Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеню бакалавра

студента *Савченка Максима Андрійовича*
академічної групи *125-20-2*
спеціальності *125 Кібербезпека*
спеціалізації[1]
за освітньо-професійною програмою *Кібербезпека*
на тему *Інтеграція багатофакторної аутентифікації у веб-додаток на C#*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	проф. Магро. В.І.			
розділів:				
спеціальний	проф. Магро. В.І.			
економічний	к.е.н., доц. Пілова Д.П.	90	відмінно	
Рецензент				
Нормоконтролер	ст. викл. Мєшков В.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
безпеки інформації та телекомунікацій
_____ д.т.н., проф. Корнієнко В.І.

«_____»_____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавра

студенту ***Савченку Максим Андрійовичу*** **академічної групи** ***125-20-2***
(прізвище ім'я по-батькові) (шифр)

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

на тему ***Інтеграція багатофакторної аутентифікації у веб-додаток на C#***

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 № 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз сучасних проблем аутентифікації та впровадження багатофакторної аутентифікації.	02.06.2024
Розділ 2	Дослідження та вибір фреймворків та бібліотек для розробки аутентифікації додатку, інтеграція багатофакторної аутентифікації та тестування цього додатку.	10.06.2024
Розділ 3	Розрахунок витрат, оцінка можливого збитку від атаки та аналіз економічної ефективності системи.	20.06.2024

Завдання видано _____ Валерій МАГРО
(підпис керівника) (ім'я, прізвище)

Дата видачі: 01.06.2024р.
Дата подання до екзаменаційної комісії: 04.06.2024р.
Прийнято до виконання _____ Максим САВЧЕНКО
(підпис студента) (ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 57 с., 14 рис., 1 табл., 4 додатка, 20 джерел.

Об'єкт досліджень: інтеграція багатофакторної аутентифікації у веб-додаток.

Предмет розробки: методи та засоби захисту багатофакторної аутентифікації веб-додатку.

Мета роботи: розробка та впровадження ефективних методів інтеграції багатофакторної аутентифікації, які зможуть запобігти несанкціонованому доступу до систем, забезпечити цілісність та конфіденційність інформації.

В першому розділі було розглянуто стан питання та постановку задачі кваліфікаційної роботи, також проведено аналіз ризиків і загроз безпеки, що пов'язані з аутентифікацією.

В спеціальній частині було детально описано технічні аспекти реалізації багатофакторної аутентифікації. Описано архітектуру веб-додатку та процес розробки аутентифікації користувача. Проведено тестування багатофакторної аутентифікації для забезпечення її належного функціонування та захисту користувачів від можливих загроз.

В економічному розділі розраховані основні показники, які показують економічну доцільність та ефективність впровадження розроблених методів захисту інформації.

Практична цінність розробки полягає у забезпеченні ефективного захисту вебресурсів через інтеграцію багатофакторної аутентифікації, що запобігає несанкціонованому доступу до облікових записів користувачів та забезпечує цілісність і конфіденційність інформації.

КІБЕРБЕЗПЕКА БЕЗПЕКА ІНФОРМАЦІЇ, БАГАТОФАКТОРНА АУТЕНТИФІКАЦІЯ, ЗАХИСТ ВЕБ-ДОДАТКІВ, ЗАХИСТ ВІД АТАК, ЕКОНОМІЧНА ДОЦІЛЬНІСТЬ.

ABSTRACT

Explanatory note: 57 pp., 14 pic, 1 table, 4 addition, 20 sources,

Object of research: integration of multi-factor authentication in a web application.

Development subject: methods and means of protection of multi-factor authentication of a web application.

The purpose of the project: development and implementation of effective methods of integration of multi-factor authentication, which will be able to prevent unauthorized access to systems, ensure the integrity and confidentiality of information.

In the first section, the state of the issue and the setting of the task of the qualification work were considered, as well as the analysis of risks and security threats related to authentication was carried out.

The technical aspects of implementing multifactor authentication were described in detail in a special part. The web application architecture and user authentication development process are described. Multifactor authentication has been tested to ensure its proper functioning and protect users from possible threats.

In the economic section, the main indicators are calculated, which show the economic feasibility and effectiveness of the implementation of the developed information protection methods.

The practical value of the development lies in providing effective protection of web resources through the integration of multi-factor authentication, which prevents unauthorized access to user accounts and ensures the integrity and confidentiality of information.

CYBER SECURITY INFORMATION SECURITY, MULTI-FACTOR AUTHENTICATION, PROTECTION OF WEB APPLICATIONS, PROTECTION AGAINST ATTACKS, ECONOMIC FEASIBILITY.

СПИСОК УМОВНИХ СКОРОЧЕНЬ

БФА та MFA – багатофакторна аутентифікація;

2FA – Two Factor Authentication (двофакторна аутентифікація);

OTP – One-time password (одноразовий пароль);

JWT – JSON Web Token;

DI – Dependency Injection (контейнер залежностей);

Malware – шкідливе програмне забезпечення;

DNS - Domain Name System;

URL - Uniform Resource Locator;

SSO - Single sign-on;

USB - Universal Serial Bus;

SMS - Short Message Service;

API - Application Programming Interface;

HTTP - Hyper Text Transfer Protocol.

ЗМІСТ

ВСТУП

У сучасних умовах інформаційної безпеки, питання захисту даних стає надзвичайно важливим. З кожним роком зростає кількість кіберзлочинів, спрямованих на отримання несанкціонованого доступу до конфіденційної інформації. Одним із ефективних методів підвищення рівня безпеки є впровадження багатофакторної аутентифікації, яка значно ускладнює можливість несанкціонованого доступу навіть у разі компрометації одного з факторів аутентифікації. Впровадження багатофакторної аутентифікації у веб-додатках є важливим кроком до забезпечення надійного захисту даних користувачів.

Метою даної роботи є розробка та впровадження багатофакторної аутентифікації у веб-додаток з використанням фреймворків на C#.

Об'єктом дослідження є процес аутентифікації користувачів у веб-додатках. Предметом дослідження є методи та інструменти впровадження багатофакторної аутентифікації у веб-додаток з використанням фреймворків.

У роботі використані методи аналізу, порівняння, системного підходу, а також методи програмування для реалізації і тестування програмного продукту. Аналіз існуючих методів багатофакторної аутентифікації здійснювався шляхом вивчення наукової літератури та технічної документації. Розробка і тестування рішення виконувалися за допомогою сучасних інструментів та фреймворків.

Практичне значення роботи полягає у розробці методології впровадження багатофакторної аутентифікації у веб-додатки, яка може бути використана розробниками для підвищення безпеки своїх продуктів. Запропоновані рішення можуть бути адаптовані до різних веб-додатків та інтегровані у існуючі системи для забезпечення надійного захисту даних користувачів.

РОЗДІЛ 1. СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз ризиків і загроз безпеки, що пов'язанні з аутентифікацією

Аутентифікація є критично важливим елементом у забезпеченні безпеки інформаційних систем, оскільки вона дозволяє підтвердити ідентичність користувачів, які намагаються отримати доступ до системи. Однак цей процес може бути вразливим до різних атак і загроз. Наприклад, однією з найбільш поширених загроз є фішинг, коли зловмисники використовують підроблені веб-сайти або електронні листи для отримання облікових даних користувачів. Крім того, існують ризики, пов'язані з викраденням паролів через шкідливе програмне забезпечення або використання слабких паролів, які легко зламати за допомогою атак грубої сили або словникових атак.

Також є загрози, що виникають через соціальну інженерію, коли зловмисники маніпулюють користувачами, щоб ті розкрили свої облікові дані. Ще однією проблемою є перехоплення даних під час передачі, що може призвести до компрометації облікових записів. Уразливості в програмному забезпеченні, яке використовується для аутентифікації, також можуть бути використані для отримання несанкціонованого доступу до системи.

Для вирішення цих проблем може використовуватись багатофакторна аутентифікація, яка значно підвищує рівень безпеки. MFA вимагає від користувача надання двох або більше різних факторів для підтвердження своєї ідентичності. Ці фактори можуть включати щось, що користувач знає (наприклад, пароль), щось, що він має (наприклад, смартфон або токен), і щось, що він є (біометричні дані, такі як відбитки пальців або розпізнавання обличчя). Використання різних факторів ускладнює завдання для зловмисників, оскільки навіть якщо один з факторів буде скомпрометований, інші все ще захищатимуть обліковий запис.

Таким чином, розглянувши основні ризики та загрози, пов'язані з аутентифікацією, можна зробити висновок, що багатофакторна аутентифікація є

ефективним способом їх вирішення. Вона забезпечує додатковий рівень захисту і допомагає мінімізувати ризики, пов'язані з компрометацією облікових даних користувачів.

1.2 Поняття та сутність багатофакторної аутентифікації

Багатофакторна аутентифікація є процесом перевірки автентичності користувача за допомогою двох або більше факторів, що належать до різних категорій. Це дозволяє значно підвищити рівень безпеки системи, оскільки навіть у разі компрометації одного з факторів, зловмисник не зможе отримати доступ до системи без інших необхідних компонентів аутентифікації.

Зазвичай фактори аутентифікації поділяються на три основні категорії.

