який включає 'openid'. Після автентифікації користувача сервер авторизації надає клієнту код авторизації, який може бути обміняний на токени доступу та ID Token на Token Endpoint.

**Token Endpoint** використовується для обміну коду авторизації на токени доступу та ID Token. Клієнт надсилає запит до цього ендпоінту, включаючи код авторизації, ідентифікатор клієнта, секрет клієнта (client secret) та URI перенаправлення. У разі успішної перевірки сервер авторизації повертає токени доступу та ID Token, які клієнт може використовувати для доступу до захищених ресурсів та для отримання інформації про користувача з UserInfo Endpoint.

### 1.3.3 Порівняння

**Механізми автентифікації та авторизації**

У SAML автентифікація відбувається шляхом передачі SAML Assertions від IdP до SP через браузер користувача. Ці твердження містять інформацію про успішну автентифікацію користувача та інші атрибути, необхідні для авторизації.

OIDC, з іншого боку, використовує токени доступу та оновлення (refresh token), що надаються через OAuth 2.0. ID Token містить інформацію про автентифікацію користувача та може бути перевірений безпосередньо SP без потреби у додаткових запитах до IdP. Це забезпечує більш швидку та ефективну автентифікацію.

**Безпека та шифрування**

SAML використовує XML Signature та XML Encryption для забезпечення цілісності та конфіденційності даних. Це дозволяє захистити SAML Assertions від підробки та несанкціонованого доступу під час передачі між IdP та SP.

OIDC використовує JSON Web Signature (JWS) та JSON Web Encryption (JWE) для забезпечення безпеки токенів. JWT токени можуть бути підписані та зашифровані, що забезпечує високий рівень безпеки при їх передачі та зберіганні.

**Сумісність та інтеграція**

SAML є більш складним у налаштуванні та інтеграції, оскільки вимагає глибокого розуміння XML та пов'язаних з ним технологій. Він також часто використовується у великих корпоративних середовищах, де необхідна висока ступінь безпеки та контроль доступу.

OIDC є більш гнучким та простим у використанні завдяки своєму JSON формату та тісній інтеграції з OAuth 2.0. Це робить його ідеальним для використання у сучасних веб-додатках та мобільних додатках, де важлива швидкість та простота інтеграції.

**Сфери використання OIDC:**

1. **Веб-додатки** - однією з головних сфер використання OIDC є автентифікація користувачів у веб-додатках. Веб-додатки можуть швидко та безпечно перевіряти особу користувача та отримувати необхідну інформацію про нього, що робить OIDC ідеальним для інтерактивних сайтів та платформ.

2. **Мобільні додатки** - OIDC є надзвичайно корисним для мобільних додатків, оскільки забезпечує легку автентифікацію користувачів через мобільні інтерфейси. Використання OIDC дозволяє розробникам інтегрувати безпечну автентифікацію з різними мобільними платформами, що спрощує процес логіну та покращує досвід користувачів. Мобільні додатки можуть використовувати ID Token для перевірки автентичності та отримання профільної інформації користувача.

**3. Single Page Applications (SPA)** - також широко використовують OIDC для автентифікації користувачів. Оскільки SPA вимагають динамічної взаємодії з сервером без перезавантаження сторінки, OIDC дозволяє легко реалізувати автентифікацію та управління сесіями. Це забезпечує безперебійний та безпечний користувацький досвід у таких додатках.

**4. Взаємодія між сервісами (API) -** OIDC використовується для автентифікації та авторизації доступу до API. Завдяки підтримці OAuth 2.0, OIDC дозволяє розробникам легко інтегрувати захист API, надаючи токени доступу, які можуть бути використані для авторизації запитів до серверів. Це особливо корисно для мікросервісної архітектури, де взаємодія між сервісами потребує високого рівня безпеки та контролю доступу.

**5. Електронна комерція (e-commerce)** - OIDC ідеально підходить для інтеграції в системи електронної комерції, забезпечуючи безпечну та зручну автентифікацію користувачів. Використовуючи OIDC, інтернет-магазини можуть легко впроваджувати функціонал єдиного входу (Single Sign-On), покращуючи користувацький досвід та забезпечуючи безпечний доступ до облікових записів клієнтів.

**Cфери використання SAML**

**1. Корпоративне використання** - Корпорації використовують SAML для реалізації єдиної системи входу (SSO) у своїх внутрішніх системах. Це дозволяє співробітникам отримувати доступ до різноманітних додатків та ресурсів, використовуючи один набір облікових даних. SAML забезпечує високий рівень безпеки через підписання та шифрування тверджень, що гарантує цілісність та конфіденційність переданих даних. Крім того, SAML підтримує складні політики автентифікації, що дозволяє корпоративним IT-відділам ефективно керувати доступом до ресурсів.

**2. Урядові та освітні установи -** SAML також широко використовується в урядових установах для забезпечення безпечної автентифікації громадян та співробітників. Уряди використовують SAML для інтеграції різних державних порталів, що дозволяє громадянам отримувати доступ до різних державних послуг, використовуючи одну пару облікових даних. Це підвищує зручність та безпеку, знижуючи необхідність у створенні окремих облікових записів для кожного сервісу.

Порівнюючи SAML та OIDC для впровадження в інтернет-магазині, OIDC є очевидним переможцем завдяки своїй простоті, гнучкості та підтримці сучасних технологій. SAML забезпечує високу безпеку багато функціоналу, але його складність може бути зайвою для інтернет-магазину, де важливі швидкість та легкість інтеграції. Таким чином, OIDC є кращим вибором для забезпечення автентифікації та авторизації в інтернет-магазині.

## **Висновки до розділу 1**

У першому розділі було проведено глибокий аналіз теоретичних основ автентифікації з використанням технології Single Sign-On (SSO). Було визначено, що традиційні методи автентифікації, такі як використання паролів, вже не забезпечують належного рівня безпеки через вразливість до атак. Було показано, що використання багатофакторної автентифікації та біометричних методів підвищує рівень безпеки. Також розглянуто протоколи SAML та OIDC, де OIDC було визначено як більш гнучкий та зручний для інтеграції у сучасні веб-додатки та мобільні додатки. Таким чином, SSO є перспективною технологією, яка дозволяє забезпечити зручний та безпечний доступ до ресурсів, знижуючи ризики несанкціонованого доступу та покращуючи загальний рівень безпеки системи.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ РІШЕННЯ

## **2.1. Формування вимог до проекту відповідно до бізнес логіки інтернет-магазину**

### 2.1.1 Стратегічні вимоги

Основою бізнес логіки інтернет магазину є модель SaaS (Software as a Service) — це модель розповсюдження програмного забезпечення, при якій користувачі отримують доступ до програм через інтернет, використовуючи веб-браузер. У цій моделі програмне забезпечення розміщується на серверах постачальника і керується ним, що дозволяє клієнтам використовувати його без необхідності встановлювати або підтримувати його на своїх власних комп'ютерах чи серверах.

Однією з головних переваг SaaS є те, що вона знижує витрати на IT-інфраструктуру та обслуговування. Це особливо актуально для e-commerce рішень, які часто потребують складних і потужних програмних рішень для управління своїми операціями. Використання такого підходу дозволить нашому замовнику зосередитися на основному бізнесі, не відволікаючись на технічні аспекти підтримки програмного забезпечення.

Для нас SaaS має кілька важливих переваг. По-перше, це швидкий запуск. У порівнянні з традиційними моделями, Наш продукт може бути розгорнути значно швидше, оскільки не вимагатиме складної інсталяції та налаштування на місці. Це особливо важливо для бізнесів, які прагнуть швидко вийти на ринок і почати продавати свої товари своїм покупцям.

По-друге, це масштабованість. Замовники можуть змінювати розмір своїх SaaS-рішень відповідно до зростання або скорочення бізнесу. SaaS-платформи часто пропонують гнучкі плани підписки, які дозволяють збільшувати або зменшувати обсяг використання ресурсів без значних витрат чи зусиль. Це означає, що замовники можуть легко адаптувати свої технологічні потреби до мінливих умов ринку.

По-третє, це доступ до новітніх технологій та оновлень. SaaS-постачальники регулярно оновлюють свої продукти, додаючи нові функції та виправляючи виявлені проблеми. Наші клієнти зможуть автоматично отримувати доступ до оновлень, що дозволяє їм залишатися на передовій технологічних інновацій без додаткових витрат. Це особливо важливо в умовах швидкоплинного розвитку електронної комерції, де конкурентна перевага може бути отримана завдяки впровадженню нових технологій.

Крім того, SaaS-рішення часто інтегруються з іншими сервісами та платформами, що дозволяє інтернет-магазинам створювати комплексні екосистеми для управління всіма аспектами бізнесу. Наприклад, інтеграція з платіжними системами, логістичними сервісами, CRM-системами та маркетинговими інструментами дозволяє автоматизувати багато процесів та підвищити ефективність роботи.

Важливим аспектом SaaS є також зручність у користуванні. Більшість SaaS-платформ розроблені з урахуванням потреб користувачів, що забезпечує інтуїтивно зрозумілий інтерфейс і легкість у використанні. Це означає, що навіть співробітники без глибоких технічних знань можуть швидко навчитися працювати з програмним забезпеченням, що знижує витрати на навчання та підтримку користувачів.

Загалом, використання SaaS для інтернет-магазину є стратегічним рішенням, що дозволяє знизити витрати, підвищити гнучкість, забезпечити безпеку та доступність, а також швидко впроваджувати нові технології. Це дозволяє інтернет-магазинам залишатися конкурентоспроможними та ефективними в умовах динамічного ринку електронної комерції.

Відповідно стратегічним подходом до розвитку автентифікації магазину буде використання моделі **Identity as a Service (IDaaS)** — це модель розповсюдження послуг з управління ідентифікацією користувачів, яка надається через хмару. Ця модель включає в себе автентифікацію, авторизацію, керування доступом та інші функції, пов'язані з ідентифікацією, які допомагають організаціям захистити свої дані та ресурси. Використання IDaaS для SaaS-рішень матиме численні переваги, особливо для замовників, які прагнуть забезпечити безпеку своїх даних та забезпечити зручний доступ для своїх користувачів.

Однією з основних причин, чому варто використовувати IDaaS для SaaS, є підвищення рівня безпеки. У сучасному цифровому світі захист даних стає критично важливим аспектом діяльності будь-якого бізнесу. Замовники, що працюють з великими обсягами особистих та фінансових даних клієнтів, повинні забезпечити високий рівень безпеки, щоб уникнути витоків даних та кіберзагроз. IDaaS-рішення за замовчуванням пропонують багато рівнів захисту, включаючи багатофакторну автентифікацію, шифрування даних та інші механізми безпеки, що знижують ризики несанкціонованого доступу.

Ще однією важливою перевагою IDaaS є централізоване управління ідентифікаціями. Це особливо корисно для наших замовників, які мають велику кількість користувачів і потребують ефективних засобів для управління доступом. Централізоване управління дозволяє адміністраторам легко створювати, змінювати та видаляти облікові записи користувачів, а також надавати або обмежувати доступ до різних ресурсів на основі ролей та привілеїв. Це значно полегшує процеси управління та забезпечує кращу контрольованість доступу.

До того ж ми можемо виграти від зниження витрат на управління ідентифікацією. Замість того, щоб інвестувати у власні системи безпеки та управління доступом, ми можемо скористатися послугами постачальника IDaaS, що забезпечить всі необхідні функції в рамках одного рішення. Це дозволяє значно знизити витрати на підтримку та розвиток IT-інфраструктури, одночасно отримуючи доступ до найсучасніших технологій та експертизи.

Також, важливим аспектом використання IDaaS для нас є відповідність вимогам законодавства та регуляторних норм. Багато країн та регіонів мають суворі правила щодо захисту персональних даних, такі як GDPR у Європейському Союзі. Використання IDaaS дозволяє інтернет-магазинам легше дотримуватися цих вимог, оскільки постачальники IDaaS часто інтегрують у свої рішення необхідні механізми для забезпечення відповідності. Це допомагає уникнути можливих штрафів та зберегти довіру клієнтів.

Крім того, IDaaS забезпечує гнучкість та масштабованість, що є критично важливими для нас. Клієнти можуть швидко зростати або змінювати свої потреби. IDaaS-рішення можна легко адаптувати до змін у кількості користувачів або нових вимог до доступу, що дозволяє бізнесу швидко реагувати на нові виклики. Ця гнучкість також означає, що інтернет-магазин може легко інтегрувати нові сервіси або додатки без необхідності значних змін у своїй інфраструктурі.

Важливою перевагою IDaaS є можливість інтеграції з іншими хмарними та локальними системами. Це дозволяє нашим клієнтам створювати єдину екосистему для управління ідентифікацією, яка охоплює всі їхні ресурси та сервіси.

На додаток до всіх технічних переваг, використання IDaaS може також сприяти покращенню взаємодії з клієнтами. Завдяки більш зручним та безпечним методам автентифікації, кінцеві клієнти наших замовників отримують кращий досвід користування, що може підвищити їхню лояльність та задоволеність. Це особливо важливо длянас, адже ми прагнемо забезпечити високий рівень обслуговування та створити позитивний імідж на ринку.

Таким чином, використання підходу IDaaS для у нашому інтернет-магазині є стратегічним кроком, що забезпечує безпеку, зручність, ефективність та відповідність законодавчим вимогам. Це дозволить зосередитися на основному бізнесі, знижуючи витрати та ризики, одночасно забезпечуючи найвищий рівень обслуговування для наших клієнтів.

### 2.1.2 Функціональні вимоги

Основна бізнес логіка інтернет-магазину полягає в управлінні товарами, категоріями та користувачами. Користувачі з відповідними ролями можуть переглядати доступні товари, додавати їх до кошика, оформляти замовлення та здійснювати покупки. Адміністратори мають можливість створювати, редагувати та видаляти товари і категорії, управляти сторінками сайту, а також налаштовувати аутентифікацію та авторизацію користувачів.

**Управління товарами та категоріями**

Товари організовані за категоріями, що дозволяє користувачам легко знаходити потрібні продукти. Адміністратори мають можливість додавати нові товари, редагувати їхні характеристики, включаючи назву, опис, ціну та зображення, а також видаляти непотрібні товари. Категорії також можуть бути створені, редаговані та видалені адміністраторами, що дозволяє підтримувати актуальність та структуру каталогу товарів.

**Управління сторінками**

Адміністратори можуть створювати та редагувати статичні сторінки сайту, такі як інформаційні сторінки, сторінки з умовами використання, політикою конфіденційності тощо. Це забезпечує можливість швидкої адаптації контенту під потреби користувачів та підтримки актуальної інформації на сайті.

**Управління користувачами та ролями**

Адміністратори мають можливість управляти обліковими записами користувачів, включаючи створення нових користувачів, редагування існуючих даних та видалення облікових записів. Користувачі можуть мати різні ролі, що визначає їх права доступу до певних функцій системи. Наприклад, адміністратори мають повний доступ до управління контентом та налаштувань системи, тоді як звичайні користувачі можуть переглядати товари та здійснювати покупки.

**Оформлення замовлень та управління кошиком**

Користувачі можуть додавати товари до кошика, переглядати вміст кошика, змінювати кількість товарів або видаляти їх з кошика. Після завершення покупок користувачі можуть оформити замовлення. Система зберігає інформацію про замовлення, що дозволяє адміністраторам обробляти їх, оновлювати статуси замовлень та здійснювати інші необхідні операції для виконання замовлень.

Бізнес логіка цього інтернет-магазину побудована таким чином, щоб забезпечити зручність користувачів, ефективне управління контентом та товарами, а також високий рівень безпеки та інтеграції з зовнішніми сервісами. Система дозволяє адміністраторам гнучко налаштовувати та управляти магазином, а користувачам - легко знаходити та купувати товари.

### 2.1.3 Сформовані вимоги

Інтернет-магазин повинен підтримувати інтеграцію з зовнішніми IDaaS сервісами про протоколу OpenID Connect, при тому зберегти весь свій функціонал.

**Захист та безпека**

Система повинна забезпечити захист даних користувачів, використовуючи сучасні методи шифрування та захисту від атак.

**Загальні вимоги**

1. Система автентифікації. повинна забезпечувати безпечний доступ до ресурсів та функціоналу інтернет-магазину, сформованих в пункті 2.1.2, використовуючи сучасні протоколи автентифікації та авторизації на базі OIDC.

2. Модель безпеки повинна підтримувати мультифакторну автентифікацію (MFA) для підвищення рівня безпеки користувачів.

3. Підтримка ролей. Система повинна мати можливість призначати ролі користувачам, забезпечуючи диференційований доступ до функцій та даних інтернет-магазину.

**Вимоги до користувачів**

1. Реєстрація користувачів. Система повинна забезпечувати можливість реєстрації нових користувачів через різних IdP.

2. Управління користувачами. Система повинна дозволяти адміністраторам додавати, редагувати та видаляти облікові записи користувачів, а також змінювати їх ролі та права доступу.

**Вимоги до інтеграції з IdP**

1. Підтримка стандартів. Інтеграція з постачальниками ідентифікації повинна здійснюватися з використанням стандартних протоколів, на базі OIDC.

2. Динамічна конфігурація. Система повинна підтримувати динамічну конфігурацію постачальників ідентифікації, що дозволить адміністраторам легко змінювати параметри підключення.

## **2.2 Проектування рішення**

### 2.2.1 Огляд технологічного стеку інтернет магазину

Основою для нашого інтернет магазину був обраний **ASP.NET MVC** – це одна з популярних платформ для створення веб-додатків, яка була розроблена компанією Microsoft. Вона надає можливість будувати веб-додатки з використанням шаблону проектування MVC, що сприяє розділенню логіки програми, користувацького інтерфейсу та обробки даних. Розглянемо основні можливості ASP.NET MVC детально.