1. Що знає користувач: паролі, ПІН-коди, відповіді на секретні запитання.
2. Що має користувач: смарт-картки, мобільні пристрої, токени.
3. Що є користувачем: відбитки пальців, розпізнавання обличчя, сканування райдужної оболонки ока.

Сутність БФА полягає в тому, що для успішної аутентифікації користувача необхідно підтвердити його особу за допомогою кількох незалежних факторів. Це робить процес аутентифікації набагато більш надійним порівняно з однофакторною аутентифікацією, де використовується лише один фактор, наприклад пароль.

Процес багатофакторної аутентифікації може бути поділений на декілька етапів.

1. Введення первинного фактору: користувач вводить пароль або ПІН-код.
2. Підтвердження вторинного фактору: користувач надає додатковий фактор, наприклад, вводить код з SMS або використовує відбиток пальця.
3. Перевірка факторів: система перевіряє кожен фактор на відповідність збереженим даним.

4. Надання доступу: у разі успішної перевірки всіх факторів, користувач отримує доступ до системи.

Переваги БФА

- Підвищення рівня безпеки: БФА значно ускладнює доступ до системи для зловмисників.
- Зниження ризиків компрометації даних: використання кількох факторів робить систему менш вразливою до атак типу "фішинг" і "brute force".
- Покращення користувацького досвіду: БФА може забезпечити більш безпечний і водночас зручний доступ до системи.

Недоліки БФА

- Ускладнення процесу аутентифікації: користувачам необхідно виконувати додаткові дії для доступу до системи.
- Збільшення витрат на впровадження: впровадження БФА потребує додаткових ресурсів і інфраструктури.
- Проблеми з доступністю: не всі користувачі можуть мати доступ до необхідних факторів, наприклад, до мобільного пристрою для отримання SMS-кодів.

Також БФА може захистити від наступних атак або вразливостей через потребу декількох факторів.

Фішинг

Фішинг є методом, за допомогою якого зловмисники використовують соціальну інженерію для отримання якогось фактору аутентифікації. Наприклад, зловмисники можуть створити фальшивий вебсайт, який виглядає дуже подібним до справжнього, і обманом змусити користувача ввести свої дані. Ці фальшиві сайти часто супроводжуються електронними листами або повідомленнями, які виглядають як офіційні комунікації від відомих організацій. Користувачів можуть

налякати або спонукати до негайних дій, щоб вони не задумувалися над справжністю повідомлення.

Атаки типу "людина посередині"

Атаки типу "людина посередині" передбачають перехоплення комунікацій між користувачем і системою аутентифікації. Зловмисники можуть створити небезпечну Wi-Fi мережу, до якої користувач підключиться, або використовувати техніки DNS-спуфінгу для перенаправлення користувача на підроблений вебсайт. У цьому випадку зловмисник може бачити та змінювати всі дані, що передаються між користувачем і сервісом, включаючи облікові дані.

Перехоплення SMS

Перехоплення SMS є ще одним способом отримання одноразових паролів. Зловмисники можуть використовувати різні методи для перехоплення SMS, такі як атаки на мобільну мережу або перенаправлення повідомлень. Наприклад, зловмисники можуть обманом отримати дублікати SIM-картки жертви, щоб отримувати всі її SMS.

Malware

Шкідливе програмне забезпечення може бути встановлене на пристрої користувача для перехоплення факторів аутентифікації. Такий malware може записувати введені дані з клавіатури, знімати скріншоти або навіть автоматично передавати аутентифікаційні дані зловмиснику. Крім того, malware може змінювати налаштування системи безпеки, щоб забезпечити постійний доступ до пристрою.

Соціальна інженерія

Соціальна інженерія включає методи обману користувачів або співробітників організацій з метою отримання доступу до їхніх аутентифікаційних даних. Це може включати телефонні дзвінки, підроблені електронні листи або особисті зустрічі, де зловмисник представляється співробітником компанії або іншою довіреною особою. Мета таких атак —

створити довірливу атмосферу і змусити жертву розкрити конфіденційну інформацію.

Біометричні уразливості

Біометричні уразливості також є серйозною загрозою. Хоча біометричні дані, такі як відбитки пальців або розпізнавання обличчя, вважаються надійними методами аутентифікації, вони можуть бути скомпрометовані. Високоякісні зображення або відбитки пальців можуть бути використані для створення фальшивок, які обдурюють біометричні системи. Крім того, якщо біометричні дані зламані, їх не можна змінити, на відміну від паролів.

1.3 Види та методи багатофакторної аутентифікації

Системи багатофакторної аутентифікації можуть бути реалізовані через різноманітні види та методи, які відрізняються за своїми принципами роботи і сферою застосування. Важливо зрозуміти різницю між термінами «метод» та «вид» у контексті БФА. Під терміном «метод» мається на увазі конкретна технологія або інструмент, що використовується для реалізації аутентифікації. Наприклад, це можуть бути одноразові паролі, що генеруються мобільними додатками, апаратні токени, або біометричні дані, такі як відбитки пальців або розпізнавання обличчя. Кожен з цих методів має свої особливості, переваги та недоліки, які визначають їх придатність для певних сценаріїв використання.

Є декілька основних видів БФА.

Двофакторна аутентифікація

Двофакторна аутентифікація є основою багатофакторної аутентифікації, оскільки вона вимагає надання двох незалежних факторів для підтвердження особи користувача. Це може бути поєднання чогось, що користувач знає, і чогось, що він має. Наприклад, найбільш поширеним варіантом є комбінація пароля та одноразового коду, що генерується мобільним додатком або надсилається через SMS. Іншим прикладом може бути пароль у поєднанні з фізичним токеном, таким

як смарт-карта або USB-ключ, який генерує одноразові паролі. Двофакторна аутентифікація значно підвищує рівень безпеки в порівнянні з використанням лише одного фактору, оскільки зловмиснику потрібно отримати доступ до обох факторів для успішної автентифікації.

Багатофакторна аутентифікація

Багатофакторна аутентифікація передбачає використання трьох і більше факторів для забезпечення доступу до системи. Це забезпечує ще більш високий рівень захисту в порівнянні з 2FA. Наприклад, користувач може спочатку ввести пароль, потім підтвердити свою особу за допомогою біометрії, наприклад, відбитка пальця або розпізнавання обличчя, і нарешті ввести одноразовий код, згенерований мобільним додатком. Використання трьох факторів забезпечує більш надійний захист, оскільки зловмиснику необхідно одночасно обійти всі три рівні автентифікації, що робить злом надзвичайно складним і малоймовірним.

Адаптивна аутентифікація

Адаптивна аутентифікація є сучасним підходом, який динамічно оцінює ризики на основі контексту і відповідно коригує вимоги до автентифікації. Контекстні фактори можуть включати місце розташування користувача, тип пристрою, з якого здійснюється вхід, час доби, звичайні шаблони поведінки користувача та інші змінні. Наприклад, якщо користувач намагається увійти в систему з нового або підозрілого місця, система може вимагати додаткового фактору автентифікації, такого як одноразовий код або підтвердження через мобільний додаток. У випадках низького ризику, коли контекст відповідає звичайному поведінкові користувача, система може обмежитися лише основним фактором автентифікації, таким як пароль. Таким чином, адаптивна аутентифікація забезпечує баланс між безпекою і зручністю для користувача, автоматично підвищуючи рівень захисту у випадках підвищеного ризику.

Також існує декілька методів БФА.

Апаратні токени

Апаратні токени є фізичними пристроями, які генерують одноразові паролі або криптографічні ключі. Вони забезпечують високий рівень безпеки завдяки фізичному носію аутентифікаційної інформації. Такі токени часто мають вбудовані дисплеї, які показують одноразовий пароль, або використовують USB-інтерфейси для безпосереднього підключення до комп'ютера. Вони є ефективним методом захисту, оскільки для їх використання потрібен фізичний доступ до самого пристрою. Однак їх недоліком може бути вартість та необхідність фізичного носіння пристрою користувачем.

Софтверні токени

Софтверні токени генерують одноразові паролі або криптографічні ключі за допомогою спеціальних програмних додатків. Ці токени, такі як Google Authenticator або Authy, використовують алгоритми на основі часу (TOTP) або подій (HOTP) для створення одноразових паролів. Вони є зручними, оскільки не потребують фізичного пристрою і можуть бути встановлені на смартфонах або інших мобільних пристроях. Однак, їх безпека залежить від безпеки самого пристрою, на якому встановлено додаток.

Біометричні методи

Біометричні методи використовують фізичні характеристики користувача для підтвердження особи. До них відносяться відбитки пальців, розпізнавання обличчя та сканування райдужної оболонки ока. Ці методи забезпечують високий рівень безпеки, оскільки біометричні дані є унікальними для кожної людини. Використання біометрії зручно для користувачів, оскільки не потребує запам'ятовування паролів або носіння додаткових пристроїв. Однак, впровадження таких методів може вимагати додаткового обладнання та забезпечення конфіденційності біометричних даних.