**Модель (Model)** – це компонент, який відповідає за логіку даних програми. Модель взаємодіє з базою даних та іншими джерелами даних, обробляючи їх і забезпечуючи доступ до даних для контролера та представлення. У ASP.NET MVC модель може бути реалізована за допомогою Entity Framework, який дозволяє працювати з базами даних за допомогою об’єктів. Це забезпечує високу абстракцію і зручність роботи з даними.

**Контролер (Controller)** – це компонент, який відповідає за обробку запитів користувача, взаємодію з моделлю та вибір відповідного представлення для відображення даних. Контролер отримує вхідні дані від користувача через URL-запити, обробляє ці дані, звертається до моделі для отримання необхідних даних і передає їх у представлення для відображення. Контролери в ASP.NET MVC зазвичай реалізуються як класи, що успадковуються від базового класу Controller, і містять методи-дії (action methods), які відповідають на різні запити користувача.

**Представлення (View)** – це компонент, який відповідає за відображення даних на веб-сторінці. Представлення в ASP.NET MVC зазвичай реалізуються за допомогою Razor-синтаксису, що дозволяє поєднувати HTML з C# кодом. Razor-вид представлення дозволяє створювати динамічні веб-сторінки, де дані з моделі можуть бути інтегровані в HTML-шаблони для відображення.

Однією з ключових переваг ASP.NET MVC є чітке розділення обов'язків між компонентами моделі, представлення та контролера. Це забезпечує високу підтримуваність і масштабованість веб-додатків. Розділення обов'язків дозволяє різним розробникам працювати над різними частинами програми незалежно один від одного, що значно прискорює процес розробки.

ASP.NET MVC підтримує безліч протоколів та стандартів для роботи з даними та взаємодії з іншими системами. Одним з найпоширеніших протоколів є HTTP (Hypertext Transfer Protocol), який використовується для передачі даних між клієнтом і сервером. ASP.NET MVC надає розширені можливості для обробки HTTP-запитів, включаючи підтримку GET, POST, PUT, DELETE та інших методів HTTP. Це дозволяє створювати RESTful веб-сервіси, які можуть взаємодіяти з іншими веб-додатками та мобільними додатками.

В якості бази даних інтернет магазин використовує **Microsoft SQL Server (MSSQL)**. Microsoft SQL Server є потужною реляційною базою даних, яка надає широкий спектр можливостей для підтримки інтернет-магазинів. Цей сервер баз даних забезпечує високу продуктивність, масштабованість та безпеку, що робить його ідеальним вибором для обробки великих обсягів даних і підтримки складних бізнес-процесів.

Однією з ключових можливостей Microsoft SQL Server є його здатність обробляти великі обсяги транзакцій в режимі реального часу. Інтернет-магазини часто стикаються з великим потоком запитів на покупку, оновлення запасів, обробку платежів і управління клієнтськими даними. SQL Server забезпечує високу продуктивність завдяки використанню індексів, кешування запитів та оптимізації виконання запитів, що дозволяє швидко обробляти тисячі запитів одночасно.

SQL Server підтримує різні типи даних і забезпечує потужні механізми для управління складними структурами даних. Це дозволяє зберігати і обробляти дані про товари, категорії, клієнтів, замовлення, оплати та інші аспекти інтернет-магазину. Завдяки підтримці реляційних і нереляційних даних, SQL Server дозволяє створювати гнучкі та ефективні схеми баз даних, що відповідають специфічним потребам нашого бізнесу.

Для забезпечення високої доступності та безперервності бізнесу, SQL Server підтримує різні механізми резервного копіювання та відновлення даних. Ви можете налаштувати автоматичне резервне копіювання даних і використовувати функції журналу транзакцій для відновлення бази даних до будь-якої точки часу в разі збою. Крім того, SQL Server підтримує реплікацію даних і кластеризацію, що дозволяє забезпечити високу доступність і відмовостійкість нашої бази даних навіть у разі виходу з ладу одного з серверів.

Ще однією важливою можливістю SQL Server є підтримка масштабованості. З ростом бізнесу і збільшенням обсягів даних, SQL Server дозволяє легко масштабувати базу даних, додаючи нові сервери або розподіляючи навантаження між існуючими. Ви можете використовувати горизонтальне та вертикальне масштабування для забезпечення стабільної роботи нашого інтернет-магазину навіть при значних пікових навантаженнях.

SQL Server надає потужні засоби для автоматизації управління базою даних. Ви можете використовувати SQL Server Agent для планування і виконання регулярних завдань, таких як резервне копіювання, обслуговування індексів, очищення даних та інших. Це дозволяє знизити витрати на адміністрування бази даних і забезпечити стабільну і безперебійну роботу нашого інтернет-магазину.

Окрім того, SQL Server підтримує різні протоколи для доступу до даних, такі як T-SQL (Transact-SQL), який є розширенням стандартного SQL і надає додаткові можливості для управління транзакціями, процедур, тригерів та інших об'єктів бази даних. Це забезпечує гнучкість і потужність при розробці складних запитів і бізнес-логіки для нашого інтернет-магазину.

### 2.2.2 Проектування рішення під існуючу кодову базу

Використання Microsoft.AspNetCore.Authentication для імплементації одноразового входу SSO у проекті, написаному на ASP.NET, є надзвичайно розумним і вигідним рішенням. Ця бібліотека пропонує нам потужний інструментарій для створення безпечних, масштабованих і зручних для користувачів систем автентифікації.

Однією з основних причин використання Microsoft.AspNetCore. Authentication для імплементації SSO є її глибока інтеграція з ASP.NET. Це дозволяє нам легко і швидко налаштувати автентифікацію без необхідності писати великий обсяг коду. Вбудовані механізми та можливості конфігурації забезпечують високу гнучкість і дозволяють адаптувати систему автентифікації під специфічні потреби проекту. Крім того, підтримка протоколів автентифікації таких як OpenID Connect і OAuth 2.0, Microsoft.AspNetCore.Authentication забезпечує за замовчуванням.

Microsoft.AspNetCore.Authentication також надає потужні можливості для управління сесіями користувачів. Це включає автоматичне оновлення токенів доступу, відстеження активних сесій і забезпечення безпечного завершення сесій. Завдяки цьому, розробники можуть бути впевнені, що їх додаток завжди буде відповідати найвищим стандартам безпеки і забезпечувати надійний захист персональних даних користувачів.

Ще одним важливим аспектом є підтримка багаторазових автентифікаційних схем. Це означає, що за допомогою Microsoft.AspNetCore.Authentication можна легко реалізувати різні методи автентифікації для різних частин додатку або для різних типів користувачів. Наприклад, для внутрішніх користувачів компанії можна використовувати корпоративний SSO на базі Active Directory, тоді як для зовнішніх користувачів - автентифікацію через соціальні мережі. Це забезпечує високу гнучкість і дозволяє створювати зручні та безпечні рішення, які відповідають потребам різних категорій користувачів.

Активна підтримка та оновлення з боку Microsoft є ще однією вагомою причиною для вибору Microsoft.AspNetCore.Authentication. Ми можемо бути впевнені, що вони використовують сучасні і надійні рішення, які регулярно оновлюються для усунення вразливостей і додавання нових функцій. Це дозволяє завжди залишатися на крок попереду потенційних загроз і забезпечувати стабільну роботу додатку.

До того ж, за допомогою конфігураційних файлів і вбудованих механізмів, ми можемо налаштувати одноразовий вхід з мінімальними зусиллями. Одним з традиційних підходів збереження конфігурацій в ASP.NET є конфігураційний файл appsettings.json. Зазвичай адміністратор перед тим як підняти додаток змінює цей конфігураційний файл, тому має сенс зберігати конфігурацію провайдерів інтеифікації саме там. Це робить процес інтеграції простим і швидким.

Таким чином, використання Microsoft.AspNetCore.Authentication для імплементації одноразового входу в проекті на базі ASP.NET надає численні переваги. Висока інтеграція з платформою, підтримка сучасних стандартів безпеки, зменшення кількості необхідного коду, потужні можливості управління сесіями, підтримка багаторазових автентифікаційних схем, висока продуктивність і масштабованість, а також активна підтримка з боку Microsoft роблять цю бібліотеку ідеальним вибором для розробників, які прагнуть створювати безпечні, зручні і ефективні рішення.

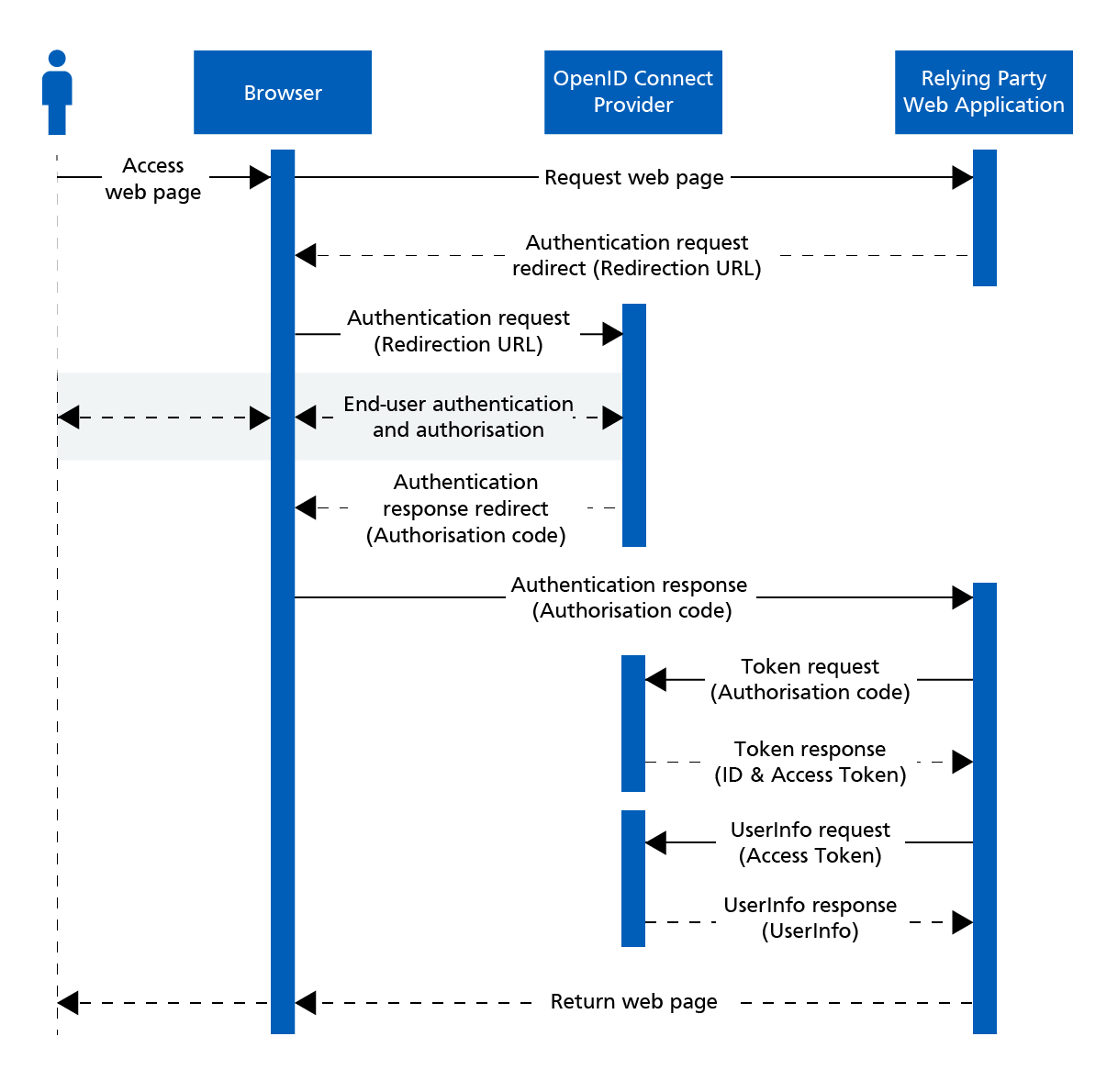
### 2.2.3 Проектування взаємодії між основними компонентами алгоритму

Існують два основних підходи для реалізації автентифікації через сторонні сервіси, такі як OIDC: Popup і Redirect. Popup відкриває невелике вікно для введення облікових даних, що дозволяє користувачу залишатися на тій самій сторінці. Redirect, навпаки, перенаправляє користувача на сторінку провайдера автентифікації, а після завершення процесу повертає його назад. Обидва підходи мають свої переваги та недоліки, що впливають на користувацький досвід і технічну реалізацію.

Ми оберемо підход з переадресацією на сторінку IdP, через наступні причини:

1. Переадресація дозволяє користувачам бачити, що вони вводять свої дані безпосередньо на сайті провайдера (наприклад, Facebook або Google), що підвищує довіру до вашого магазину, адже користувачі бачать, що їхні дані не передаються третім особам.
2. Попапи можуть викликати проблеми на мобільних пристроях і різних браузерах, оскільки вони можуть бути заблоковані, що призводить до плутанини у користувачів. Redirect працює стабільніше на всіх пристроях і браузерах.
3. Переадресація не потребує складної обробки станів або відслідковування різних сценаріїв відкриття попапів, що спрощує підтримку і розвиток функціоналу.

На зображенні зображено процес автентифікації та авторизації з використанням протоколу OpenID Connect засценарієм Redirect:



Опис процесу:

1. Access web page - користувач за допомогою браузера звертається до веб-сторінки у Веб-додатку (Relying Party Web Application).

2. Request web page – мігазин отримує запит на веб-сторінку від браузера.

3. Authentication request redirect (Redirection URL) - магазин переадресовує браузер користувача на Провайдера OpenID Connect для автентифікації, відправляючи запит на автентифікацію.

4. Authentication request (Redirection URL) - браузер надсилає запит на автентифікацію до Провайдера OpenID Connect, використовуючи наданий Redirection URL.

5. End-user authentication and authorization - провайдер OpenID Connect проводить автентифікацію та авторизацію користувача. Це може включати введення користувачем облікових даних або надання згоди на доступ до своїх даних.

6. Authentication response redirect (Authorization code) - після успішної автентифікації та авторизації, Провайдер OpenID Connect переадресовує браузер користувача назад до Веб-додатку з авторизаційним кодом (Authorization code).

7. Authentication response (Authorization code) - магазин отримує авторизаційний код від браузера.

8. Token request (Authorization code) - веб-додаток надсилає запит на отримання токенів (Token request) до Провайдера OpenID Connect, використовуючи авторизаційний код.

9. Token response (ID & Access Token) - Провайдер OpenID Connect відповідає Веб-додатку, надсилаючи ID токен та Access токен.

10. UserInfo request (Access Token) - веб-додаток надсилає запит на отримання інформації про користувача (UserInfo request) до Провайдера OpenID Connect, використовуючи Access токен.

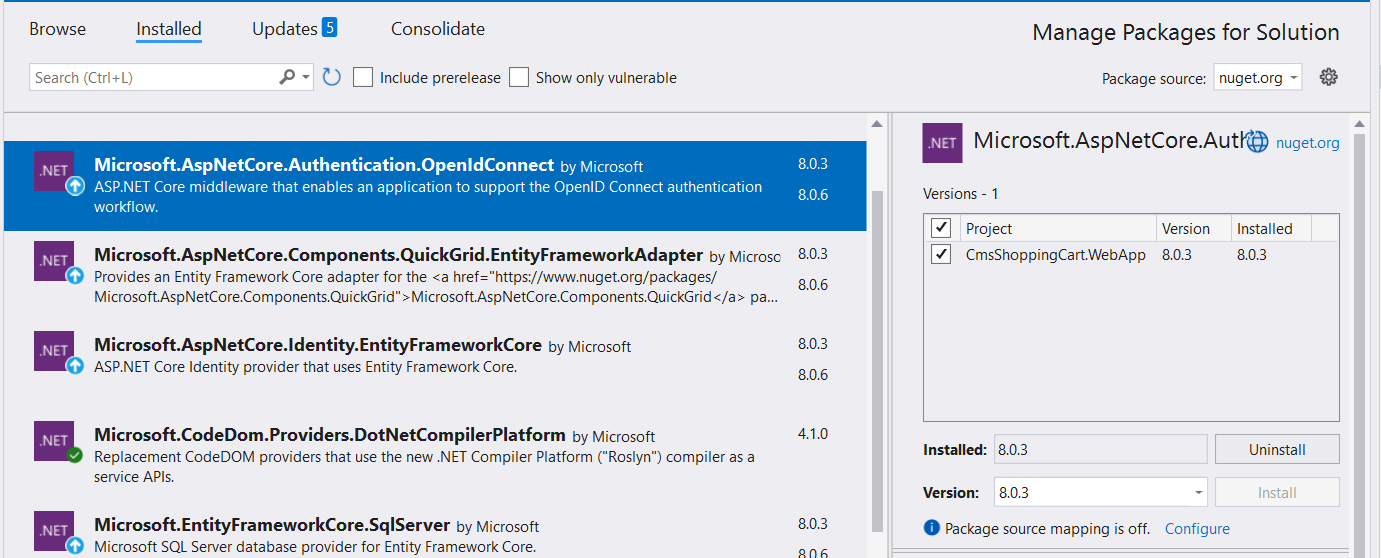
11. UserInfo response (UserInfo) - провайдер OpenID Connect відповідає Веб-додатку, надаючи інформацію про користувача (UserInfo).

12. Return web page - браузер користувача отримує доступ до веб-сторінки у Веб-додатку, вже будучи автентифікованим.

## **2.4 Імплементація рішення**

### 2.4.1. Динамічна реєстрація IdP

1. Спочатку, додамо бібліотеку до проекту, за допомогою NuGet Package Manager:



*Мал. 2.4.1.1 Втсановлення бібліотеки для протоколу OIDC*

В XML налаштуваннях проекту можемо бачити що посилання на пакунок успішно додалося до проекту:

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>net8.0</TargetFramework>

</PropertyGroup>

<ItemGroup>

<Compile Remove="wwwroot\lib\ckeditor\\*\*" />

<Content Remove="wwwroot\lib\ckeditor\\*\*" />

<EmbeddedResource Remove="wwwroot\lib\ckeditor\\*\*" />

<None Remove="wwwroot\lib\ckeditor\\*\*" />

</ItemGroup>

<ItemGroup>

<PackageReference Include="Auth0.AspNetCore.Authentication" Version="1.4.1" />

<PackageReference Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="8.0.3" />

<PackageReference Include="Microsoft.AspNetCore.Components.QuickGrid.EntityFrameworkAdapter" Version="8.0.3" />

<PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="8.0.3" />

<PackageReference Include="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" Version="4.1.0" />

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.3" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.3">

<PrivateAssets>all</PrivateAssets>

<IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

</PackageReference>

</ItemGroup>

</Project>

2. Тепер імплементуємо динамічне зчитування та налаштовування налаштовує систем автентифікації з використанням. Нам потрібно зчитати з файлу конфігурації наступну структуру:

1. Autority – посилання на сервер авторизації, що в результаті буде автентифіковувати кінцевого користувача.
2. ClientId – валідний для серверу автентифікації ідентифікатор клієнта, що налаштовується на самому сервері ідетифікації, відповідно до його документації.
3. Secret – секрет, за допомогою якого наш сервер авторизації буде розпізнавати що це саме наш інтернет магазин а не зловмисник.
4. CallbackPath – шлях, на який повернеться користувач від IdP з результами авторизації користувача.
5. LoginButtonImageUrl – посилання на картинку, що буде відображена в кнопці для авторизації.
6. LoginButtonText – текст який буде відображено на кнопці автентифікації.

Ці властивості будуть спільними для всіх інтеграцій, тому структура нашого налаштування буде виглядати наступним чином:

"SSO": {

"OIDCProdider1": {

"Authority": "OIDCProvider1 authority URL",

"ClientId": "OIDCProvider1 application client id",

"ClientSecret": "secret",

"CallbackPath": "/OIDCProvider1/signin-oidc",

"LoginButtonImageUrl": "~/media/OIDCProvider1-logo.png",

"LoginButtonText": "Login With OIDCProvider1"

},

" OIDCProdider2": {

"Authority": "OIDCProvider2 authority URL",

"ClientId": "OIDCProvider2 application client id",

"ClientSecret": "secret",

"CallbackPath": "/ OIDCProvider2/signin-oidc",

"LoginButtonImageUrl": "~/media/OIDCProvider2-logo.png",

"LoginButtonText": "Login With OIDCProvider2"

}

}

Додмо клас який буде використовується для збереження конфігураційних параметрів, необхідних для налаштування OpenID Connect провайдерів:

public class SSOConfiguration

{

public string Authority { get; set; }

public string ClientId { get; set; }

public string ClientSecret { get; set; }

public string CallbackPath { get; set; }

public string LoginButtonImageUrl { get; set; }

public string LoginButtonText { get; set; }

}

Зчитування конфігурації:

var b = new AuthenticationBuilder(services);

var providers = Configuration["SSO"];

var config = JsonSerializer.Deserialize<Dictionary<string, SSOConfiguration>>(providers);

Створюється екземпляр `AuthenticationBuilder`, який використовується для додавання та налаштування автентифікаційних схем у додатку, в нього треба передати services типу IserviceCollection. Він є частиною системи впровадження залежностей (Dependency Injection) в ASP.NET Core. Це інтерфейс, який представляє контейнер для реєстрації служб, необхідних нашого додатку.

З конфігураційного файлу зчитуються параметри для всіх SSO провайдерів і десеріалізуються в словник, де ключ - це ім'я провайдера, а значення - об'єкт `SSOConfiguration`.

Далі напишемо код, що дозволяє налаштувати декілька провайдерів OpenID Connect для автентифікації користувачів. Кожен провайдер буде налаштовуватися з використанням своїх параметрів:

foreach (var provider in config)

{

var providerName = provider.Key;

var providerConfig = provider.Value;

b.AddOpenIdConnect(providerName, o =>

{

o.Authority = providerConfig.Authority;

o.ClientId = providerConfig.ClientId;

o.ClientSecret = providerConfig.ClientSecret;

o.CallbackPath = providerConfig.CallbackPath;

o.Scope.Add("email");

o.UsePkce = true;

o.ResponseType = OpenIdConnectResponseType.Code;

o.SignInScheme = IdentityConstants.ApplicationScheme;

o.ClaimActions.Add(new SimpleClaimAction("AuthenticationType", providerName));

o.TokenValidationParameters.RoleClaimType = "account\_role";

o.Events = new OpenIdConnectEvents()

{

OnRedirectToIdentityProviderForSignOut = context =>

{

var logoutUri = "/";

context.Response.Redirect(logoutUri);

context.HandleResponse();

return Task.CompletedTask;

}

};

});

}

- Зчитується ім'я провайдера (`providerName`) та його конфігурація (`providerConfig`).

- Додається схема автентифікації OpenID Connect для цього провайдера з налаштуванням властивостей:

- Встановлюються параметри `Authority`, `ClientId`, `ClientSecret`, `CallbackPath` які беруться з конфігурації провайдера.

- Додається скоуп `email` для запитів до OpenID провайдера. Це означає, що додаток запитує доступ до електронної пошти користувача. За замовчуванням, OpenIdConnectOptions додаються скоупи openid та profile.

- Використовується PKCE (Proof Key for Code Exchange) для додаткової безпеки.

- Встановлюється тип відповіді як `Code`.

- Визначається схема входу. Саме схема ApplicationScheme використовується для авторизації в інтернет магазині і на ній зав’язана існуюча логіка.

- Визначається account\_role як клейм для ролі в токені.

- Налаштовуються подія для обробки виходу з системи, Це потрібно щоб перед тим як перенаправити користувача до IdP вказати на який шлях в в інтернет магазині, потрібно повернути кристувача, а саме на головну.

Також додається клейм `AuthenticationType` в токен, в якому закодовується назва провайдеру. Це потрібно для того щоб клієнтський код зміг ідентифікувати що поточний користувач автентифікувався за допомогою OIDC. За замовчуванням ClaimActions не має такого механізму хардкоду значення в токен. Тому ми додамо свій власний ClaimAction, маємо:

public class SimpleClaimAction : ClaimAction

{

private string SavedValue { get; set; }

public SimpleClaimAction(string claimType, string valueType) : base(claimType, valueType)

{

SavedValue = valueType;

}

public override void Run(JsonElement userData, ClaimsIdentity identity, string issuer)

{

identity.AddClaim(new Claim(ShopClaimTypes.AuthenticationScheme, SavedValue));

}

}

Робота цього класу досить проста. Конструктор приймає два параметри: `claimType` і `valueType`, які використовуються для визначення типу та значення клейма. Викликається базовий конструктор класу `ClaimAction` з переданими параметрами. Значення `valueType` присвоюється властивості `SavedValue`.

Метод Run є перевизначеним методом з базового класу `ClaimAction` і викликається, коли відбувається обробка клеймів. В методі створюється новий клейм за допомогою значення `SavedValue`, яке було збережено під час ініціалізації об'єкта `SimpleClaimAction`. Клейм додається до інформації про автентифікацію користувача.

### 2.4.2. Автентифікація через IdP

Спочатку визначимо, які властивості будемо передавати представленню. Все що нам потрібно це схема автентифікації, задля того щоб потім вказати за яким провайдером ми будемо автентифікуватися, текст кнопки та логотип провайдера, щоб відрендерити ту кнопку на яку користувач натисне для автентифікації. Для цього наступний клас:

public class IdentiryProviderViewModel

{

public string AuthenticationScheme { get; set; }

public string LoginButtonText { get; set; }

public string LoginButtonImageUrl { get; set; }

}

Тепер створимо метод, який буде, обробляти запит на сторінку входу:

[AllowAnonymous]

public IActionResult Login(string returnUrl)

{

var oidcModels = new List<OIDCIdentiryProviderViewModel>();

var providers = configuration["SSO"];

var configurations = JsonSerializer.Deserialize<Dictionary<string, SSOConfiguration>>(providers);

foreach (var configuration in configurations)

{

var providerName = configuration.Key;

var providerConfig = configuration.Value;

oidcModels.Add(new OIDCIdentiryProviderViewModel()

{

AuthenticationScheme = providerName,

LoginButtonText = providerConfig.LoginButtonText,

LoginButtonImageUrl = providerConfig.LoginButtonImageUrl

}); ;

}

var loginViewModel = new LoginViewModel()

{

ReturnUrl = returnUrl,

IdentityProviders = oidcModels

};

return View(loginViewModel);

}

Метод має атрибут [AllowAnonymous], що дозволяє доступ до цього методу без аутентифікації. Метод приймає параметр returnUrl, який використовується для перенаправлення користувача після успішного входу.

У методі створюється список об'єктів типу OIDCIdentiryProviderViewModel. Далі з конфігурації зчитуються налаштування провайдерів єдиного входу (SSO) у вигляді рядка JSON. Цей JSON десеріалізується для подальшого використання.

У циклі foreach обробляється кожен провайдер зі словника конфігурацій. Для кожного провайдера створюється новий об'єкт OIDCIdentiryProviderViewModel з відповідними властивостями: схема аутентифікації, текст кнопки входу та URL зображення кнопки входу. Цей об'єкт додається до списку oidcModels.

Після обробки всіх провайдерів створюється об'єкт LoginViewModel, куди встановлюються returnUrl та список провайдерів аутентифікації (oidcModels).

В результаті метод повертає подання (View) з цим LoginViewModel, що буде використовуватись для відображення сторінки входу.

Модифікуємо представлення. Після основної форми входу додамо динамічне створення форм для входу через налаштовані IdP, перелічених у властивості IdentityProviders моделі. Для кожного провайдера створюється окрема форма з прихованими полями для ReturnUrl та AuthenticationScheme, а також кнопкою, яка містить зображення та текст для відповідного провайдера. Також не мало важливо вказати що при підтвердженні форми буде викликатися asp-action – LoginWithSSO Виглядати це буде наступним чином:

@model CmsShoppingCart.WebApp.Models.Authentication.LoginViewModel

@{

ViewData["Title"] = "Login";

}

<h1>Login</h1>

<div class="row">

<div class="col-md-10">

<**form** **asp-action**="Login">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<**input** **type**="hidden" **asp-for**="ReturnUrl" />

<div class="form-group">

<**label** **asp-for**="Email" class="control-label"></**label**>

<**input** **asp-for**="Email" class="form-control" />

<**span** **asp-validation-for**="Email" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Password" class="control-label"></**label**>

<**input** **asp-for**="Password" class="form-control" />

<**span** **asp-validation-for**="Password" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="Log in" class="btn btn-primary" />

</div>

</**form**>

</div>

@foreach (var provider in Model.IdentityProviders)

{

<div class="col-md-10">

<**form** **asp-controller**="Account" **asp-action**="LoginWithSSO">

<**input** **type**="hidden" **asp-for**="ReturnUrl" />

<**input** **type**="hidden" **asp-for**="AuthenticationScheme" **value**="@provider.AuthenticationScheme" />

<div class="form-group">

<button type="submit" class="btn btn-outline-secondary">

<img src="@provider.LoginButtonImageUrl" width="48" alt="Auth0 logo" />

<span>@provider.LoginButtonText</span>

</button>

</div>

</**form**>

</div>

}

</div>

Імплементація методу LoginWithSSO:

[AllowAnonymous]

[HttpPost]

[ValidateAntiForgeryToken]

public async Task LoginWithSSO(string authenticationScheme, string returnUrl = "/")

{

if (authenticationScheme.IsNullOrEmpty())

throw new ArgumentException("Scheme cannot be null or empty", nameof(authenticationScheme));

var authenticationProperties = new LoginAuthenticationPropertiesBuilder()

.WithRedirectUri(returnUrl)

.Build();

await HttpContext.ChallengeAsync(

authenticationScheme,

authenticationProperties

);

}

Цей метод обролює вже сам запит. Він має три атрибути: [AllowAnonymous] дозволяє доступ до цього методу без аутентифікації, [HttpPost] вказує, що метод обробляє тільки POST-запити, а [ValidateAntiForgeryToken] забезпечує захист від підробки міжсайтових запитів (CSRF).

Метод приймає два параметри: `authenticationScheme`, який вказує на схему аутентифікації, і `returnUrl`, який задає URL для перенаправлення після успішного входу, за замовчуванням встановлений на "/".

Перший крок у методі перевіряє, чи `authenticationScheme` не є порожнім або null, і якщо це так, викидає ArgumentException з відповідним повідомленням.

Далі створюється об'єкт `authenticationProperties` з використанням класу `LoginAuthenticationPropertiesBuilder`. Цей об'єкт налаштовується з редіректом на `returnUrl` після успішної аутентифікації.

Нарешті, метод викликає `HttpContext.ChallengeAsync`, передаючи `authenticationScheme` і `authenticationProperties`. Це ініціює процес аутентифікації через вказану схему, що може включати перенаправлення користувача до IdP.

В результаті, ми маємо апі запит - [POST] https://{domain}/account/loginWithSSO

### 2.4.3. Вихід з IdP

Для виходу з IdP модифікуємо вже існуючий метод Logout.

public async Task<IActionResult> Logout()

{

var authenticationProperties = new LogoutAuthenticationPropertiesBuilder()

.WithRedirectUri(Url.Action("Index", "Home"))

.Build();

var authClaim = User.Claims.FirstOrDefault(c => c.Type == ShopClaimTypes.AuthenticationScheme);

if (authClaim is not null)

await HttpContext.SignOutAsync(authClaim.Value, authenticationProperties);

await HttpContext.SignOutAsync();

return Redirect("/");

}

Для створення параметрів перенаправлення користувача після виходу з IdP, використовується `LogoutAuthenticationPropertiesBuilder`. Метод `WithRedirectUri` вказує, що після виходу користувач буде перенаправлений на сторінку "Index" контролера "Home", тобто на головну сторінку.

Далі іде пошук клейму, який має тип `ShopClaimTypes.AuthenticationScheme`, адже в ньому зберігається інформація про IdP, через який користувача автентифіковано.

Якщо інформація про зовнішній IdP знайдена, викликається метод `SignOutAsync` для відключення користувача з IdP.

Якщо клейм не знайдений,то цей користувач автентифікувався через нашу внутрішню систему автентифікації, тому викликаємо метод SignOutAsync без параметрів, що за замовчуванням завершить внутрішн сесію.

### 2.4.4. Розмежування ролей автентифікованого користувача

Розмежування доступу у ASP.NET MVC забезпечується на рівні контролерів за допомогою атрибутів [Authorize], які можуть бути застосовані як до всього контролера, так і до окремих дій (методів) у контролері. Атрибут [Authorize] дозволяє обмежити доступ до ресурсів лише для автентифікованих користувачів або для користувачів з певними ролями або політиками.

Так буде виглядати захит контроллеру:

[Authorize(Roles = "Admin")]

[Area("Admin")]

public class ShopController : Controller

{ }

Так буде виглядати захит окремого методу контроллеру:

[Area("Shop")]

public class ShopController1 : Controller

{

[Authorize(Roles = "Customer")]

public async Task<IActionResult> Index()

{

return View();

}

}

Вся авторизація ASP.NET MVC побудована на http контексті і великим плюсом нашої імплементації є можливість користуватися стандартними інструментами автентифікації фреймворку.