Одноразові паролі

Одноразові паролі є тимчасовими паролями, які можуть бути надіслані через SMS, електронну пошту або згенеровані токеном. Вони дійсні лише

протягом короткого проміжку часу, що зменшує ризик їх несанкціонованого використання. OTP є простими у використанні та інтеграції, і їх можна використовувати як додатковий рівень захисту до основного пароля. Однак, їх безпека може бути під загрозою у випадку перехоплення повідомлення або компрометації мобільного пристрою.

Push-сповіщення

Push-сповіщення надсилаються на мобільні пристрої і вимагають від користувача підтвердження або відхилення запиту на аутентифікацію. Цей метод зручний для користувачів, оскільки дозволяє підтверджувати вхід без введення додаткових паролів. Push-сповіщення забезпечують високий рівень безпеки, оскільки запит на аутентифікацію надходить безпосередньо на зареєстрований пристрій користувача. Однак, їх ефективність залежить від надійності інтернет-з'єднання та безпеки самого мобільного пристрою.

1.4 Висновок

У цьому розділі було детально розглянуто поняття та сутність багатофакторної аутентифікації, а також основні види та методи її реалізації. Було висвітлено ключові аспекти БФА, що забезпечують підвищення рівня безпеки інформаційних систем шляхом використання кількох незалежних факторів для підтвердження особи користувача.

Переваги БФА включають значне підвищення рівня безпеки, зниження ризиків компрометації даних та покращення користувацького досвіду. Однак, впровадження БФА може ускладнити процес аутентифікації, збільшити витрати на впровадження та створити проблеми з доступністю для деяких користувачів.

Таким чином, багатофакторна аутентифікація є ефективним засобом підвищення безпеки інформаційних систем, що забезпечує надійний захист від широкого спектру загроз та атак. Вибір конкретних видів та методів БФА повинен базуватися на специфічних потребах та вимогах безпеки кожної окремої системи.

РОЗДІЛ 2. СПЕЦІАЛЬНА ЧАСТИНА

2.1 Аналіз фреймворків для реалізації багатофакторної аутентифікації

ASP.NET Core Identity є одним з найпопулярніших фреймворків для аутентифікації та авторизації у додатках на основі ASP.NET Core. Він підтримує вбудовану багатофакторну аутентифікацію та забезпечує широкий спектр можливостей для реалізації безпечної аутентифікації.

Можливості

ASP.NET Core Identity підтримує кілька методів багатофакторної аутентифікації, таких як паролі, SMS або електронна пошта. Це дозволяє налаштувати політики безпеки та механізми відновлення доступу, що підвищує загальний рівень захисту користувацьких облікових записів. Фреймворк також інтегрується з різними сервісами та підтримує стандарти OAuth2 і OpenID Connect, що робить його гнучким і масштабованим рішенням для багатьох сценаріїв використання.

Переваги

ASP.NET Core Identity легко інтегрується з додатками на основі ASP.NET Core, що спрощує процес впровадження аутентифікації у веб-додатках. Велика кількість ресурсів для підтримки розробників, включаючи документацію, приклади та спільноти, робить цей фреймворк доступним для широкого кола розробників. Він також підходить для великих та складних додатків, забезпечуючи високу гнучкість та надійність.

Недоліки:

Однак, ASP.NET Core Identity може бути не підходящим для інших платформ або фреймворків, оскільки він тісно інтегрований з екосистемою .NET. Для початківців може бути складно налаштувати всі параметри безпеки, що може вимагати додаткового часу та ресурсів для вивчення та впровадження.

IdentityServer4 є відкритим фреймворком для реалізації аутентифікації та авторизації на основі протоколів OpenID Connect та OAuth2. Він забезпечує високий рівень безпеки та гнучкість у налаштуванні.

Можливості

Він підтримує багатофакторну аутентифікацію, дозволяючи інтеграцію з різними методами аутентифікації, такими як SMS, TOTP, U2F. Гнучка архітектура IdentityServer4 дозволяє налаштовувати та розширювати функціональність відповідно до потреб, забезпечуючи підтримку єдиної точки входу для кількох додатків.

Переваги

IdentityServer4 забезпечує високий рівень безпеки, відповідаючи сучасним стандартам безпеки. Він має широку функціональність, що підтримує різні сценарії аутентифікації та авторизації, а також активну спільноту розробників, що забезпечує регулярні оновлення та підтримку.

Недоліки

Проте, цей фреймворк має відносно високу складність і вимагає глибокого розуміння протоколів OpenID Connect та OAuth2. Для повної інтеграції може знадобитися додаткове налаштування зовнішніх сервісів, що може бути складним для менш досвідчених розробників.

Microsoft Azure AD B2C є хмарним рішенням для управління ідентифікацією та доступом, яке забезпечує підтримку багатофакторної аутентифікації для веб-додатків та мобільних додатків.

Можливості

Це рішення інтегрується з різними постачальниками ідентифікації, такими як Google, Facebook, Microsoft, а також локальними обліковими записами. Воно підтримує різні методи БФА, включаючи SMS, електронну пошту та додатки для аутентифікації, що робить його дуже гнучким у використанні.

Переваги

Azure AD B2C відзначається простотою налаштування, що дозволяє швидко впроваджувати його у існуючі додатки. Його масштабованість робить його придатним для додатків будь-якого розміру, а хмарна архітектура забезпечує високу доступність і безперебійну роботу.

Недоліки

Основними недоліками є залежність від інтернет-з'єднання, оскільки для роботи необхідне стабільне з'єднання з інтернетом, та вартість, яка може бути високою для малих підприємств або особистих проектів.

Okta є ще одним хмарним рішенням для управління ідентифікацією, яке забезпечує підтримку багатофакторної аутентифікації та інтеграцію з різними додатками.

Можливості

Воно підтримує різні методи БФА, такі як SMS, електронна пошта, додатки для аутентифікації та U2F. Okta інтегрується з широким спектром додатків, підтримуючи різні платформи та технології, і дозволяє гнучко налаштовувати політики безпеки відповідно до потреб організації.

Переваги

Okta відзначається простотою інтеграції, що дозволяє легко впроваджувати його у існуючі додатки та сервіси. Він також підтримує масштабування, роблячи його придатним для великих підприємств та складних систем. Високий рівень безпеки, що забезпечується Okta, відповідає сучасним стандартам безпеки та конфіденційності.

Недоліки

Проте, вартість цього рішення може бути високою для малих та середніх підприємств. Крім того, залежність від хмарного провайдера вимагає стабільного з'єднання з інтернетом для забезпечення безперебійної роботи системи.

2.2 Архітектура веб-додатку

Вибраною архітектурою додатку є Onion Architecture. Onion Architecture є архітектурним підходом, який спрямований на створення високої гнучкості та зниження залежностей між компонентами програмного забезпечення. Цей підхід особливо корисний у великих та складних проектах, де важливо забезпечити легку підтримку та розширення функціональності.

Є декілька основних принципів Onion Architecture

Відокремлення бізнес-логіки від інфраструктури

Основна ідея цього принципу полягає в тому, що бізнес-логіка не повинна залежати від зовнішніх сервісів, баз даних чи інших технологій. Це означає, що ядро додатка залишається незалежним і може функціонувати автономно, тоді як зовнішні компоненти залежать від бізнес-логіки. Такий підхід забезпечує захист від змін технологій, дозволяючи легко адаптуватися до нових інструментів та сервісів без необхідності змінювати основну бізнес-логіку. Відокремлення також сприяє більш зрозумілій та чіткій структурі коду, що полегшує його підтримку та розвиток.

Зменшення залежностей

Кожен шар архітектури в Onion Architecture має чітко визначені обов'язки і мінімальні залежності від інших шарів. Це означає, що кожен компонент системи функціонує незалежно, що сприяє модульності та дозволяє легко змінювати або замінювати окремі частини системи без впливу на інші компоненти. Зменшення залежностей також знижує складність системи, робить її більш стійкою до помилок та полегшує її тестування і підтримку.

Тестованість

Чітка структура Onion Architecture забезпечує високу тестованість додатків. Завдяки відокремленню бізнес-логіки від інфраструктури та мінімізації залежностей, кожен шар системи можна легко тестувати ізольовано від інших шарів. Це дозволяє створювати модульні тести для кожного компонента, що

підвищує якість коду та знижує ризик появи помилок. Тестування окремих шарів також полегшує процес виявлення та виправлення помилок, що сприяє більш ефективному та надійному розвитку програмного забезпечення.

В даному випадку будуть використовуватись чотири шари Onion Architecture:

- Api Layer
- Application Layer
- Domain Layer
- Infrastructure Layer

Api Layer відповідає за обробку вхідних запитів від клієнтів та передачу їх до відповідних сервісів бізнес-логіки. Він також формує відповіді, які надсилаються назад клієнтам. У випадку веб-додатків цей шар зазвичай реалізується у вигляді контролерів API.

Основні функції:

- Прийом HTTP-запитів.
- Валідація вхідних даних.
- Виклик методів Application Layer.
- Формування HTTP-відповідей.

Application Layer відповідає за реалізацію бізнес-логіки та координує роботу між різними частинами системи. Він не містить складної логіки, але забезпечує взаємодію між Domain Layer та іншими частинами системи.

Основні функції:

- Координація бізнес-логіки.
- Виклик методів Domain Layer.
- Обробка транзакцій.