Для представлень Razor доступ до інформації про ролі буде досягатись наступним чином:

@if (User?.Identity?.IsAuthenticated @@ (User.IsInRole("Admin") | User.IsInRole("Editor"))

{

<li class="nav-item ml-1">

<**a** class="btn btn-success"

**asp-area** = "Admin" **asp-controller**="Pages" **asp-action**="Index">Admin Area</**a**>

</li>

}

## **Висновки до розділу 2**

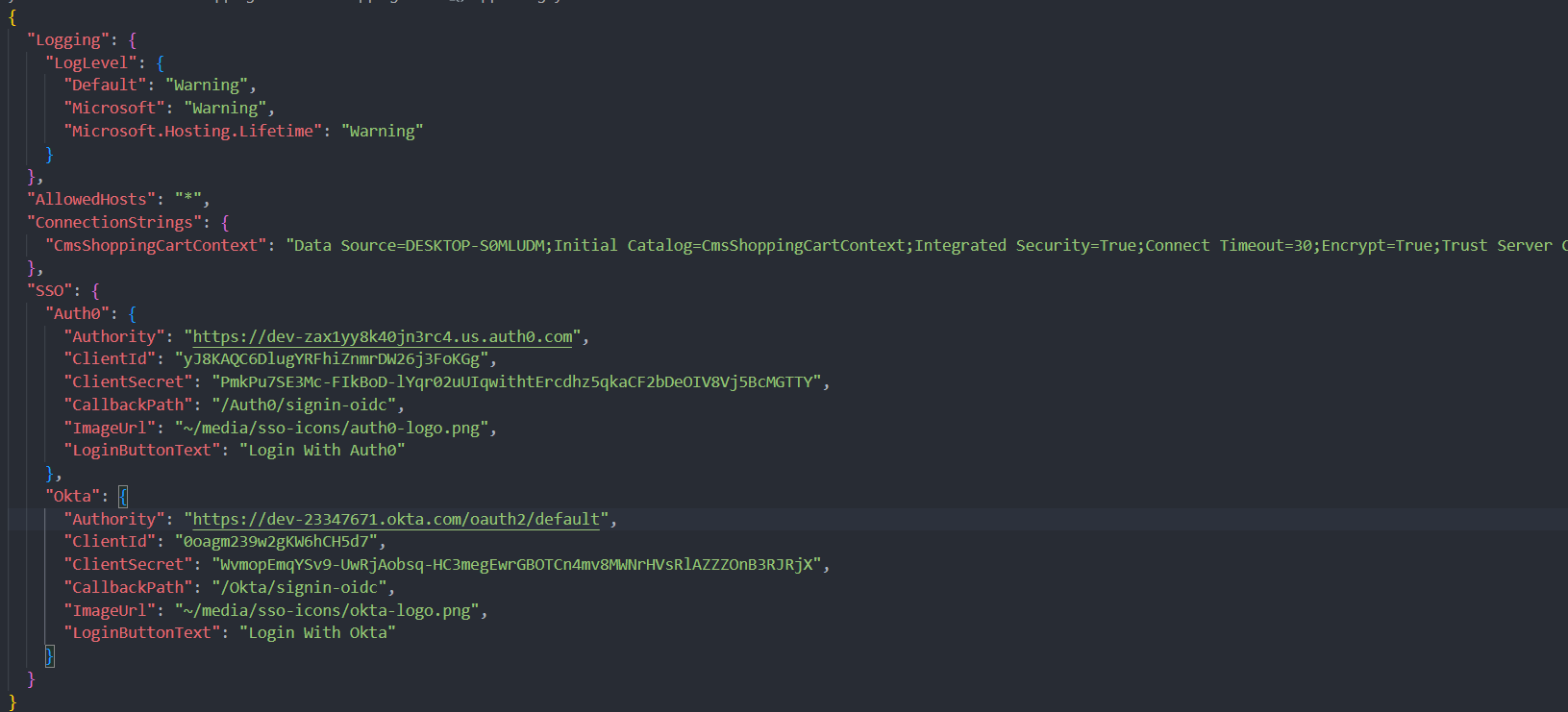
Другий розділ був присвячений проектуванню рішення для автентифікації в онлайн-магазині на основі протоколу OpenID Connect. Було визначено стратегічні та функціональні вимоги до проекту, а також розглянуто бізнес-логіку інтернет-магазину. Основною перевагою використання SaaS та IDaaS моделей було визначено зниження витрат на ІТ-інфраструктуру та підвищення рівня безпеки завдяки централізованому управлінню ідентифікацією користувачів. Проектування рішення включало інтеграцію з зовнішніми сервісами ідентифікації, підтримку мультифакторної автентифікації та диференційованого доступу до функцій і даних інтернет-магазину. В результаті було створено архітектуру системи, яка дозволяє ефективно керувати доступом та забезпечувати високий рівень безпеки користувачів.

# РОЗДІЛ 3 ТЕСТУВАННЯ ІНФРАСТРУКТУРИ

## **3.1 Тестування роботоздатності інтернет магазину**

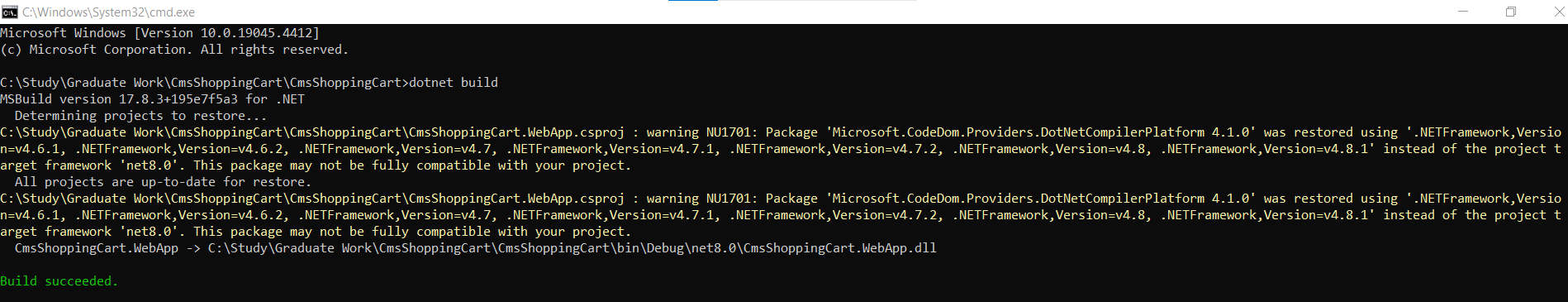
Спочатку нам потрібно сконфігурувати всі IdP в конфіграційному файлі appsettings.json. Для тестування налаштуємо провайдери Okta та Auth0. Детальні налаштування провайдерів приведено в пунктах 3.2 та 3.3 . Відповідно нас цікавить поле “SSO”, в якому потрібно на кожний IdP додати наступні налаштування:

1. Autority – посилання на сервер авторизації, що в результаті буде автентифіковувати кінцевого користувача.
2. ClientId – валідний для серверу автентифікації ідентифікатор клієнта, що налаштовується на самому сервері ідетифікації, відповідно до його документації.
3. Secret – секрет, за допомогою якого наш сервер авторизації буде розпізнавати що це саме наш інтернет магазин а не зловмисник.
4. CallbackPath – шлях, на який повернеться користувач від IdP з результами авторизації користувача.
5. LoginButtonImageUrl – посилання на картинку, що буде відображена в кнопці для авторизації.
6. LoginButtonText – текст який буде відображено на кнопці автентифікації.



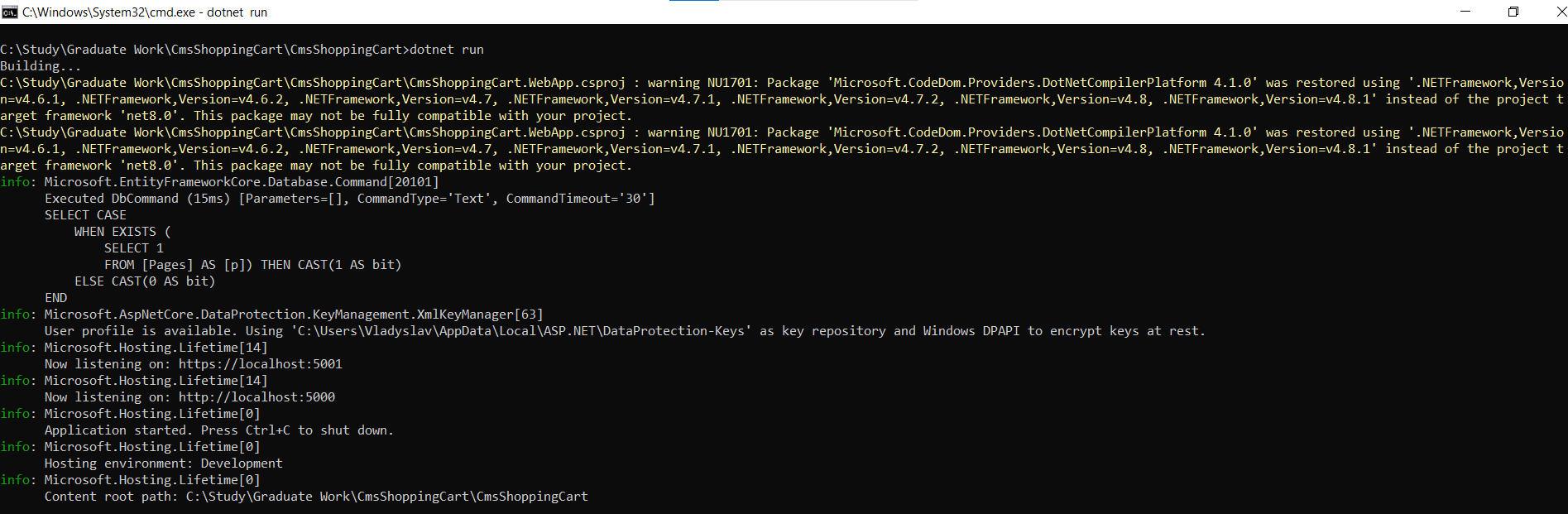
*Мал. 3.1.1 Конфігурація двох IdP в конфіграційному файлі appsettings.json*

Після розробки технології, нам потрібно перевірити чи немає run time помилок при запуску додатку, пому переходимо в теку з проектом, відкриваємо в цій теці консоль, та прописуємо команду «dotnet build»:



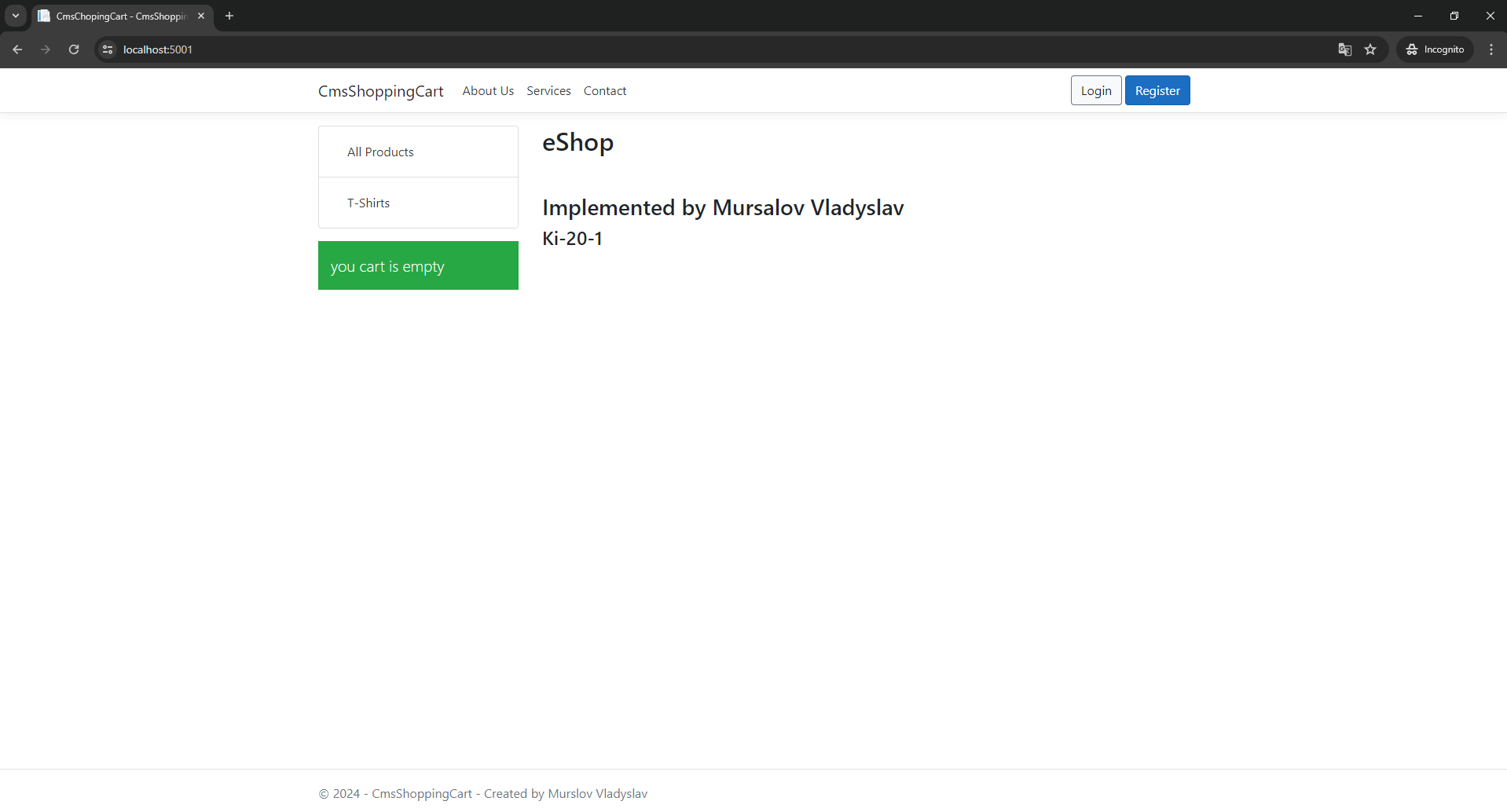
*Мал. 3.1.2 Успішна збірка проекту*

Далі щоб підняти застосунок потрібно прописати команду «dotnet run». Можемо бачити що застосунк успішно запутився та адресу на якій знаходиться наш інтернет-магазин:



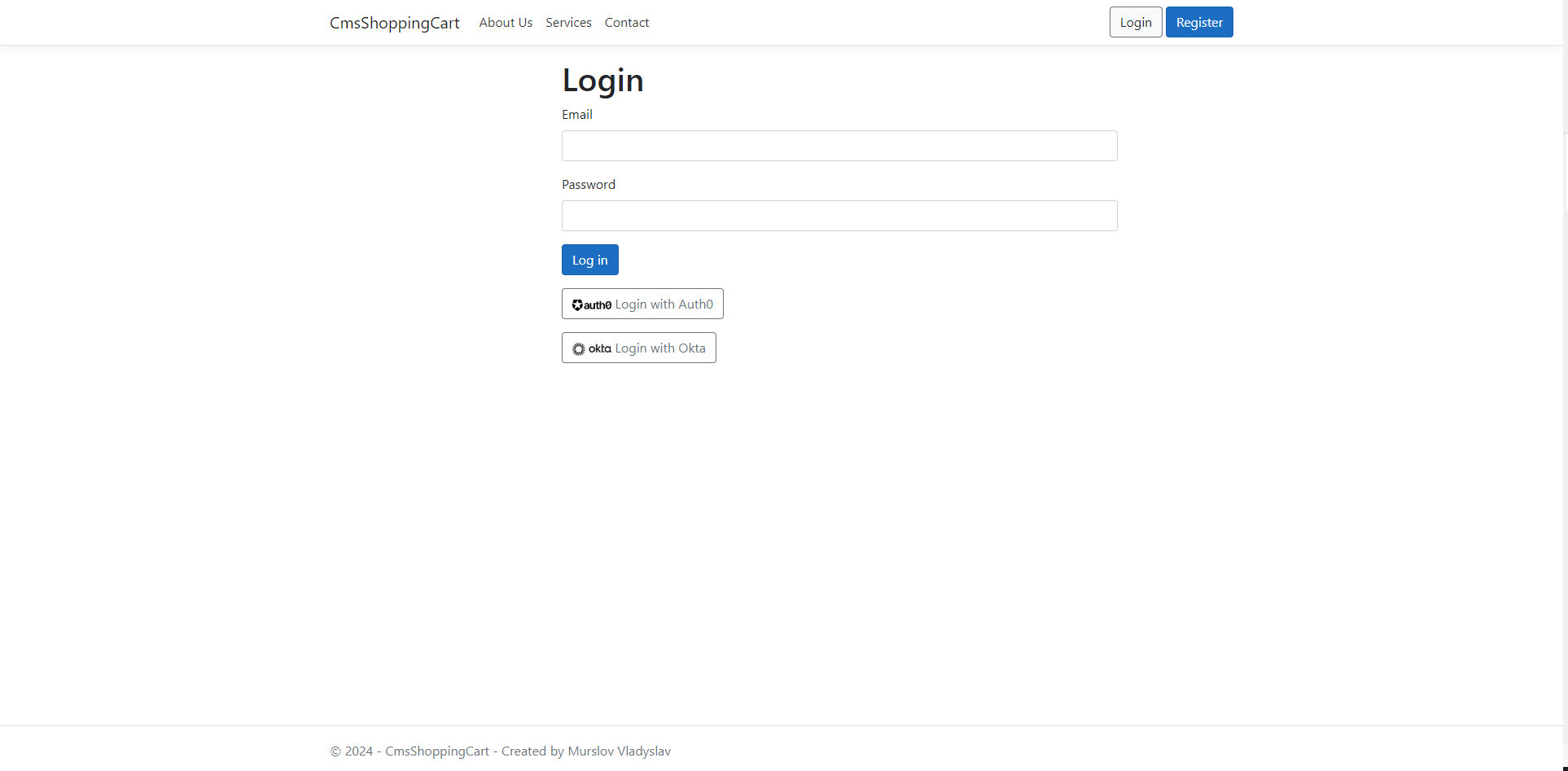
*Мал. 3.1.3 Успішний запуск з консолі*

Тепер переходимо на головну сторінку, щоб запевнитися що все добре.