Domain Layer є ядром архітектури та містить всі сутності, значення об'єктів та бізнес-логіку. Цей шар повністю ізольований від інших шарів та не залежить від жодної інфраструктури.

Основні функції:

- Моделювання бізнес-сутностей.
- Реалізація бізнес-правил та логіки.
- Визначення інтерфейсів для репозиторіїв та сервісів.

Infrastructure Layer відповідає за реалізацію всіх технологічних деталей, таких як робота з базою даних, зовнішніми сервісами та інші технічні аспекти. Цей шар взаємодіє з Domain Layer через інтерфейси, визначені у Domain Layer.

Основні функції:

- Реалізація репозиторіїв.
- Взаємодія з базами даних та зовнішніми сервісами.
- Інтеграція з іншими технологіями.

2.3 Розробка аутентифікації користувача

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 8.0.6

dotnet add package Microsoft.Extensions.Identity.Core --version 8.0.6
```

Для початку додамо бібліотеки в створений веб-додаток на ASP.NET Core.

Microsoft.AspNetCore.Authentication.JwtBearer - це бібліотека, яка забезпечує підтримку аутентифікації за допомогою JWT. JWT токени широко використовуються для передачі інформації між клієнтом та сервером у безпечний спосіб, оскільки вони можуть бути підписані та зашифровані. Це допомагає забезпечити захист даних та зниження ризиків компрометації облікових записів.

Microsoft.Extensions.Identity.Core - це бібліотека, яка є частиною ASP.NET Core Identity, фреймворку для управління користувачами, ролями та правами

доступу. Вона забезпечує основні функції управління ідентифікацією користувачів, включаючи створення, збереження та перевірку користувачів та їх ролей. Ця бібліотека дозволяє легко інтегрувати аутентифікацію та авторизацію в додаток, надаючи готові до використання класи та методи для роботи з обліковими записами, паролями, ролями та іншими аспектами управління користувачами.

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAuthentication(options =>
    {
        options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = "issuer",
            ValidAudience = "audience",
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("secret"))
        };
    });
```

Створимо екземпляр WebApplicationBuilder для налаштування всіх сервісі вебдодатку та додамо аутентифікацію в додаток з валідацією токена.

AddAuthentication додасть всі необхідні сервіси та залежності для аутентифікації в DI контейнер. Також налаштовує схему аутентифікації за замовчуванням, як JwtBearer.

AddJwtBearer додає всі необхідні залежності та схему аутентифікації для JWT токена в DI контейнер.

Також налаштовуємо валідацію токена за допомогою TokenValidationParameters. В даному випадку валідуються видавець, аудиторія, час життя та підпис токена.

```
builder.Services
    .AddIdentityCore<IdentityUser>()
    .AddRoles<IdentityRole>()
    .AddSignInManager<SignInManager<IdentityUser>>()
    .AddDefaultTokenProviders()
    .AddIdentityStore();
```

Сконфігуруємо серівси ASP.NET Core Identity для упавління користувачами та ролями у додатку.

Додамо основні сервіси для роботи з ідентифікацією користувачів за допомогою AddIdentityCore<IdentityUser>(). IdentityUser є вбудованим типом, який представляє користувача системи. Цей метод реєструє всі необхідні служби для роботи з користувачами, такі як управління користувачами, токенами та іншими аспектами ідентифікації.

Додамо підтримку управління ролями за допомогою AddRoles<IdentityRole>(). IdentityRole представляє роль користувача в системі. Це дозволяє створювати, змінювати та видаляти ролі, а також призначати їх користувачам.

Додамо управління аутентифікації та авторизації користувачів за допомогою AddSignInManager<SignInManager<IdentityUser>>(). SignInManager надає методи для входу в систему, виходу з системи, перевірки двофакторної аутентифікації та інші функції, пов'язані з управлінням сеансами користувачів.

Додамо набір вбудованих токен-провайдерів, які використовуються для генерації та перевірки токенів за допомогою AddDefaultTokenProviders(). Ці токени можуть використовуватися для різних цілей, таких як підтвердження електронної пошти, скидання пароля або двофакторна аутентифікація.

І додамо місце збереження всієї інформації про користувачів ASP.NET Core Identity за допомогою AddIdentityStore(). Це особистий метод, створений раніше для додавання збереження інформації у вже існуючу базу даних.

```
public interface IUserService
{
      Task<string> GenerateTokenAsync(IdentityUser user);
}
```

Створимо інтерфейс для полегшення роботи за даними користувача, в якому створимо методу для генерації токена.

```
internal sealed class UserService : IUserService
{
    private readonly UserManager<IdentityUser> _userManager;
    private readonly JwtSettings _jwtSettings;
    private readonly DateTime _jwtExpirationTime;

    public UserService(
        UserManager<IdentityUser> userManager,
        IOptions<JwtSettings> jwtOptions)
    {
        _userManager = userManager;
        _jwtSettings = jwtOptions.Value;

        _jwtExpirationTime =
DateTime.Now.AddHours(_jwtSettings.ExpireTimeInHours);
    }
}
```

Створимо клас, який спадкується від щойно створеного інтерфейсу, та додамо декілька залежностей. UserManager додається для створення секретного ключа для додатку аутентифікатора. JwtSettings додаються для додаткових налаштувань генерації JWT токена, таких як: час життя або секрета.

```
private async Task<IEnumerable<Claim>> GenerateClaimsAsync(IdentityUser
user)
{
    var roles = await _userManager.GetRolesAsync(user);
```

```
    if (roles.Count > 1)
    {
        throw new ArgumentException("User cannot have more roles than
1.");
    }

    var role = roles[0];

    var claims = new List<Claim>
    {
        new (JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new (JwtRegisteredClaimNames.Iat, DateTime.Now.Ticks.ToString()),
        new (ClaimTypes.NameIdentifier, user.Id),
        new (JwtRegisteredClaimNames.Name, user.UserName!),
        new (JwtRegisteredClaimNames.Exp,
_jwtExpirationTime.Ticks.ToString()),
        new (ClaimTypes.Role, role)
    };

    return claims;
}
```

Також створимо додатковий метод, для створення інформації про користувача, яка буде зберігатись в токені. За допомогою _userManager.GetRolesAsync(user), отримаємо ролі користувача.

```
public async Task<string> GenerateTokenAsync(IdentityUser user)
{
    var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.Secret));
    var credentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

    var claims = await GenerateClaimsAsync(user);

    var token = new JwtSecurityToken(
        issuer: _jwtSettings.Issuer,
        audience: _jwtSettings.Audience,
        claims: claims,
        expires: _jwtExpirationTime,
        signingCredentials: credentials);

    var jwt = new JwtSecurityTokenHandler().WriteToken(token);

    return jwt;
}
```

Створимо метод для генерації JWT токена. Створимо ключ за яким буде підписуватись токен, зробимо підпис, отримаємо інформацію про користувача, яку додамо в токен, та створимо токен за попередньою інформацією.

```
builder.Services.AddScoped<IUserService, UserService>();
```

Додамо сервіс в контейнер залежностей. За допомогою AddScoped<IUserService, UserService>(), можна буде отримувати один і той самий об'єкт при отриманні цієї залежності в одному запиті, та різні об'єкти в різних запитах.

```
public sealed class AccountsController : DefaultController
{
    private readonly IUserService _userService;
    private readonly UserManager<IdentityUser> _userManager;

    public AccountsController(
        IUserService userService,
        UserManager<IdentityUser> userManager)
    {
        _userService = userService;
        _userManager = userManager;
    }
}
```

Створимо контролер, та додамо декілька залежностей. IUserService допоможе створювати JWT токен, а UserManager створювання нового користувача.

```
[HttpPost("register")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> Register(
    [FromBody] RegisterModel registerModel)
{
    var user = new IdentityUser(registerModel.Username) { Email =
registerModel.Email };
    var registerResult = await _userManager.CreateAsync(user,
registerModel.Password);

    if (!registerResult.Succeeded)
```

```
    {
        return BadRequest(registerResult.Errors.ToDictionary(error =>
error.Code, error => error.Description));
    }

    await _userManager.AddToRoleAsync(user, UserConstants.Roles.DEFAULT);

    var token = await _userService.GenerateTokenAsync(user);

    return Ok(new { AccessToken = token });
}
```

Створюємо маршрут «/register», який обробляє POST запити, для реєстрації користувача. Спочатку об'єкт нового користувача, та спробуємо додато його в систему. При успішному створенні, додаємо йому роль звичайного користувача, генеруємо JWT токен, та повертаємо його у відповідь з статус кодом 200.

```
[HttpPost("login")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> Login(
    [FromBody] LoginModel loginModel)
{
    var user = await _userManager.FindByNameAsync(loginModel.Username);

    if (user is null)
    {
        return Error("User cannot be found.",
StatusCodes.Status404NotFound);
    }

    var correctPassword = await _userManager.CheckPasswordAsync(user,
loginModel.Password);

    if (!correctPassword)
    {
        return Error("Incorrect password.");
    }

    var token = await _userService.GenerateTokenAsync(user);

    return Ok(new { AccessToken = token });
}
```