*Мал. 3.1.3 Головна сторінка магазину*

Тепер перейдемо на сторінку авторизації, щоб запевнитися чи додались посилання на авторизацію на наші 2 провайдери, маємо:



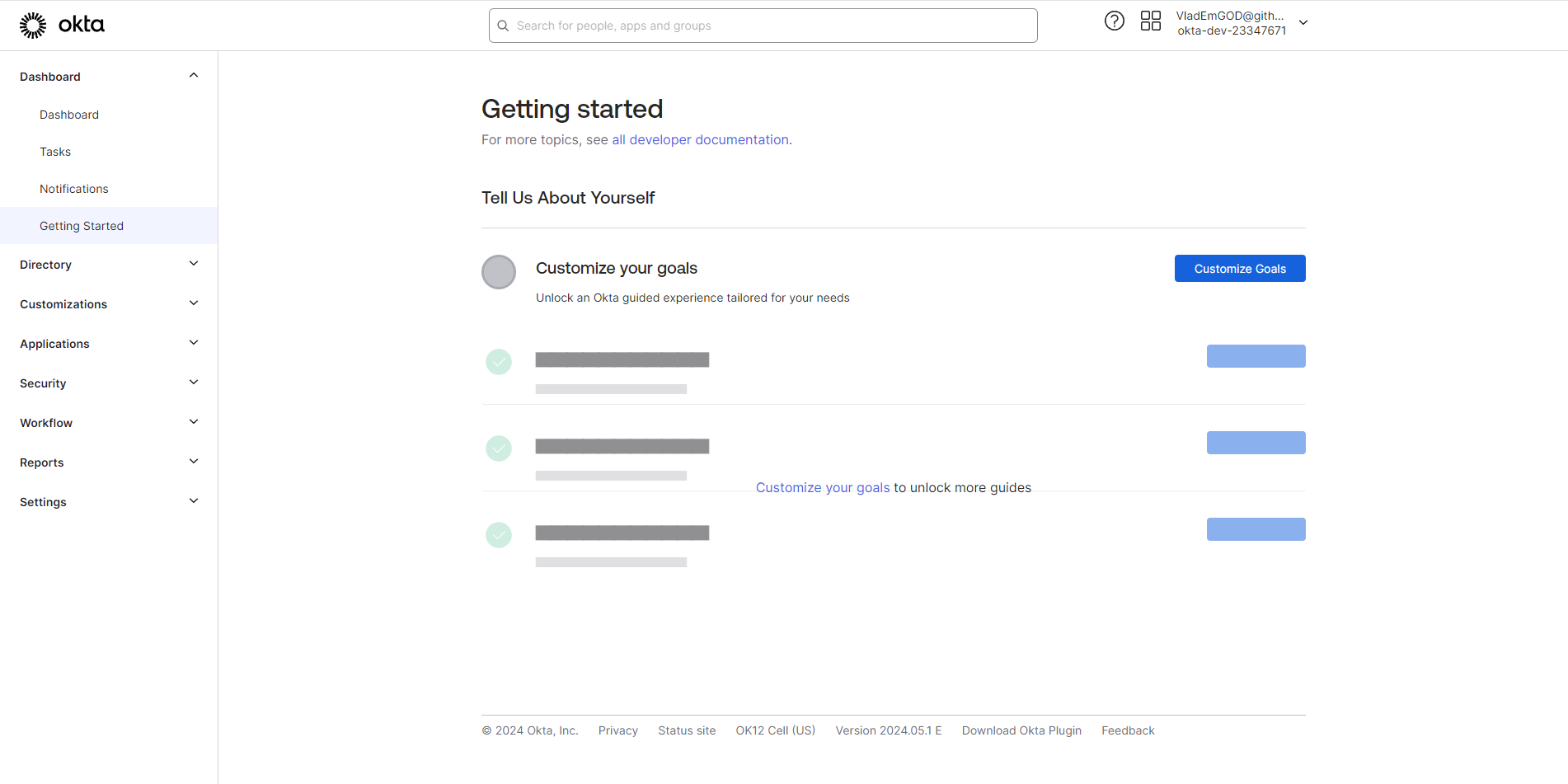
*Мал. 3.1.3 Сторінка авторизації*

Можемо бачити що кнопки успішно додалися і ми налаштували все правильно.

## 3.2. Налаштування Okta як серверу автентифіації

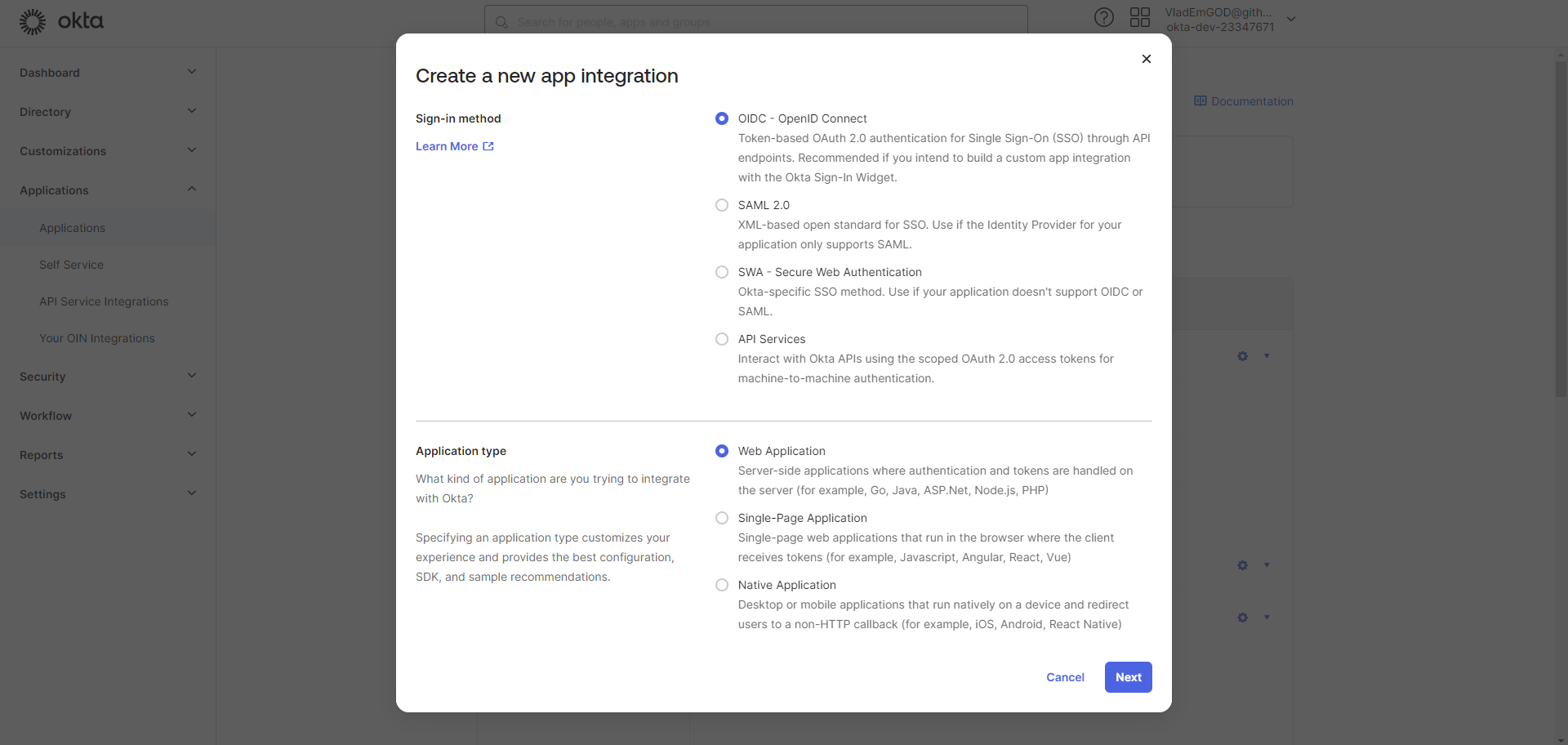
**Налаштування інтеграції**

Спочатку потрібно пройти стандарту процедуру автентифікації на офіційному сайті <https://www.okta.com>, де потрібно буде ввести всю необхідну для сервісу персональну інформацію. Після чого ми зможемо потрапити в директорію:



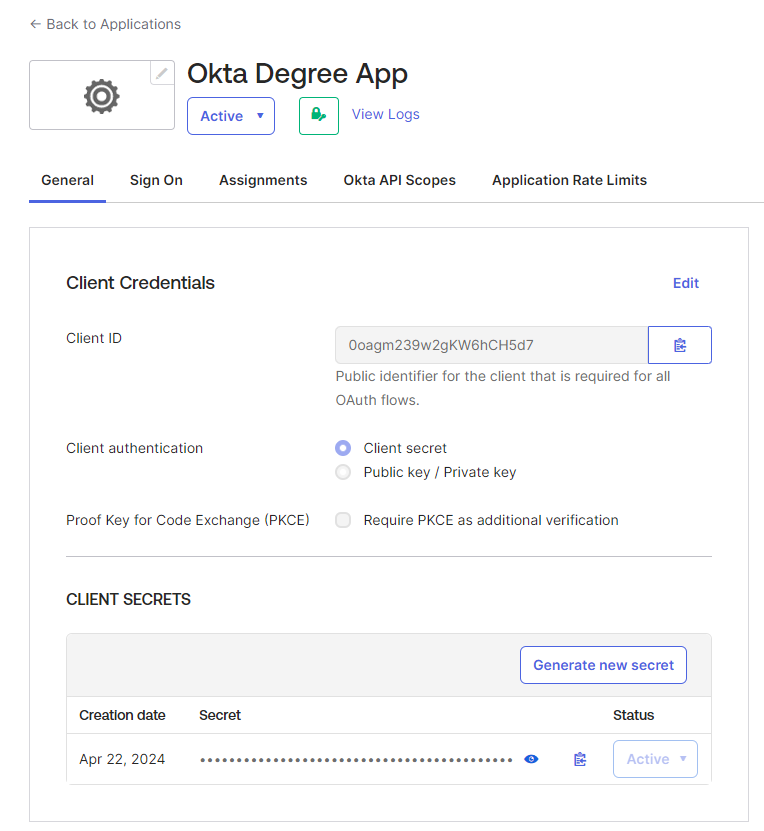
*Мал. 3.2.1 Директорія Okta*

Тепер нам потрібно стоврити інтеграцію для нашого інтернет магазину. треба буде перейти у панелі керування до вкладки "Applications". Тут ми можемо бачити список всіх застосунків. Натискаємо кнопку "Create App Integration", і Okta запропонує нам вибрати тип застосунку. Вибираємо протокол Open Id Connect. Та вибираємо темпейт "Web application".



*Мал. 3.2.2 Створення нової інтеграції для магазину в директорії*

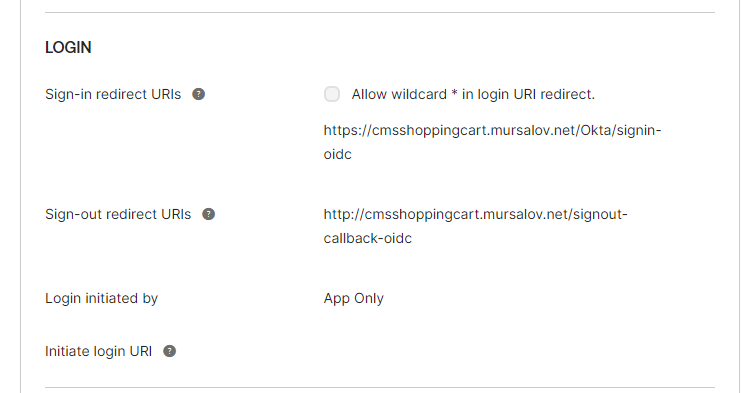
Після створення застосунку, можемо бачити потрібні нам згенеровані client-id та client secret.



*Мал. 3.2.3. Створена інтеграція в провайдері*

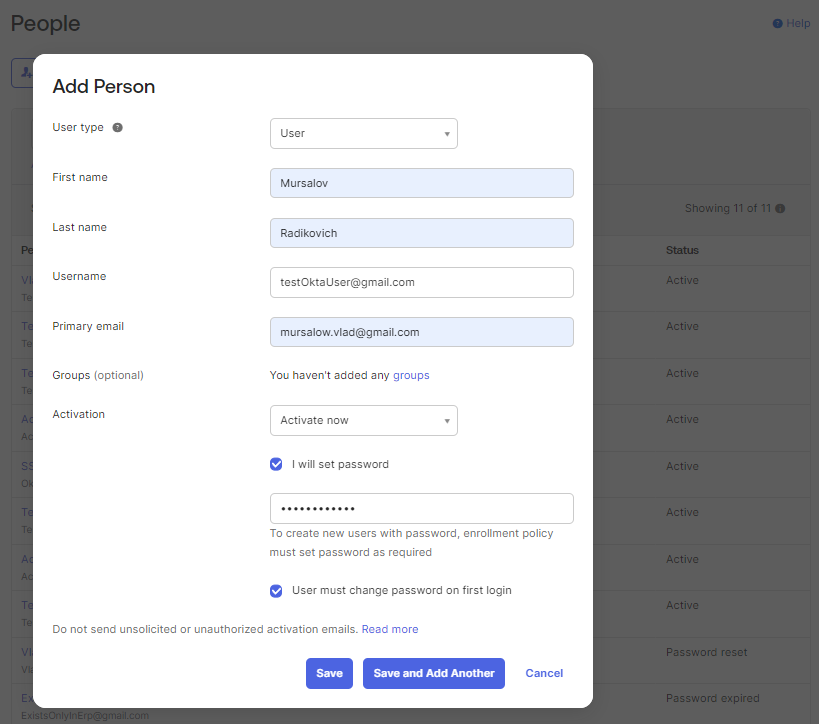
Далі для застосунку потрібно налаштувати URL-адреси. Ці URL-адреси вказуються для спрямування користувачів після упізнання та успішної або неуспішної автентифікації. Sign-in Redirect URL – це URL, на який користувач буде перенаправлений після успішної аутентифікації на сервері OIDC.

Sign-out Redirect URL - це URL, на який користувач буде перенаправлений після виходу з системи або розлогінення.



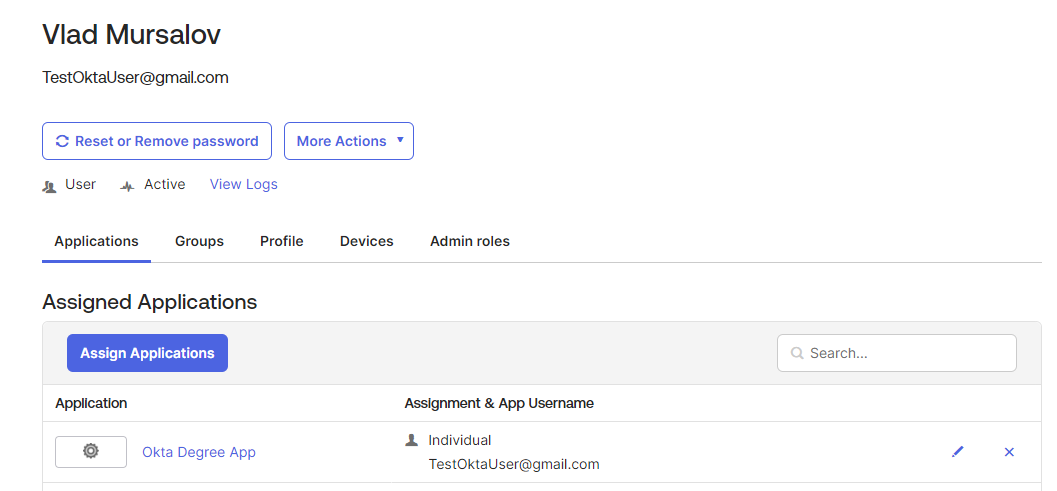
*Мал. 3.2.4. Налаштування Sign-in та Sing-Out Redirect URL*

Тепер додамо першого користувача. Для додавання користувача в Okta нам потрібно увійти до панелі керування Okta та перейти до вкладки "People". Натиснути кнопку "Add Person" та заповнити наступну форму, де потрібно вказати необхідні дані:



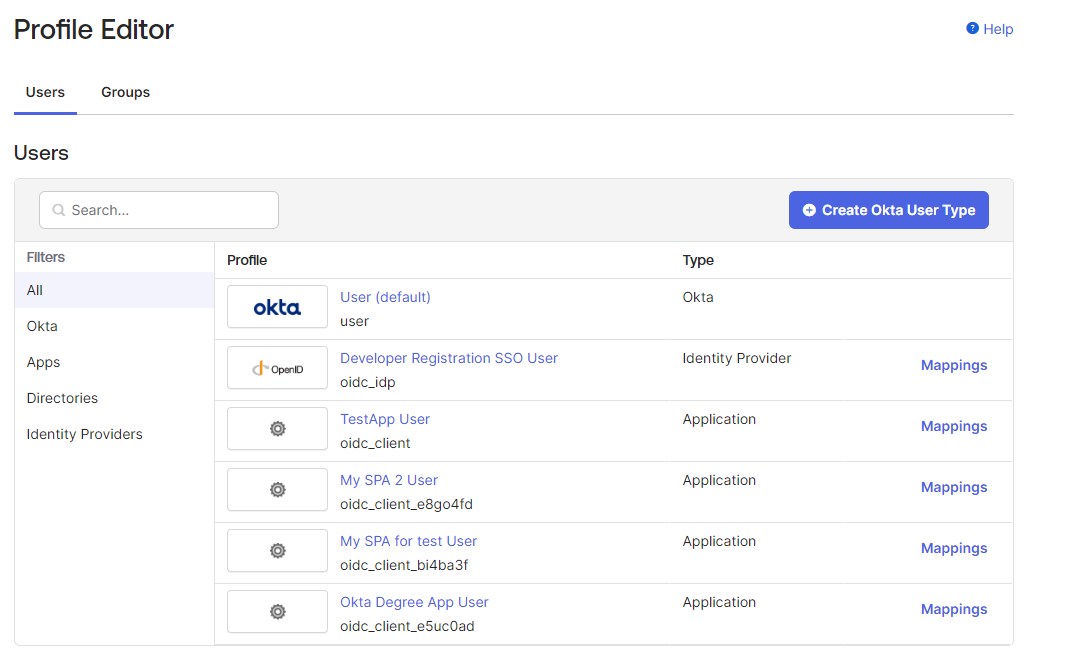
*Мал. 3.2.5. Створення користувача в провайдері*

Далі потрібно додати цього користувача до цієї інтеграції, щоб вім мав змогу через цю інтеграцію автентифікуватися. Спочатку потрібно відкрити потрібний додаток у вкладці "Applications". Потім перейти до вкладки "Assignments" у налаштуваннях застосунка. Тут можна додати користувачів, вибравши їх із списку. Можемо бачити наступне:



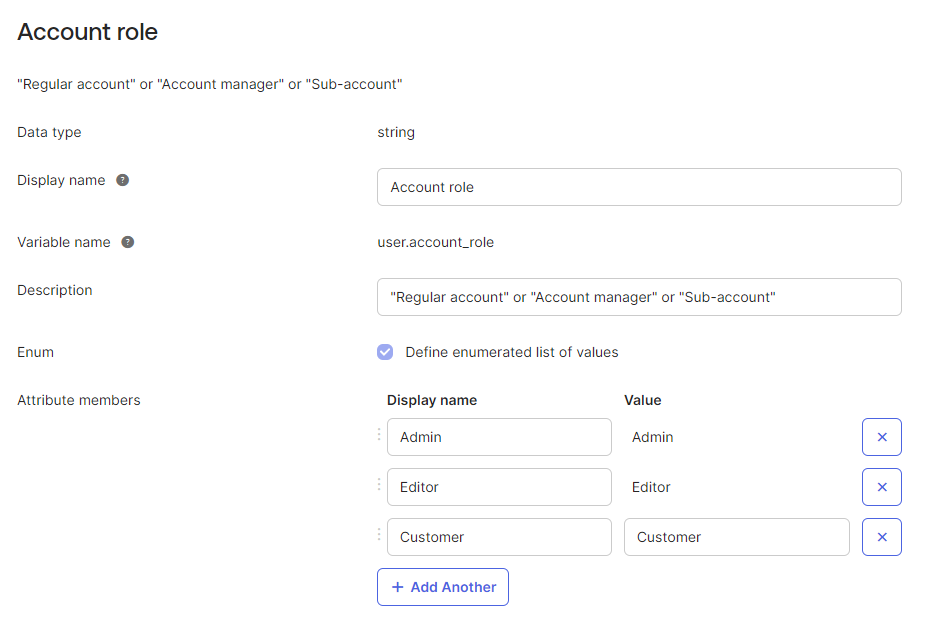
*Мал. 3.2.6. Прив’язка користувача до відповідного додатку*

Тепер потрібно додати користувачу роль в інтернет магазині. Окта дозволяє надати користувачу роль в адмін директорії але ц роль не передається в Id токен. Але рішення є, в окті можна сконфігурувати свої клейми. Потрібно перейти по шляху Directory > Profile Editor:



*Мал. 3.2.7. Створення користувача в провайдері*

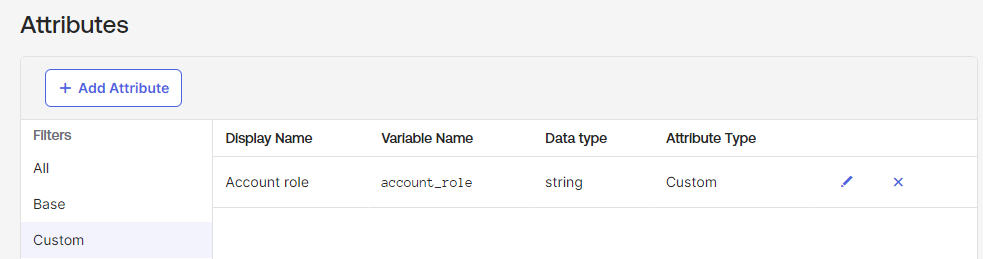
Можемо бачити тут профілі користувачів для всіх інтеграції які створені в директорії. Спочатку нам порібно відредагувати профіль користувача за замовчуванням. Цей профіль є базою для всіх інтеграцій, це користувач з найбільшою кількістью даних в директорії. Потім інтеграції вибирають потрібні їм дані з нього. Отже, створюємо новий клейм account\_role:



*Мал. 3.2.8. Створення користувача в провайдері*

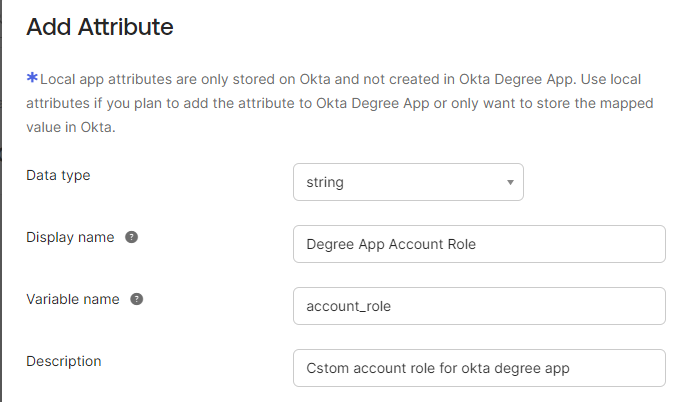
Налаштовуємо тип дляних як string і для зручності заздалегідь визначаємо 3 ролі,

Після чогоможемо бачити клейм в профілі користувача за амовчуванням:



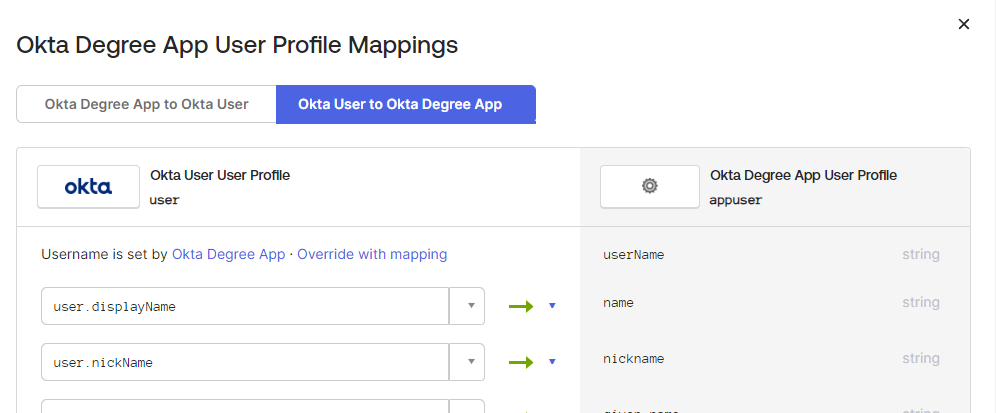
*Мал. 3.2.9. Створення користувача в провайдері*

Тепер портібно визначити такий атрибут персонально для нашої інтеграції. Заходимо в профіль користувача інтеграції, додаємо атрибут з іменем змінної account\_role, описом та читабельним іменем:



*Мал. 3.2.10. Створення користувача в провайдері*

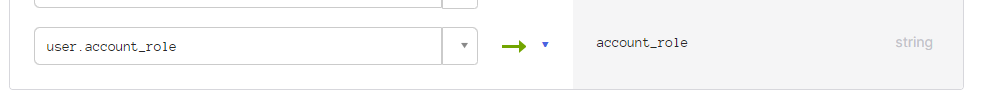
І нарешті потрібно зіставити атрибут профілю користувача за замовчуванням з профілем користувача нашої інтеграції. Для цього нам потрібно зайти по тому ж шляху та натиснути кнопку “Mappings”:



*Мал. 3.2.11. Створення користувача в провайдері*

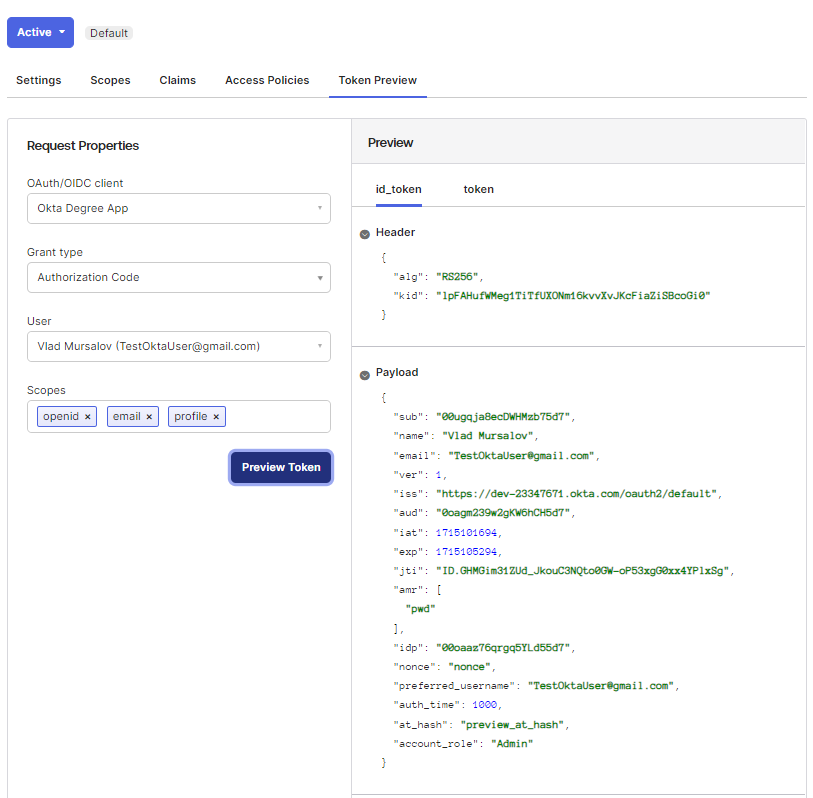
Можемо бачити 2 таби: перша це сторінка з зіставлення атрибутів користувача інтеграції з атрибутом користувача за замовчуванням, друга – навпаки.

Відповідно зіставляємо 2 створених атрибути.



*Мал. 3.2.12. Створення користувача в провайдері*

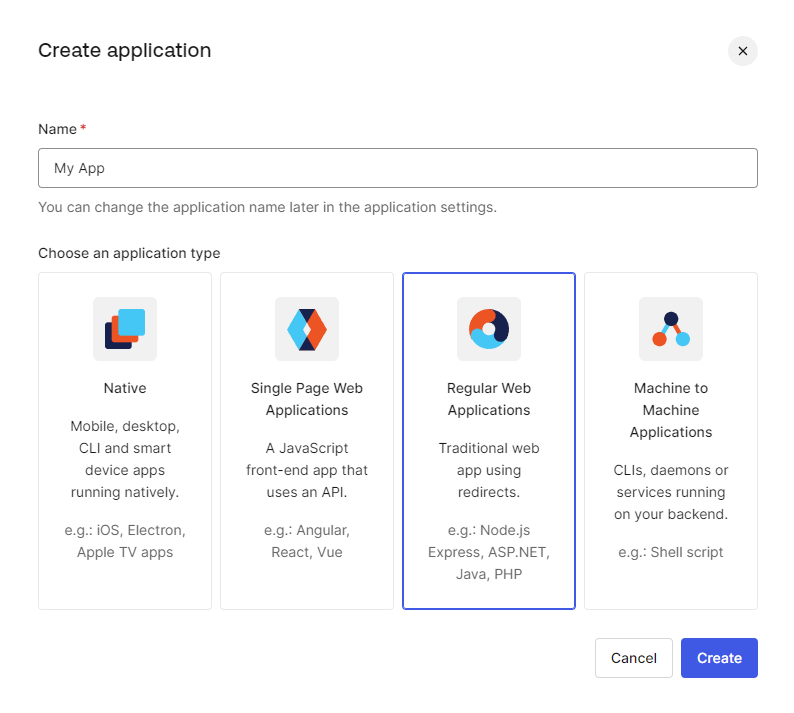
На цьому налаштування можна вважати закінченим. Тепер можемо побачити структуру Id токену для цього користувача при автентифікації до цього додатку. Треба перейти в Security > Api > Authorization servers. Перейти в потрібний нам сервер та обрати фічу Token Preview. Після чого вибираємо створений додаток, тип автентифікації – Authorization code, тільки що створеного користувача та передаємо потрібні скоупи. Переконуємось інформація в токені що токені вірна.



*Мал. 3.2.13. Прив’язка користувача до відповідного додатку*

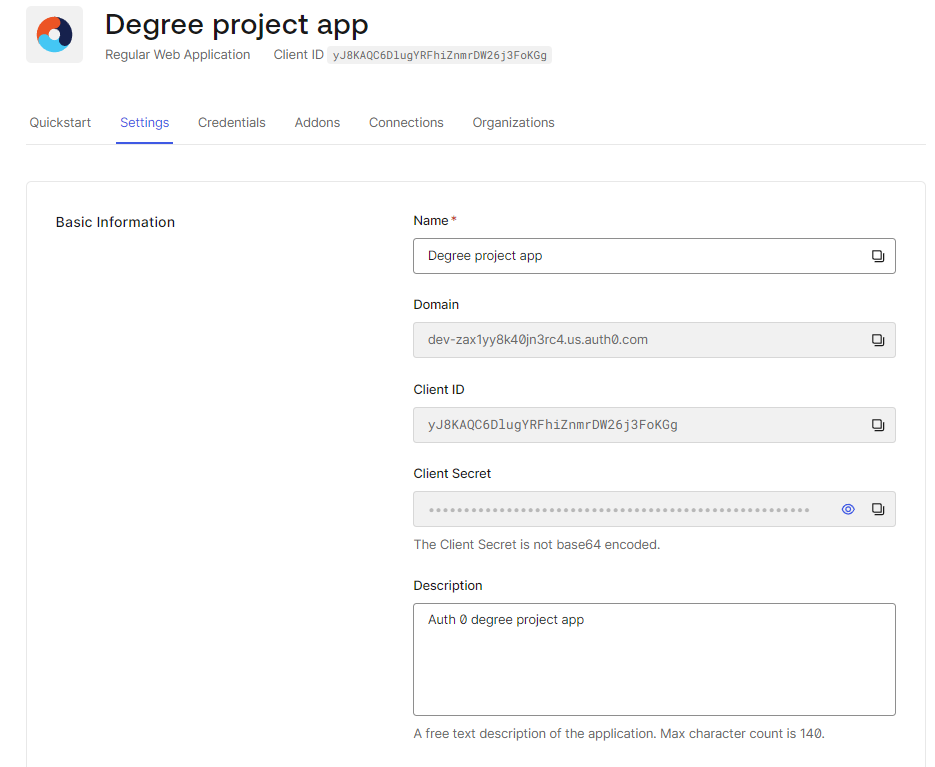
## **3.3. Налаштування Auth0 як серверу автентифіації**

Як і в Окті, потрібно пройти базову автентифікацію на сервісі та створити першу директорію. Потім потрібно створити додаток з викристанням шаблону “Regular Web Applications”:



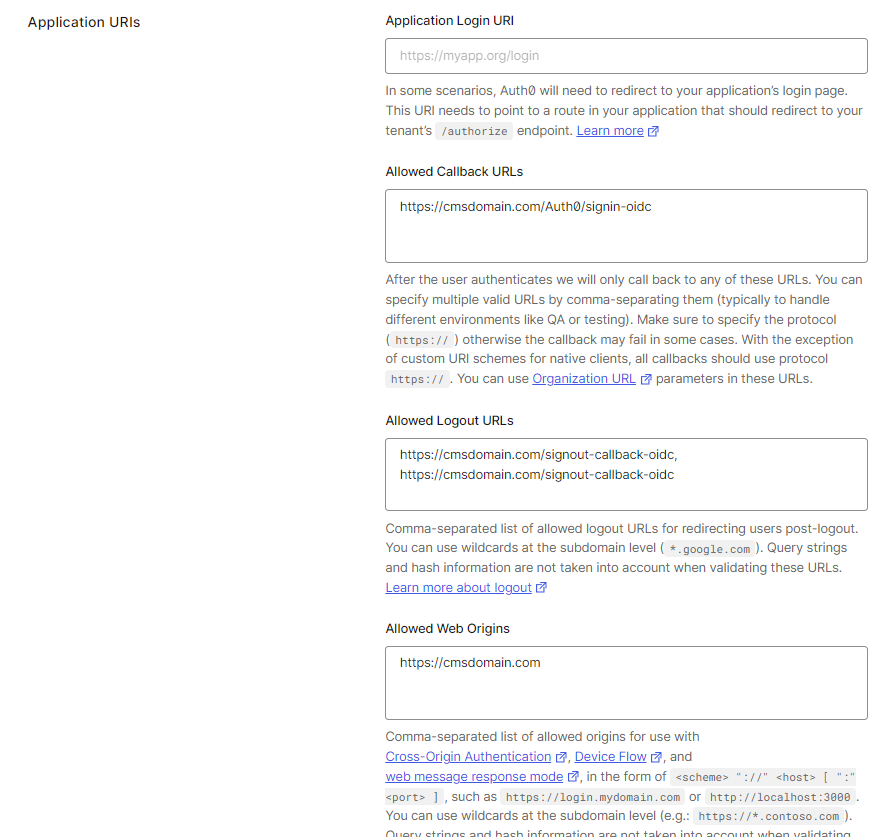
*Мал. 3.3.1. Створення нового додатку-інтеграції для Auth0*

Після створення додатку-інтеграції, відповідно можемо бачити домен нашого серверу автентифікації, ідентифікатор цього додатку та секрет:



*Мал. 3.3.2. Дані для підключення магазину до додатку Auth0*

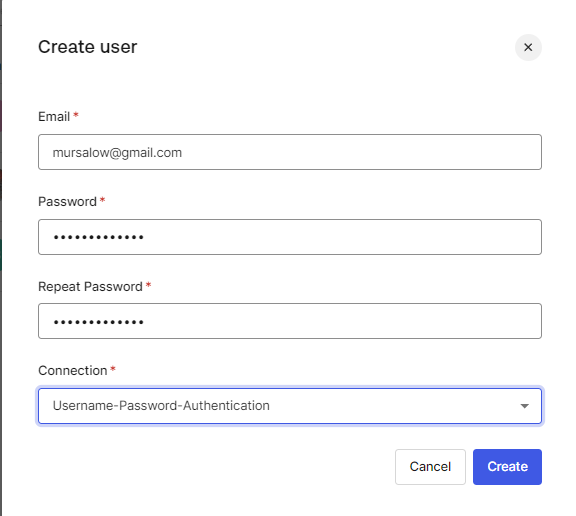
Відповідно до протоколу потрібно ввести маршрути для взаємодії з Auth0 та інтернет магазину. Вони будуються за таким же принципом як і для Окти:



*Мал. 3.3.3. Налаштування маршрутів для працездатності OIDC*

**Створення користувачів:**

Переходимо в User management > Users та тиснемо на кнопку Create new user та вводимо необхідні дані:



*Мал. 3.3.4. Створення користувачів в Auth0*

**Налаштування додаткових клеймів в Id токен:**

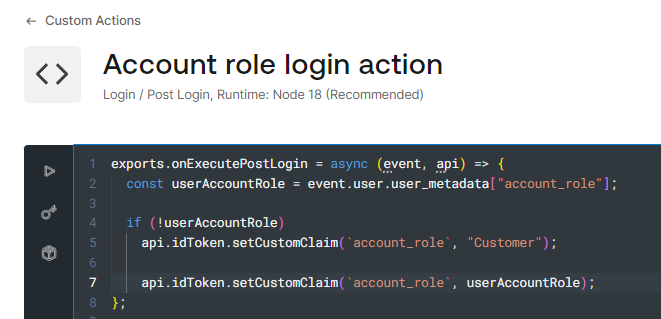
Як і в Окті, роль користувача налаштувати можна, але ця роль розповсюджується на адміністрування в директорії і не передається в Id токен. Тому як і в випаду з Октою нам треба створити власний клейм.