Створюємо маршрут «/login», який обробляє POST запит, для отримання токену, який використовується для отримання даних, яке потребують аутентифікації. Спочатку шукаємо користувача за його ім'ям. Якщо користувач був знайдений, то перевіряється пароль на правильність. Та при успішній перевірці, генерується токен та повертається у відповідь.

```
[HttpPost]
[Authorize]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> Create(
    [FromBody] CreateProductCommand createCommand,
    CancellationToken cancellationToken = default)
{
    try
    {
        var product = await _mediator.Send(createCommand,
cancellationToken);

        return CreatedAtAction(nameof(GetById), new { Id = product.Id },
product);
    }
    catch (Exception exception)
    {
        _logger.LogError(exception.Message);

        return BadRequest();
    }
}
```

Додамо у вже існуючий контролер ProductsController, новий маршрут, який обробляє POST запити, для створення нового продукту, з доступом тільки для авторизованих користувачів за допомогою атрибуту «[Authorize]». Отримаємо продукт та повернемо його у відповідь з статус кодом 201.

```
[HttpPut("{id}")]
[Authorize]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> Update(
    [FromRoute] string id,
    [FromBody] UpdateProductModel productModel,
    CancellationToken cancellationToken = default)
{
```

```
    try
    {
        var command = _mapper.Map<UpdateProductCommand>(productModel);
        command.Id = id;

        var product = await _mediator.Send(command, cancellationToken);

        return Ok(product);
    }
    catch (ProductNotFoundException notFound)
    {
        return Error(notFound.Message, StatusCodes.Status404NotFound);
    }
}
```

Створимо новий маршрут «/{id}», який обробляє PUT запит, для оновлення вже існуючого продукту. Створюємо команду на оновлення продукту. Відправляємо її та отримаємо новий продукт. Повертаємо новий продукт з статус кодом 200.

```
[HttpDelete("{id}")]
[Authorize]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult> Delete(
    [FromRoute] string id,
    CancellationToken cancellationToken = default)
{
    try
    {
        var command = new DeleteProductCommand(id);

        await _mediator.Send(command, cancellationToken);

        return NoContent();
    }
    catch (ProductNotFoundException notFound)
    {
        return Error(notFound.Message, StatusCodes.Status404NotFound);
    }
}
```

Створимо маршрут «/{id}», який обробляє DELETE запит, для видалення існуючого продукту. Спочатку створимо команду на видалення продукту, відправимо її та повернемо користувачу пусту відповідь з статус кодом 204.

2.4 Інтеграція багатофакторної аутентифікації

```
public interface IUserService
{
    Task<string> GenerateTokenAsync(IdentityUser user);
    Task<string> GenerateAuthenticatorUriAsync(IdentityUser user);
}
```

Додамо, метод створення посилання для додатка аутентифікатора за користувачем в інтерфейс.

```
public async Task<string> GenerateAuthenticatorUriAsync(IdentityUser
user)
    {
        var authenticatorKey = await
_userManager.GetAuthenticatorKeyAsync(user);

        if (string.IsNullOrEmpty(authenticatorKey))
        {
            await _userManager.ResetAuthenticatorKeyAsync(user);
            authenticatorKey = await
_userManager.GetAuthenticatorKeyAsync(user);
        }

        var authenticatorUri =
$"otpauth://totp/{Uri.EscapeDataString(_jwtSettings.Issuer)}:" +
                                $"{Uri.EscapeDataString(user.Email!)}?" +
                                $"secret={authenticatorKey}&" +

$"issuer={Uri.EscapeDataString(_jwtSettings.Issuer)}";

        return authenticatorUri;
    }
```

Та реалізуємо цей метод в сервісі. Спочатку отримаємо ключ для додатку аутентифікатора, та якщо його немає створимо новий. Згенеруємо посилання для додатка аутентифікатора та повернемо його у відповідь.

```
[HttpGet("2fa/uri")]
[Authorize]
[ProducesResponseType(StatusCodes.Status200OK)]
public async Task<IActionResult> GetAuthenticatorUri()
{
    var user = await GetUserAsync();
    var authenticatorUri = await
_userService.GenerateAuthenticatorUriAsync(user);
```

```
    return Ok(authenticatorUri);
}
```

Створимо маршрут «/2fa/uri», який обробляє GET запит, для отримання посилання для додатку аутентифікатора. Спочатку отримаємо користувача, далі посилання аутентифікатора, та повернемо посилання.

```
public async Task<string> GenerateAuthenticatorUriAsync(IdentityUser
user)
{
    var authenticatorKey = await
_userManager.GetAuthenticatorKeyAsync(user);

    if (string.IsNullOrEmpty(authenticatorKey))
    {
        await _userManager.ResetAuthenticatorKeyAsync(user);
        authenticatorKey = await
_userManager.GetAuthenticatorKeyAsync(user);
    }

    var authenticatorUri =
$"otpauth://totp/{Uri.EscapeDataString(_jwtSettings.Issuer)}:" +
                           $"{Uri.EscapeDataString(user.Email!)}?" +
                           $"secret={authenticatorKey}&" +

$"issuer={Uri.EscapeDataString(_jwtSettings.Issuer)}";

    return authenticatorUri;
}
```

Створимо метод генерації посилання, для додатків аутентифікаторів. Спочатку отримаємо ключ аутентифікатора і якщо він пустий, то створимо новий. Згенеруємо посилання за допомогою пошти користувача, та ключа аутентифікатора.

```
[HttpPost("2fa/toggle")]
[Authorize]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> ManageTwoFactor(
    [FromBody] TwoFactorModel twoFactorModel)
{
    var user = await GetUserAsync();
```

```
    var isCorrectCode = await IsCorrectTwoFactorCodeAsync(user,
twoFactorModel.Code);

    if (!isCorrectCode)
    {
        return Error("Incorrect TwoFactorCode.");
    }

    var newTwoFactorState = !user.TwoFactorEnabled;

    await _userManager.SetTwoFactorEnabledAsync(user, newTwoFactorState);

    return NoContent();
}
```

Створимо маршрут «/2fa/toggle», який обробляє POST запит, для включення або виключення двофакторної аутентифікації для користувача. Спочатку отримаємо користувача. Перевіримо чи правильний код двофакторної аутентифікації. Якщо код правильний, перемкнемо теперішній стан для користувача, та повернемо пусту відповідь з статус кодом 204.

```
[HttpPost("login")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> Login( [FromBody] LoginModel loginModel)
    {
        var user = await
_userManager.FindByNameAsync(loginModel.Username);

        if (user is null)
        {
            return Error("User cannot be found.",
StatusCodes.Status404NotFound);
        }

        var correctPassword = await _userManager.CheckPasswordAsync(user,
loginModel.Password);

        if (!correctPassword)
        {
            return Error("Incorrect password.");
        }

        if (user.TwoFactorEnabled)
```

```
        {
            If (loginModel.TwoFactorCode is null) return
Error("TwoFactorCode must be not null.");

            var result = await IsCorrectTwoFactorCodeAsync(user,
loginModel.TwoFactorCode);

            if (!result)
            {
                return Error("Incorrect TwoFactorCode.");
            }
        }

        var token = await _userService.GenerateTokenAsync(user);

        return Ok(new { AccessToken = token });
    }
```

Модифікуємо маршрут «/login», який обробляє POST запит для логіну користувача, щоб інтегрувати багатофакторну аутентифікацію під час логіну.

Додамо перевірку чи включена БФА, після чого перевіримо чи передав користувач код аутентифікації та правильність коду. Якщо все правильно, згенеруємо та повернемо токен.

2.5 Тестування аутентифікації

Для спрощення процесу тестування та забезпечення більшої прозорості API, буде використовуватись Swagger. Swagger є інструментом, який дозволяє автоматично генерувати інтерактивну документацію для API на основі вихідного коду. Це забезпечує зручний спосіб візуалізації та тестування всіх доступних кінцевих точок API без необхідності написання додаткового коду або документації вручну.

Swagger надає веб-інтерфейс, де користувачі можуть переглядати всі доступні кінцеві точки, ознайомлюватися з їхніми параметрами, типами запитів та відповідей. Це значно полегшує процес розробки, оскільки розробники можуть безпосередньо у браузері перевіряти правильність роботи API, відправляти запити, переглядати відповіді сервера та знаходити потенційні помилки.

Крім того, використання Swagger сприяє підвищенню якості коду, оскільки він автоматично генерує документацію, яка завжди відповідає актуальному стану API. Це зменшує ризик виникнення невідповідностей між фактичною реалізацією API та його описом.

Тож запустимо проект, та почнемо тестування.

Після запуску відкривається вікно браузеру, з маршрутом «https://localhost:xxxx/swagger»

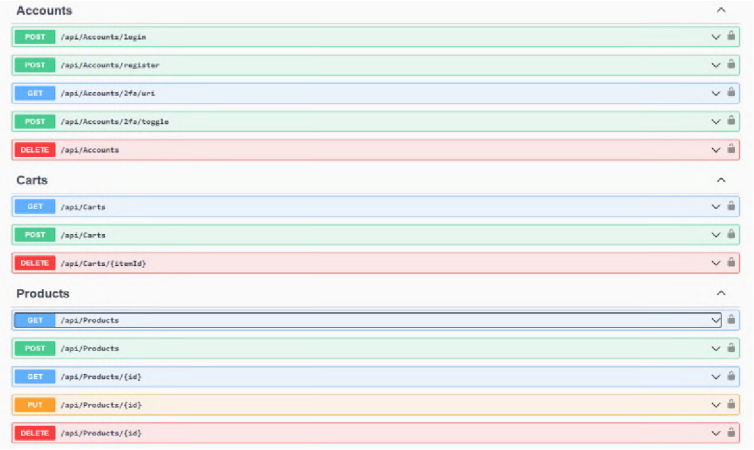

рис. 1.1 –Інтерфейс swagger та всі створенні маршрути

Спробуємо звернутися до незахищеного маршруту «/api/Products» з методом GET на отримання всіх продуктів

рис. 1.2 – Успішна відповідь від сервера з усіма продуктами та статус кодом 200

Спробуємо звернутись до захищеного маршруту «/api/Products» з методом POST на створення нового продукту

рис. 1.3 – Помилка у відповіді від сервера з статус кодом 401 (неавторизований)

Спробуємо звернутись до незахищеного маршруту «api/Accounts/register» з методом POST, щоб зареєструвати нового користувача

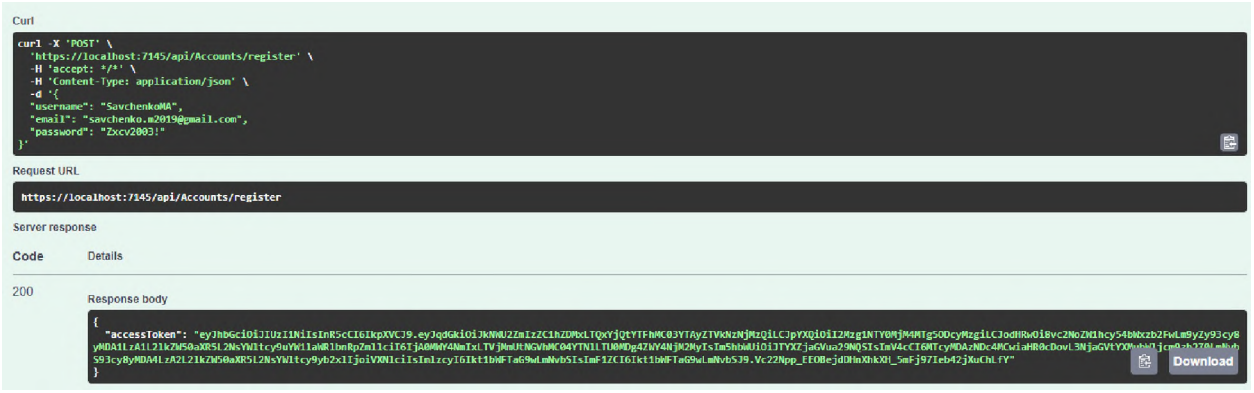

рис. 1.4 – Успішна відповідь, та згенерований JWT токен у відповіді, та статус код 200

Також, для аналізу вмісту JSON Web Token (JWT) ми будемо використовувати сторонній сервіс "jwt.io". Цей сервіс дозволяє розробникам легко декодувати, перевіряти та налагоджувати JWT токени.

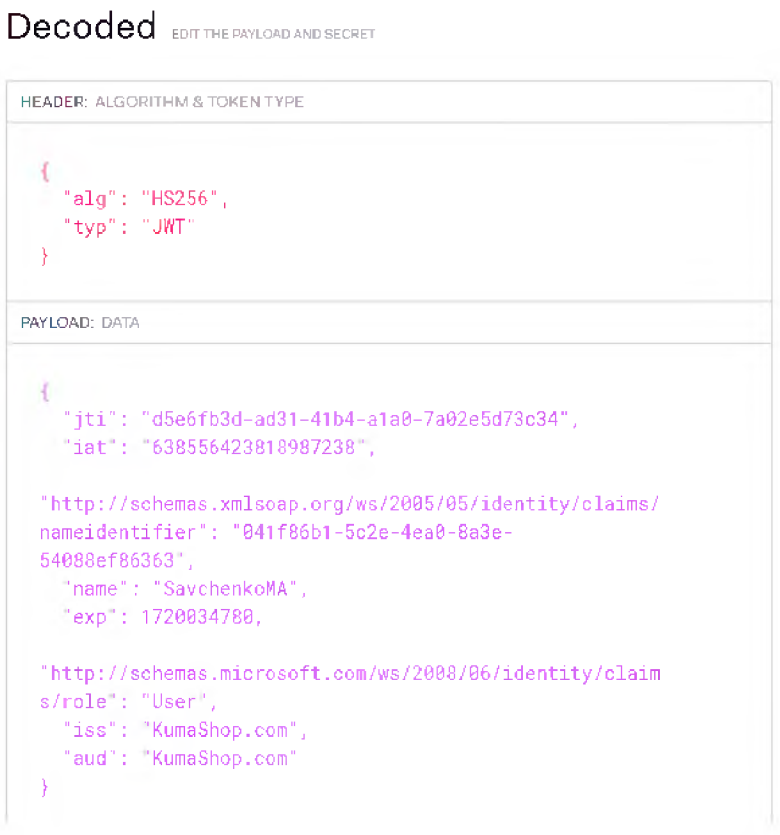

рис. 1.5 – Декодований JWT токен

Додамо токен до swagger, щоб відправляти всі запити з ним

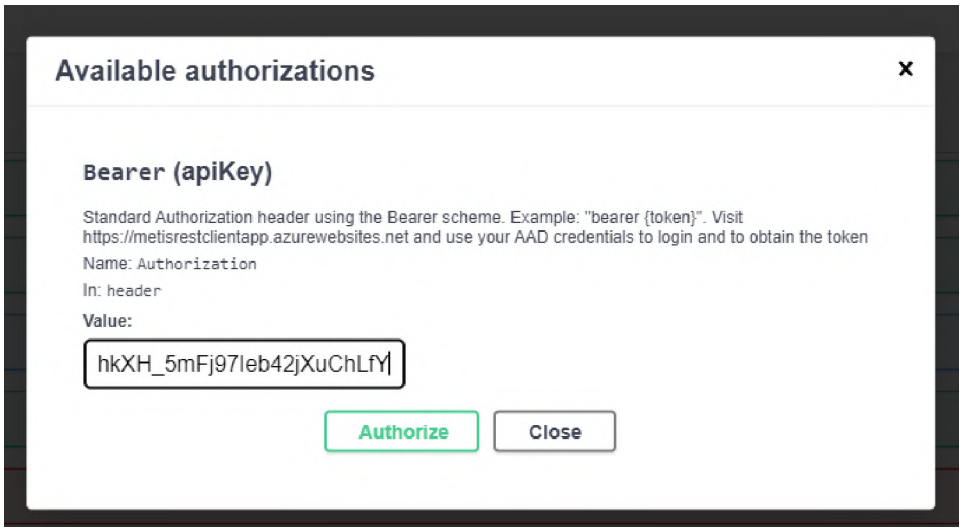

рис. 1.6 – Додання токена в swagger

Тепер спробуємо звернутись до захищеного маршруту на створення нового продукту

рис. 1.7 – Успішна відповідь, з інформацією про новий продукт та статус код 201

Тепер отримаємо посилання для додання генерації паролів в додаток аутентифікатор

рис. 1.8 – Успішна відповідь з посиланням та статус кодом 200

Для можливості тестування, скористаємося стороннім сервісом "totp.app" для генерації одноразових паролів. Цей сервіс дозволяє зручно і швидко створювати тимчасові коди для двофакторної аутентифікації, що значно полегшить перевірку функціональності нашої системи аутентифікації.

рис. 1.9 – Інтерфейс «totp.app»

Додамо генерацію одноразових кодів у сервіс «totp.app» за допомогою посилання, отриманого раніше

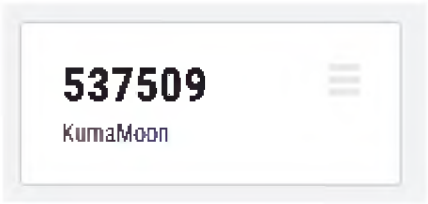

рис. 1.10 – Генерація одноразових кодів

Спробуємо звернутися до маршрута «/api/Accounts/2fa/toggle» з методом POST, для включення БФА

1.11 – Успішна відповідь з статус кодом 204

Спробуємо отримати JWT токен для створеного користувача, без одноразового коду

1.12 – Повернення помилки, так як не вказаний\вказаний неправильно одноразовий код з статус кодом 400

Спробуємо отримати JWT токен, з одноразовим кодом

рис. 1.13 – Успішна відповідь з JWT токеном та статус кодом 200

Спробуємо звернутись до захищеного маршруту на створення продукту «/api/Products» з методом POST, з отриманим токеном

рис. 1.14 – Успішна відповідь з інформацією про новий продукт та статус кодом 201

2.6 Висновок

У цьому розділі було розглянуто реалізацію аутентифікації користувачів у веб-додатку на основі ASP.NET Core. Використовуючи можливості ASP.NET Core Identity, було додано підтримку багатофакторної аутентифікації, яка забезпечує додатковий рівень безпеки для користувачів. Було розглянуто кілька популярних фреймворків для аутентифікації, включаючи ASP.NET Core Identity, IdentityServer4, Microsoft Azure AD B2C та Okta, з їхніми перевагами та недоліками.