Auth0 має досить зручну систему для модифікацій ходу автентифікації яка називається “Actions”. Кожна подія це програмований код на мові JavaScript. Щоб додати додаткову інформацію в Id токен, переходимо в профіль користувача, спускаємось на до розділу “Metadata” та вводимо додаємо account\_role клейм з відповідним значенням до метадати користувача в форматі JSON:



*Мал. 3.4.1. Додавання ролі в метадату*

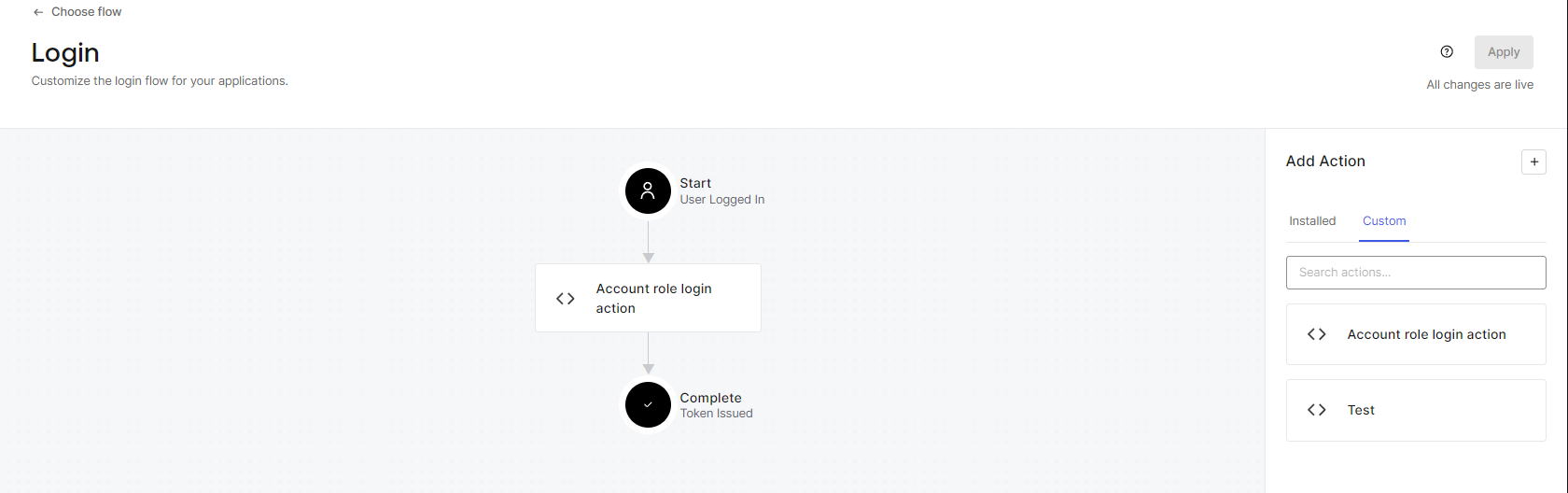
Переходимо в Actions > Library та створюємо нову подію. Тиснемо кнопкуCreate action та вибираємо опцію Build from scratch. В тіло події додаємо наступний код:



*Мал. 3.4.2. Код події для додавання ролі в токен*

Цей код спробує взяти з метаданих користувача його роль, якщо її адміністратор не вказав – будемо вважати що це звичайний покупець.

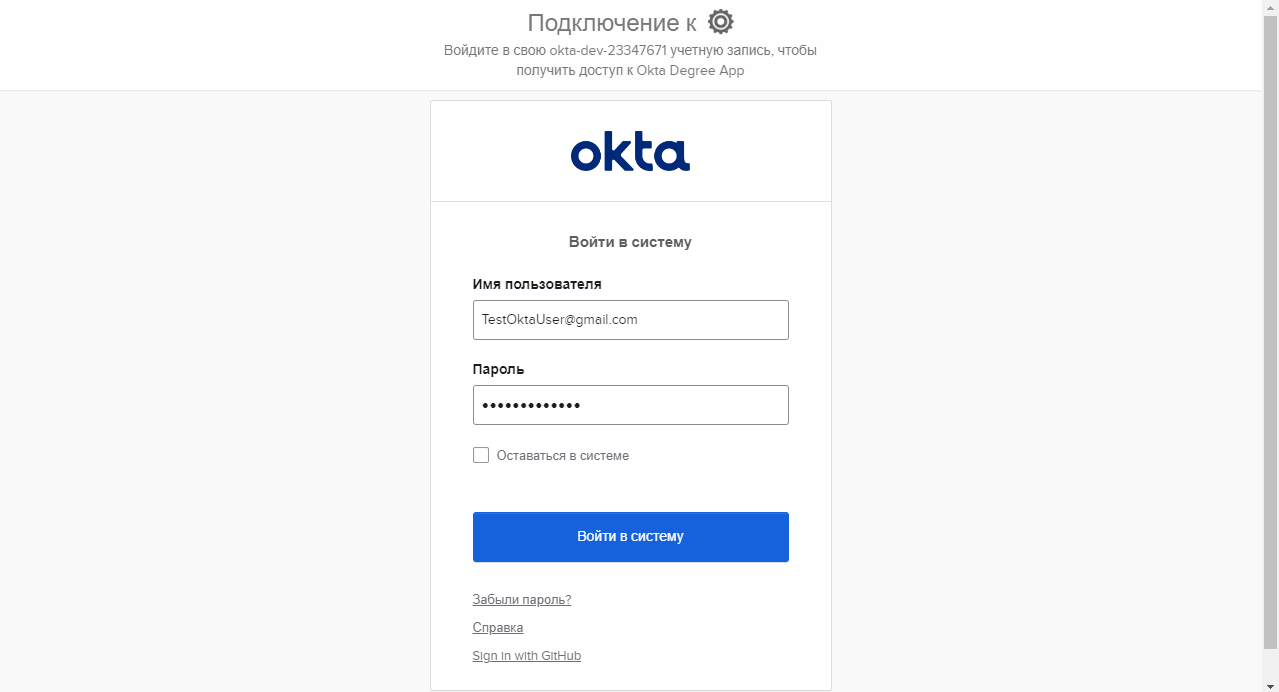
Далі переходимо в Actions > Flows. Обираємо процес Login. В правій панелі “Add action” переходимо в Cumstom табу та обираємо створену подію. Перетягуємо подію між початком та кінцем автентифікації користувача.



*Мал. 3.4.3. Додавання події в процес автентифікації користувача*

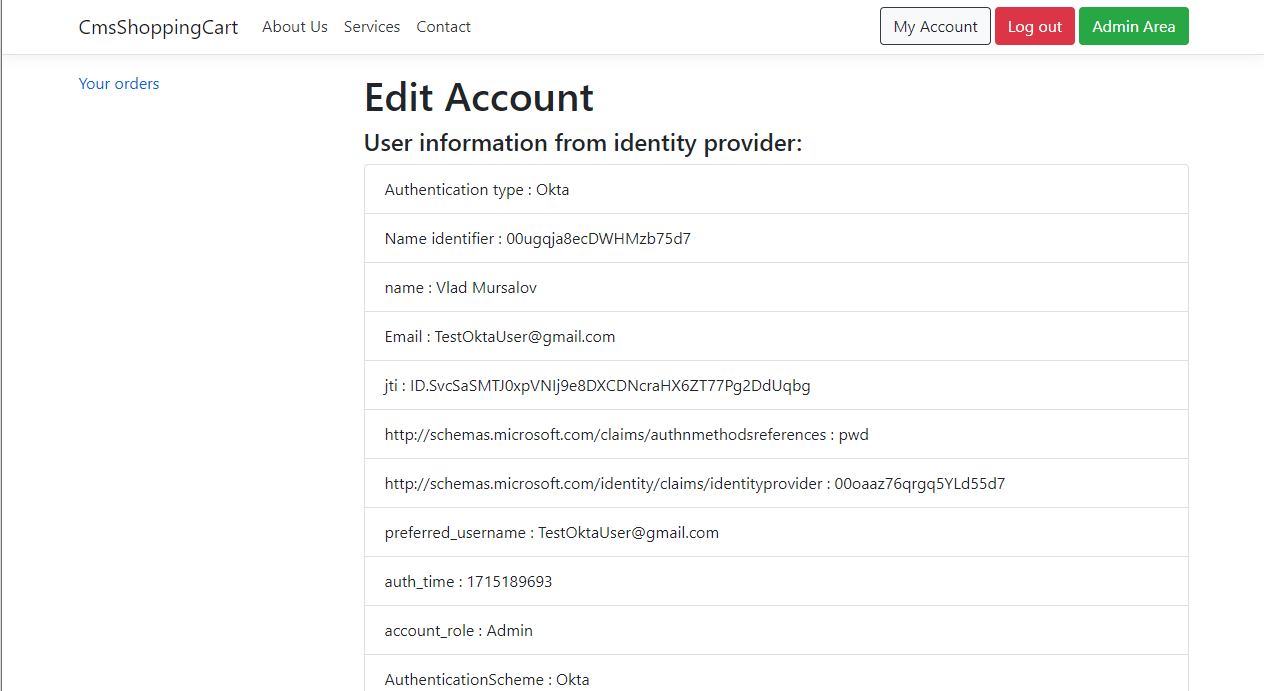
## **3.4 Тестування автентифікації за допомогою Okta**

Спочатку перевіримо чи можлива успішна автентифікація користувача. В інтернет магизині, переходимо на сторінку авторизації, натискаємо кнопку «Login with Okta» на сторінці авторизації, потрапляємо уже на сторінку автентифікації Окти:



*Мал. 3.4.1. Сторінка авторизації Okta*

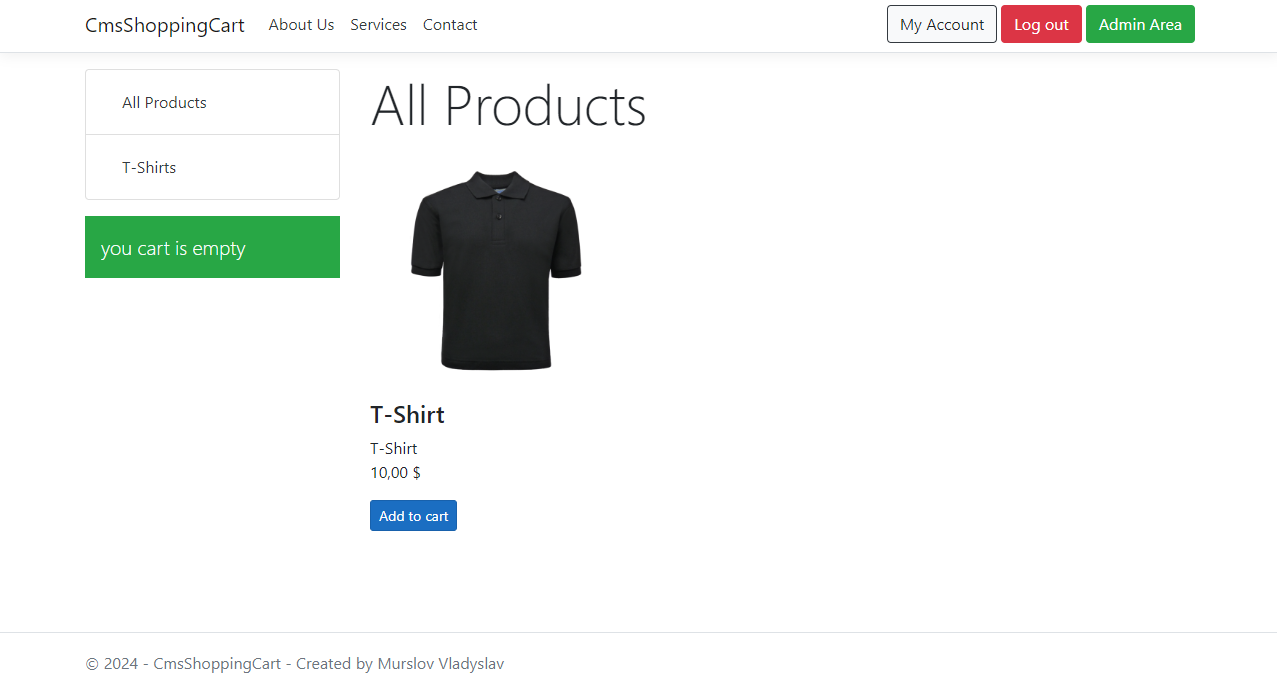
Стандартно вводимо пошту та пароль на стороні Окти. Потім нас запитають дозволи на доступ до інформації профілю та пошти, дозволяємо це, після чого нас успішно автентифікувало та перенаправило на сторінку профілю:



*Мал. 3.4.2. Інформація про авторизованого окристувача*

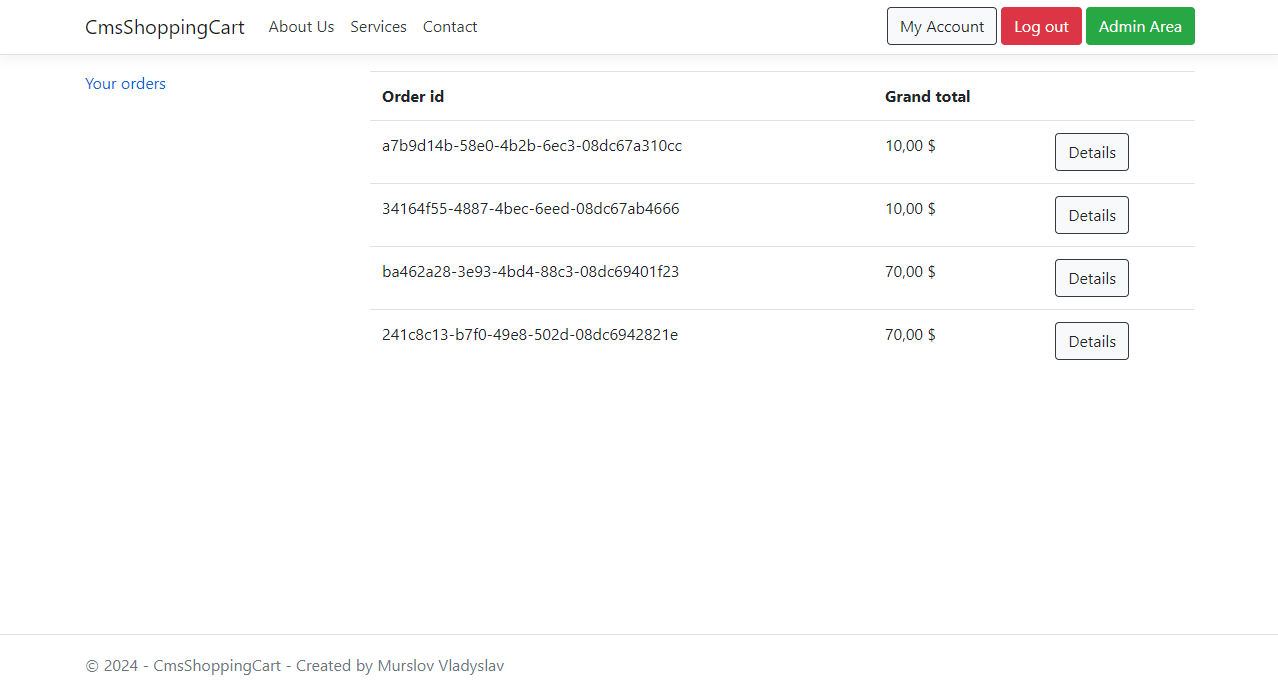
Для демонстрації ми навмисно вивели всю доступну інформацію яку отримує інтернет магазин. Можемо бачити що схема автентифікації – Okta та бачимо створений нами клейм прийшов з правильною ролью.