Основна увага була приділена реалізації Onion Architecture, що дозволяє розділити додаток на декілька основних шарів. Це сприяє високій гнучкості та зниженню залежностей між компонентами програмного забезпечення.

Також було детально описано процес інтеграції аутентифікації за допомогою JWT токенів. Зокрема, було розглянуто налаштування аутентифікації в ASP.NET Core, генерацію JWT токенів, створення користувачів та їхніх ролей. Було показано, як використовувати JWT токени для захисту маршрутів API та забезпечення доступу лише для авторизованих користувачів.

Окрім того, було розглянуто додавання багатофакторної аутентифікації за допомогою додатка аутентифікатора, що підвищує рівень безпеки, вимагаючи від користувачів додатковий фактор підтвердження під час входу в систему.

Таким чином, впровадження ASP.NET Core Identity у поєднанні з багатофакторною аутентифікацією забезпечує високий рівень безпеки та зручність у використанні, роблячи додаток більш надійним і захищеним від несанкціонованого доступу.

РОЗДІЛ 3. ЕКОНОМІЧНА ЧАСТИНА

3.1 Розрахунок витрат на впровадження багатофакторної аутентифікації

Витрати на впровадження та підтримку програмного забезпечення для інтеграції багатофакторної аутентифікації складаються з капітальних (одноразових) витрат та витрат на підтримку функціональності і технічного обслуговування нововведень, які розраховуються на період одного року.

3.2. Розрахунок капітальних витрат

Щоб розрахувати вартість створення оновлення автентифікації (Кпа), необхідно розрахувати вартість машинного часу (Смч) і помножити на кількість витрачених для оновлення політики автентифікації годин. Програмне забезпечення для використання сервісів компанії Amazon.

- P = 680 Вт/*год = 0,68 кВт/*год (номінальна потужність серверу на якому виконувались роботи).
- Ce = 4,27 грн/кВт (вартість електроенергії).
- Первісна вартість серверу становить 32,000 грн.

Сервер використовується вже третій рік. Цей технічний засіб амортизується за прямолінійним методом, а термін його корисного використання складає 6 років. Ліквідаційна вартість обладнання становить 5,000 грн, тому сума щорічних амортизаційних відрахувань дорівнює 4,000 грн, а норма амортизації (На) становить 18%. На початок цього року залишкова вартість склала:

$$Ф_{зал} = 32{,}000 - 4{,}000 = 28{,}000 \text{ грн}$$

Далі потрібно розрахувати вартість одної години машинного часу ПК ($С_{мч}$):

$$Смч = P \cdot t_{нал} \cdot Ce + \frac{Ф_{зал} \cdot На}{F_p} + \frac{К_{лпз}}{F_p}$$

де:

- P – встановлена потужність серверу, кВт;
- Ce – тариф на електричну енергію, грн/кВт·година;
- $Ф_{зал}$ – залишкова вартість серверу на поточний рік, грн;
- $Н_а$ – річна норма амортизації на сервер, частки одиниці;
- $К_{ппз}$ – вартість ліцензійного програмного забезпечення, грн;
- Fp – річний фонд робочого часу (за 40-годинного робочого тижня Fp=1920).

Розрахунок вартості машинного часу:

$$\text{Смч} = 0.68 \times 4.27 + \frac{28{,}000 \times 0.16}{1920} = 2.33 \text{ грн/год}$$

Розрахунок вартості створення оновлення автентифікації ($К_{па}$):

$$\text{Кпа} = \text{Смч} \times 100 = 2.33 \times 100 = 233 \text{ грн}$$

Витрати на розробку та впровадження включають витрати на розробку програмного забезпечення, інсталяцію та конфігурацію системи. Нижче наведені необхідні витрати:

- Витрати на розробку програмного забезпечення:
- Зарплата розробників (погодинна ставка 600 грн, 120 годин):
- Розробка = 600 грн/год × 120 год = 72,000 грн
- Витрати на інсталяцію та конфігурацію системи:
- Зарплата системних адміністраторів (погодинна ставка 400 грн, 60 годин):
- Інсталяція та конфігурація = 400 грн/год × 60 год = 24,000 грн

Таким чином, витрати на розробку та впровадження програмного забезпечення складають:

- Витрати на розробку програмного забезпечення: 37,000 грн
- Витрати на інсталяцію та конфігурацію системи: 26,000 грн
- Загальні витрати на розробку та впровадження: 63,000 грн
- Загальна сума капітальних витрат становить 63,233 грн.

3.3 Розрахунок річних поточних (експлуатаційних) витрат

Поточні витрати включають витрати на підтримку дієздатності та технічне супроводження програмного забезпечення для захисту веб-додатку. У нашому випадку це витрати на електроенергію, технічне обслуговування та підтримку системи. Нижче наведені необхідні витрати:

1. Вартість електроенергії:

- Сервер LG Edge X40 споживає 680 Вт (0.68 кВт).
- Вартість електроенергії: 4.27 грн/кВт * год.
- Річний фонд робочого часу: 1920 год.

Розрахунок вартості електроенергії:

$$В_{електроенергії} = 0.68 \text{ кВт} \times 4{,}27 \text{ грн/кВт} \cdot \text{год} \times 1920 \text{ год} = 5{,}575 \text{ грн/рік}$$

2. Технічне обслуговування та підтримка:

- Технічне обслуговування та підтримка програмного забезпечення (оплата праці IT-фахівців, погодинна ставка 600 грн, 36 год на рік):

$$В_{техпідтримка} = 600 \text{ грн/год} \times 36 \text{ год} = 21{,}600 \text{ грн/рік}$$

Таким чином, річні поточні витрати складають:

Стаття витрат	Сума, грн/рік
Вартість електроенергії	5,575
Технічне обслуговування та підтримка	21,600
Загальна сума поточних витрат	27,175

Таблиця 4.1 – Річні поточні (експлуатаційні) витрати

Таким чином, загальні річні поточні витрати на підтримку програмного забезпечення для інтеграції багатофакторної аутентифікації становлять 21,600 грн на рік.

3.4 Оцінка величини можливого збитку від атаки

Для оцінки економічної доцільності можна не враховувати всі можливі загрози та втрати від них. Достатньо розглянути лише деякі загрози, і якщо втрати від них перевищують витрати на впровадження політик безпеки, це вже свідчить про доцільність інвестицій. Для приблизної оцінки збитків за рік розглянемо ті ситуації, які мають найвищу ймовірність виникнення:

1. Помилки співробітників:

- Вірогідність реалізації загрози через джерело загрози АВн1 та вразливість С2 – 0,221.
- Вірогідність реалізації загрози через джерело загрози АЗв3 та вразливість С2 – 0,135.
- Середня ймовірність реалізації R1: $R1 = \frac{0,221+0,135}{2} = 0,178$

2. Атаки методом грубої сили:

- Вірогідність реалізації загрози через джерело загрози АВн1 та вразливість С6 – 0,235.
- Вірогідність реалізації загрози через джерело загрози Т6 та вразливість С6 – 0,128.
- Середня ймовірність реалізації R2: $R2 = \frac{0,235+0,128}{2} = 0,180$

3. Використання шкідливих програм:

- Вірогідність реалізації загрози через джерело загрози АВн1 та вразливість О5 – 0,333.
- Вірогідність реалізації загрози через джерело загрози Т6 та вразливість О5 – 0,333.

- Вірогідність реалізації загрози через джерело загрози АЗв3 та вразливість О5 – 0,333.
- Середня ймовірність реалізації R3: $R3 = \frac{0,333+0,333+0,333}{3} = 0,333$

Припустимо 3 ситуації, уявімо, що кожна відбудеться раз на рік з врахуванням ймовірнісної характеристики:

1. Перехоплення інформації про стандартне замовлення – витрати на повторне уведення інформації ($П_{ви}$) та втрати від зниження обсягу продажів (V):

$В1 = Пви + V = 700 + 40\,000 = 40\,700$ грн

Витрати на повторне уведення інформації ($П_{ви}$): 700 грн

$$Пви = \frac{\sum Зс}{F} \cdot tви$$

де:

- $\sum Зс$ – сума заробітних плат всіх співробітників, залучених до процесу;
- Тви – Час витрачений на введення інформації;
- F – Загальний фонд робочого часу;

$$Пви = \frac{70000}{100} \cdot 2 = 1400 \text{ грн}$$

Втрати від зниження обсягу продажів (V): 40 000 грн

2. Атака на комп'ютер співробітника – викрадення конфіденційної інформації про майбутні 3 замовлення:

$В2 = Пви + V = 800 + 40\,000 \times 3 = 120\,800$ грн

Витрати на повторне уведення інформації (Пви): 800 грн

$$Пви = \frac{4000}{100} \cdot 2 = 800 \text{ грн}$$

Втрати від зниження обсягу продажів (V) за майбутні 3 замовлення: 40 000 × 3 = 120 000 грн