Тепер перевіримо базу даних, спробуємо купити декілька продуктів, Переходимо на сторінку з продуктом та замовляємо його декіль каразів в різних кількостях:



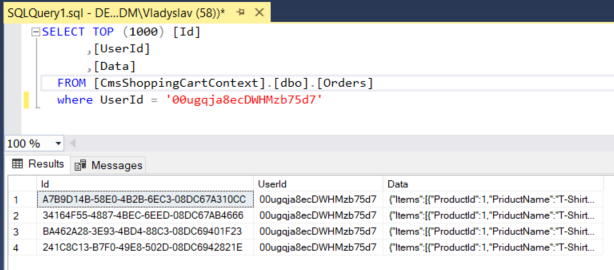
*Мал. 3.4.3. Сторінка з продуктом*

Переходимо на сторінку свого профілю, та можемо бачити 4 замовлення:



*Мал. 3.4.4. Сторінка з замовленнями*

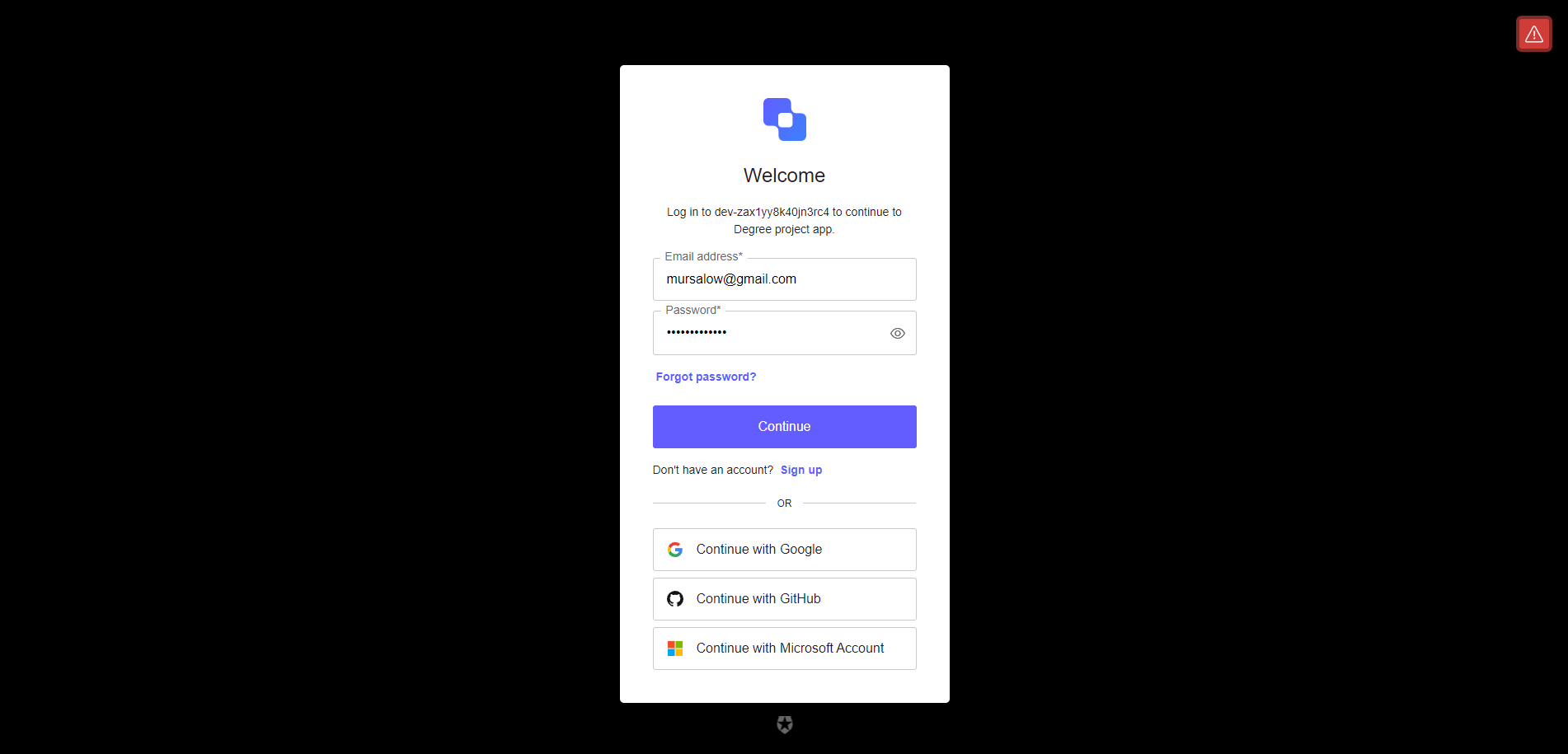
Ці ж 4 замовлення можемо побачити в базі даних, що значить що вона справно функціонує:



*Мал. 3.4.5. Замовлення в базі даних*

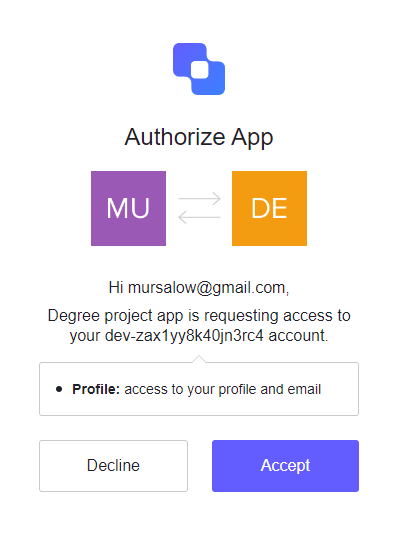
## **3.5 Тестування автентифікації за допомогою Auth0**

Нак само, перевіримо чи можлива успішна автентифікація користувача. В інтернет магизині, Натискаємо кнопку «Login with Auth0» та потрапляємо на сторінку автентифікації Auth0:



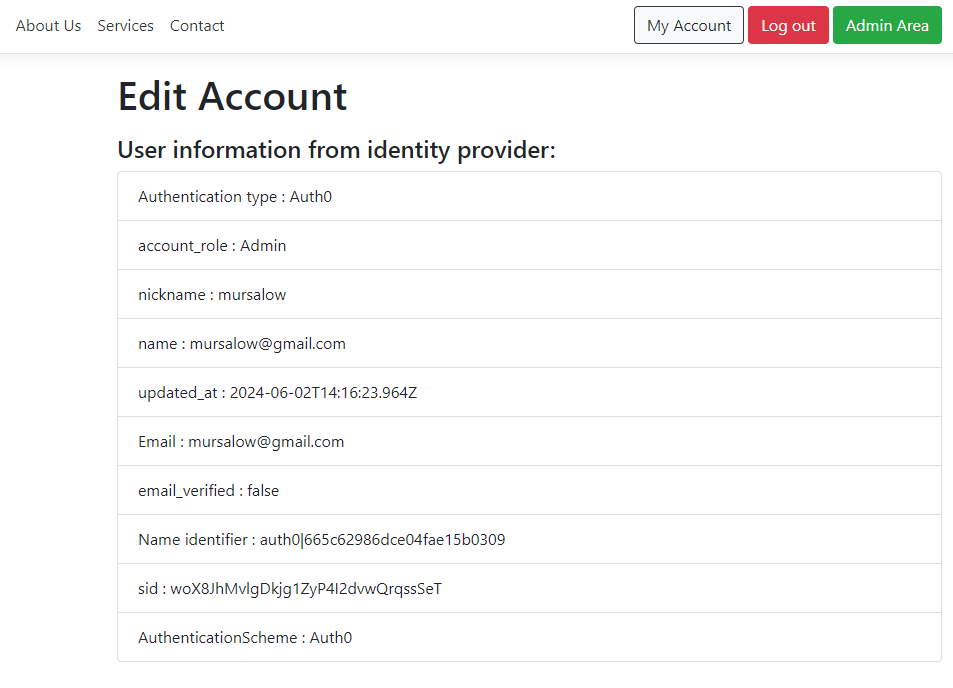
*Мал. 3.4.1. Сторінка автентифікації в Auth0*

Вводимо наші дані, після чого нас запитають дозволи на доступ до інформації профілю та пошти, дозволяємо це, після чого нас успішно автентифікувало та перенаправило на сторінку профілю.



*Мал. 3.4.2. Прийняття скоупів на доступ інформації в провайдері*

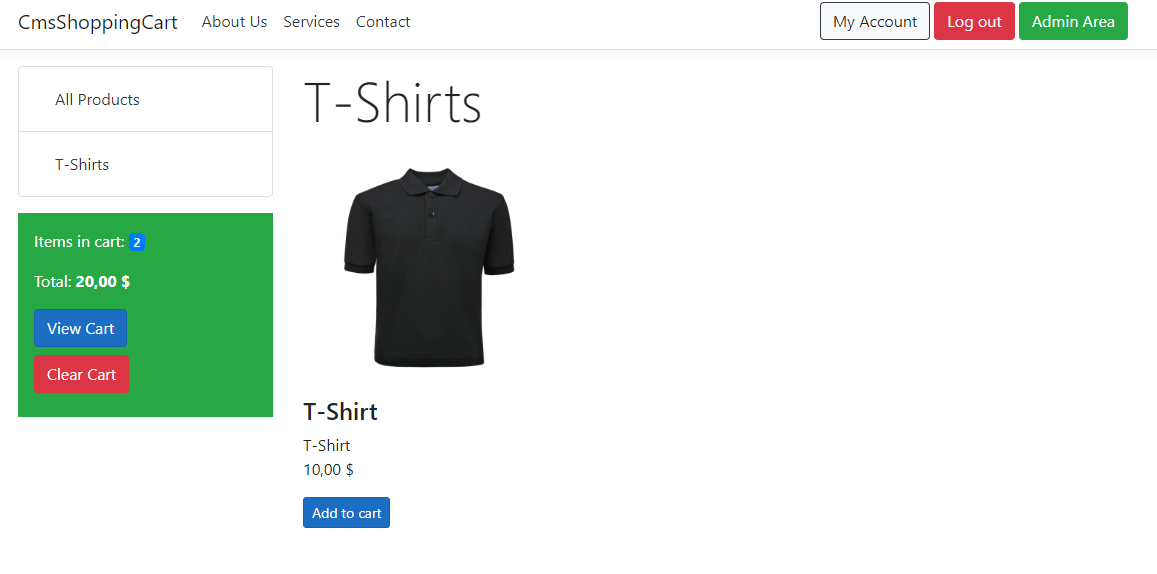
Після успішної автентифікації нас Auth0 з результатами автентифікації перенаправляє на сторінку профілю користувача з актуальною інформацію:



*Мал. 3.4.3. Отримана інформація користувача з Auth0*

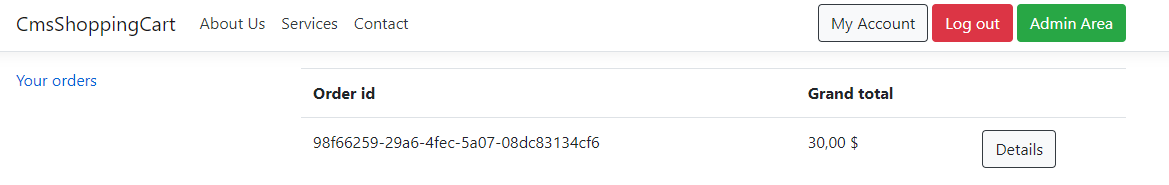
Бачимо що прийшла актуальна інформація, схема автентифікації відповідає налаштованому IdP і ми отримали потрібну нам роль налашнована роль.

Спробуємо замовити 2 продукти, переходимо на сторінку продукту, замовляємо декілька штук.



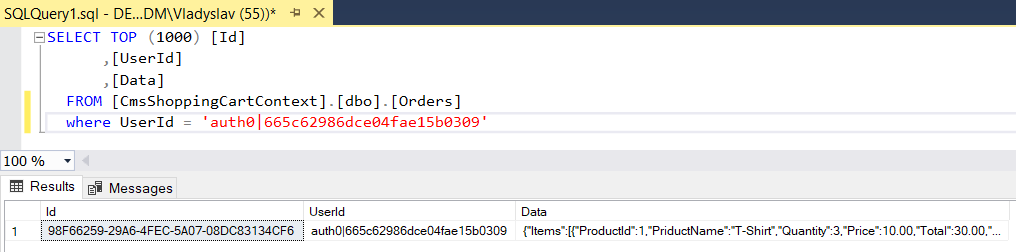
*Мал. 3.4.4. Замовлення продукту в інтернет магазині*

Після оформлення замовлення, можемо бачити його в списку замовлень:



*Мал. 3.4.5. Замовлення на сторінці замовлення*

Відповідний запис в базі даних:



*Мал. 3.4.6. Замовлення в базі даних*

Можемо бачити що це саме те замовлення по його Id. Таков можемо бачити що UserId в базі співпадає з тим Id що ми отримуємо з серверу автентифікації Auth0.

**Висновок**: Проведене тестування показало, що розроблена система динамічного додавання систем ідентифікації працює як задумано. Автентифікація користувачів через такі провайдери, як Okta та Auth0, є надійним та ефективним способом забезпечення безпеки і доступу до веб-додатків.

Під час тестування було продемонстровано:

1. Процедуру налаштування та інтеграції Okta і Auth0 як серверів автентифікації.

2. Процес автентифікації користувачів через обидва провайдери, включаючи прийняття необхідних скоупів та отримання коректних ролей.

3. Успішне створення та обробку замовлень.

## **Висновки до розділу 3**

Третій розділ зосереджувався на тестуванні інфраструктури розробленого рішення. Було проведено налаштування та тестування серверів автентифікації Okta та Auth0, а також перевірку роботоздатності інтернет-магазину. Тестування показало, що інтеграція з сервісами ідентифікації забезпечує надійний та безпечний доступ користувачів до ресурсів магазину. Використання протоколу OIDC дозволило значно спростити процес автентифікації та підвищити зручність користувачів. Також було перевірено функціонування мультифакторної автентифікації та механізмів розмежування ролей користувачів. Результати тестування підтвердили ефективність розробленого рішення та його відповідність вимогам проекту, що забезпечує безпечний та зручний доступ до ресурсів інтернет-магазину.

# **ВИСНОВКИ**

В результаті проведеної роботи вдалося досягти всіх поставлених на початку завдань. Було розроблено проект підсистеми автентифікації онлайн-магазину на базі протоколу Open ID Connect та різних провайдерів ідентифікації. У ході роботи проведено детальний аналіз існуючих методів автентифікації, вивчено їх переваги та недоліки. Особливу увагу приділено дослідженню протоколу OpenID Connect, його можливостям та перевагам порівняно з іншими протоколами, такими як SAML.

Розроблено архітектуру підсистеми автентифікації для онлайн-магазину, яка враховує обраних провайдерів ідентифікації, таких як Okta та Auth0. В результаті обрано оптимальне рішення для інтеграції, яке забезпечує високу безпеку, зручність використання та відповідність сучасним вимогам до автентифікації.

Проектування рішення включало формування вимог до системи відповідно до бізнес логіки інтернет-магазину, розробку взаємодії між основними компонентами системи та імплементацію рішення в існуючу кодову базу. Зокрема, реалізовано динамічну реєстрацію провайдерів ідентифікації, автентифікацію через IdP та розмежування ролей автентифікованих користувачів.

Проведене тестування інфраструктури підтвердило працездатність розробленого рішення. Налаштовано сервіси Okta та Auth0 як сервери автентифікації, і проведено тестування автентифікації за допомогою цих сервісів. Результати тестування показали, що система здатна забезпечити надійну автентифікацію користувачів та ефективно управляти доступом до ресурсів онлайн-магазину.

Таким чином, всі поставлені завдання були успішно виконані. Розроблене рішення відповідає вимогам до сучасних систем автентифікації, забезпечує високий рівень безпеки, зручність використання та гнучкість у налаштуванні. Проект підсистеми автентифікації на базі протоколу Open ID Connect та різних провайдерів ідентифікації може бути рекомендований для впровадження в інших подібних проектах, що сприятиме підвищенню рівня безпеки та зручності для користувачів.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. App integrations | Okta. *Okta Docs*. URL: <https://help.okta.com/en-us/content/topics/apps/apps_apps.htm> (дата звернення: 09.06.2024).
2. Auth0 ASP.NET Core MVC SDK Quickstarts: Add Login to your ASP.NET MVC application. *Auth0 Docs*. URL: <https://auth0.com/docs/quickstart/webapp/aspnet-core/interactive> (дата звернення: 09.06.2024).
3. Auth0 ASP.NET Core MVC SDK Quickstarts: Add Login to your ASP.NET MVC application. *Auth0 Docs*. URL: <https://auth0.com/docs/quickstart/webapp/aspnet-core/interactive> (дата звернення: 09.06.2024).
4. Authorization code flow - NHS England Digital. NHS England Digital. URL: https://digital.nhs.uk/services/care-identity-service/applications-and-services/cis2-authentication/guidance-for-developers/detailed-guidance/authorization-code-flow (дата звернення: 09.06.2024).
5. Authorize with a specific scheme in ASP.NET Core. *Microsoft Learn: Build skills that open doors in your career*. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/limitingidentitybyscheme?view=aspnetcore-8.0> (дата звернення: 09.06.2024).
6. Customize tokens returned from Okta with custom claims | Okta Developer. *Home | Okta Developer*. URL: <https://developer.okta.com/docs/guides/customize-tokens-returned-from-okta/main/> (дата звернення: 09.06.2024).
7. Final: OpenID Connect Core 1.0 incorporating errata set 2. *OpenID - OpenID Foundation*. URL: <https://openid.net/specs/openid-connect-core-1_0.html> (дата звернення: 09.06.2024).
8. Overview of ASP.NET Core Authentication. *Microsoft Learn: Build skills that open doors in your career*. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-8.0> (дата звернення: 09.06.2024).
9. Overview of ASP.NET Core MVC. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0 (дата звернення: 09.06.2024).
10. RFC 6749. The OAuth 2.0 Authorization Framework. На заміну RFC 5849 ; чинний від 2012-10-01. Вид. офіц. Internet Engineering Task Force (IETF), 2012. URL: <https://www.rfc-editor.org/rfc/rfc6749> (дата звернення: 09.06.2024).
11. Sign users in to your web app using the redirect model | Okta Developer. *Home | Okta Developer*. URL: <https://developer.okta.com/docs/guides/sign-into-web-app-redirect/asp-net-core-3/main/> (дата звернення: 09.06.2024).
12. What is OpenID vs SAML? Find out the differences - Auth0. *Auth0*. URL: <https://auth0.com/intro-to-iam/saml-vs-openid-connect-oidc> (дата звернення: 09.06.2024).
13. Working Draft 07. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Чинний від 2015-09-15. Вид. офіц. OASIS, 2015. URL: <https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf> (дата звернення: 09.06.2024).
14. Учасники проектів Вікімедіа. Програмне забезпечення як послуга – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Програмне\_забезпечення\_як\_послуга (дата звернення: 09.06.2024).
15. What is IDaaS? Understanding Identity as a Service and Its Applications | Okta. Employee and Customer Identity Solutions | Okta. URL: https://www.okta.com/identity-101/idaas/ (дата звернення: 09.06.2024).