3. Хакерська атака на комп'ютерну мережу – витрати на відновлення працездатності вузла або сегмента мережі ($П_{пв}$), заміна устаткування ($П_{зч}$), втрати від простою ($П_п$):

$$ВЗ = Пп + Пв + V = 1500 + 1000 + 140{,}000 = 142{,}500 \text{ грн}$$

Витрати на відновлення працездатності вузла або сегмента мережі ($П_{пв}$): 240 грн

$$Ппв = \frac{\sum Зо}{F} \cdot tв$$

де:

- $\sum З_о$ – сума заробітної плати адміністраторів;
- $t_в$ – Час на відновлення після атаки;
- F – Загальний фонд робочого часу;

$$Ппв = \frac{12000}{50} \cdot 1 = 240 \text{ грн}$$

Заміна устаткування (Пзч): 1,000 грн

Втрати від простою (Пп): 2,500 грн

$$Пп = \frac{\sum Зс}{F} \cdot tп$$

де:

- $\sum З_с$ – сума заробітної плати співробітників атакованого вузла;
- $t_п$ – Час простою вузла сегмента корпоративної мережі внаслідок атаки.
- F – Загальний фонд робочого часу;

$$Пп = \frac{50000}{100} \cdot 6 = 3000 \text{ грн}$$

Сумарний можливий збиток за рік становитиме:

$$B = B1 + B2 + B3 = 40{,}700 + 120{,}800 + 245{,}000 = 284{,}600 \text{ грн}$$

Таким чином, можливі збитки від атак значно перевищують витрати на впровадження заходів безпеки, що підтверджує економічну доцільність інтеграції програмного забезпечення для багатофакторної аутентифікації.

3.5 Загальний ефект від впровадження системи інформаційної безпеки

Загальний ефект від впровадження системи інформаційної безпеки визначається з урахуванням ризиків порушення інформаційної безпеки і становить:

$$E = \sum Bi \cdot Ri - C$$

де:

- B – загальний збиток від атаки на вузол або сегмент корпоративної мережі, тис. грн;
- R – очікувана імовірність атаки на вузол або сегмент корпоративної мережі, частки одиниці;
- C – щорічні витрати на експлуатацію системи інформаційної безпеки, тис. грн.

$$E = (40{,}700 \times 0.178 + 142{,}500 \times 0.180 + 284{,}600 \times 0.333) - 21{,}600 = 106{,}066.4 \text{ грн}$$

3.6 Визначення та аналіз показників економічної ефективності системи інформаційної безпеки

$$ROSI = \frac{E}{K}$$

де:

- E – загальний ефект від впровадження системи інформаційної безпеки, тис. грн;
- K – капітальні інвестиції за варіантами, що забезпечили цей ефект, тис. грн.

$$ROSI = \frac{106{,}066.4}{55310} = 1.92$$

3.7 Висновки

Виходячи з отриманих розрахунків та загального огляду, можна зазначити, що впровадження багатофакторної аутентифікації у вебдодаток було економічно доцільним. Загальний економічний ефект від впровадження системи інформаційної безпеки, розрахований у цьому розділі, становить позитивну величину (106,066.4 грн), що підтверджує, що корисний економічний ефект перевищує витрати. Також важливим показником є коефіцієнт повернення інвестицій. У нашому випадку він перевищує одиницю, що демонструє прибутковість вкладень.

ВИСНОВКИ

Інтеграція багатофакторної аутентифікації у веб-додаток є критично важливим кроком для забезпечення високого рівня безпеки користувацьких даних і зниження ризиків від несанкціонованого доступу. БФА додає додатковий рівень захисту. Це значно ускладнює можливість зловмисників отримати доступ до облікових записів, навіть якщо вони знають пароль.

Проведений аналіз економічної доцільності впровадження БФА показав, що хоча капітальні витрати на впровадження та підтримку можуть бути значними, потенційні збитки від кібератак і витоку даних значно перевищують ці витрати. Витрати на розробку, закупівлю технічних засобів та інтеграцію системи є виправданими з точки зору забезпечення довготривалої безпеки та збереження репутації компанії.

Оцінка економічної ефективності показала, що інвестиції в БФА є рентабельними і забезпечують значний позитивний ефект, знижуючи ймовірність фінансових втрат від можливих кібератак. Коефіцієнт повернення інвестицій вказує на те, що впровадження БФА є не лише необхідним заходом для підвищення безпеки, але й економічно вигідним рішенням.

Загалом, інтеграція багатофакторної аутентифікації у веб-додаток сприяє створенню більш захищеного середовища для користувачів, підвищуючи довіру до компанії і мінімізуючи ризики фінансових втрат від несанкціонованого доступу до системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Adam Freeman. «Pro ASP.NET Core Identity: Under the Hood with Authentication and Authorization in ASP.NET Core 5 and 6 Applications». ASP.NET Core Identity: огляд під капотом аутентифікації та авторизації в додатках ASP.NET Core 5 та 6.
2. Шортрідж, Келлі. «Security Chaos Engineering. Developing Resilience and Safety at Speed and Scale». Книга.
3. Когут, Юрій «Кібербезпека та ризики цифрової трансформації компаній». Книга.
4. Даєр-Візефорд, Нік. «Кібервійна і революція». Книга.
5. Диньї, Пей. «Authentication Codes and Combinatorial Designs». Книга.
6. Ніемі, Валттері. «Cellular Authentication for Mobile and Internet Services». Книга.і
7. Андресс, Джейсон. «Захист даних. Від авторизації до аудиту». Книга.
8. What is Multi-Factor Authentication? Що таке багатофакторна аутентифікація? Режим доступу: https://aws.amazon.com/what-is/mfa/
9. «Багатофакторна аутентифікація». Режим доступу: https://support.microsoft.com/en-us/topic/what-is-multifactor-authentication-e5e39437-121c-be60-d123-eda06bddf661
10. Аутентифікація в веб-додатках ASP.NET Core. Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-8.0
11. Стаття «Best Practice in Multi-factor Authentication». Режим доступу: https://www.researchgate.net/publication/351852137_Best_Practice_in_Multi-factor_Authentication
12. Стівенс, Тім. «Глобальна кібербезпека: нові напрями в теорії та методах». Політика та управління.

13. Bojanc, R. та Jerman-Blažič, B. Економічний модельний підхід до управління ризиками інформаційної безпеки. International Journal of Information Management
14. Когут, Юрій «Кібервійни, кібертероризм, кіберзлочинність. Концепції, стратегії, технології». Книга
15. Когут, Юрій «Кібертероризм». Книга
16. Когут, Юрій «Кібервійна та безпека об'єктів критичної інфраструктури». Книга
17. Сенґер, Девід «Досконала зброя. Війна, саботаж і страх у кіберепоху». Книга
18. Кнодель, Мелорі «Свобода в мережі. Як насправді працює інтернет». Книга
19. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах».
20. Закон України «Про інформацію».

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи

№	Формат	Найменування	Кількість листів	Примітка
1	А4	Реферат	2	
2	А4	Список умовних скорочень	1	
3	А4	Зміст	2	
4	А4	Вступ	1	
5	А4	1 Розділ	8	
6	А4	2 Розділ	26	
7	А4	3 Розділ	8	
8	А4	Висновки	1	
9	А4	Перелік посилань	2	
10	А4	Додаток А	1	
11	А4	Додаток Б	1	
12	А4	Додаток В	1	
13	А4	Додаток Г	1	

ДОДАТОК Б. Перелік документів на оптичному носії

Пояснювальна записка.docx

Пояснювальна записка.pdf

Презентація.pptx

ДОДАТОК В. Відгук керівника економічного розділу

Економічний розділ виконаний відповідно до вимог, які ставляться до кваліфікаційних робіт, та заслуговує на оцінку 90б. (« відмінно »).

Керівник розділу _____ Дар`я ПІЛОВА

ДОДАТОК Г. Відгук керівника кваліфікаційної роботи

ВІДГУК

на кваліфікаційну роботу студента групи 125-20-2

Савченка Максим Андрійович

На тему: Інтеграція багатофакторної аутентифікації у веб-додаток на C#.

Пояснювальна записка складається з титульного аркуша, завдання, реферату, списку умовних скорочень, змісту, вступу, трьох розділів, висновків, переліку посилань та додатків, розташованих на 57 сторінках та містить 14 рисунків, 1 таблицю, 20 джерел та 4 додатка.

Студент показав достатній рівень володіння теоретичними положеннями з обраної теми, показав здатність формувати власну точку зору (теоретичну позицію).

Робота оформлена та написана грамотною мовою, відповідає вимогам положення про систему запобігання та виявлення плагіату у Національному технічному університеті «Дніпровська політехніка». Містить необхідний ілюстрований матеріал. Автор добре знає проблему, уміє формулювати наукові та практичні завдання і знаходить адекватні засоби для їх вирішення.

В цілому робота задовольняє усім вимогам і може бути допущена до захисту, а його автор заслуговує на оцінку «_____».

Керівник кваліфікаційної роботи проф. Магро. В.І.

Керівник спец. розділу проф. Магро. В.І.