Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
в.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«17» травня 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Модель аутентифікації та авторизації в веб-додатках (за допомогою ASP.NET Core Identity)»

Виконавець: студентка ІІ курсу, групи КБм-21

_____ **Катерина ВЕНГРИНОВСЬКА**
(підпис) (Ім'я, ПРІЗВИЩЕ)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Сергій БУЧИК	
Нормоконтроль	Інна МИХАЛЬЧУК	

Київ 2024

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
в.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«17» листопада 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувачки КБм-21 (група) Венгриновської Катерини Володимирівни (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Модель аутентифікації та авторизації в веб-додатках (за допомогою ASP.NET Core Identity)

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 15.11.2023 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес автентифікації та авторизації в вебдодатках.
Предмет досліджень	Модель автентифікації та авторизації в вебдодатках, реалізована за допомогою ASP.NET Core Identity.
Мета	Розробка моделі автентифікації та авторизації в вебдодатках за допомогою ASP.NET Core Identity.

Вихідні дані для проведення роботи Методи, протоколи та технології автентифікації та авторизації.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна Удосконалення моделі автентифікації та авторизації для найпростіших вебдодатків за допомогою технології ASP.NET Core Identity.

Практична цінність Реалізація удосконаленої моделі автентифікації та авторизації для найпростіших вебдодатків.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	17.11.2023 – 21.01.2024
Аналіз літературних джерел	22.01.2024 – 04.02.2024
Розгляд методів автентифікації та авторизації	05.02.2024 – 18.02.2024
Дослідження протоколів автентифікації та авторизація	19.02.2024 – 03.03.2024
Ознайомлення з сучасними технологіями автентифікації та авторизації	04.03.2024 – 10.03.2024
Розробка удосконаленої моделі автентифікації та авторизації	11.03.2024 – 24.03.2024
Дослідження технології ASP.NET Core Identity для реалізації автентифікації та авторизації	25.03.2024 – 31.03.2024
Реалізація удосконаленої моделі за допомогою ASP.NET Core Identity	01.04.2024 – 18.04.2024
Тестування реалізованої моделі	19.04.2024 – 28.04.2024
Оформлення пояснювальної записки згідно методичних рекомендацій	29.04.2024 – 12.05.2024
Подача пакету документів на розгляд ЕК	13.05.2024 – 18.05.2024

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект	Збільшення ефективності розробки процесу автентифікації та авторизації у вебдодатках.
Соціальний ефект	Розроблення моделі автентифікації та авторизації сприяє зменшенню ризиків порушень безпеки даних користувачів та ресурсу.

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ Сергій БУЧИК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняла
до виконання _____ Катерина ВЕНГРИНОВСЬКА
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 17.11.2023 р.

Термін подання дипломної роботи до ЕК 17.05.2024 р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Модель аутентифікації та авторизації в веб-додатках (за допомогою ASP.NET Core Identity)»: 90 сторінок, 39 рисунків та 58 літературних джерел.

Об’єкт дослідження – процес автентифікації та авторизації в вебдодатках.

Мета роботи – розробка моделі автентифікації та авторизації в вебдодатках за допомогою ASP.NET Core Identity.

Методи дослідження – аналіз, порівняння, синтез, експеримент.

У роботі досліджено сучасні технології автентифікації та авторизації. Проведено аналіз наявних методів та протоколів автентифікації та авторизації. Запропоновано удосконалену модель автентифікації та авторизації та реалізовано за допомогою ASP.NET Core Identity.

Наукова новизна: запропоновано удосконалену модель автентифікації та авторизації для найпростіших вебдодатків за допомогою технології ASP.NET Core Identity.

Актуальність теми: сучасні вебдодатки можуть містити та зберігати конфіденційну та персональну інформацію користувачів, таку як особисті дані, фінансові деталі, тому важливим є питання забезпечення безпеки задля унеможливлення несанкціонованого доступу. Традиційна модель автентифікації та авторизації, на даному етапі розвитку технології, не здатна повноцінно протистояти викликам автентифікації та авторизації. Використання сучасних технологій для реалізації моделі автентифікації та авторизації допомагає створити надійну та ефективну систему захисту вебдодатків. Тому використання декількох методів автентифікації та авторизації в поєднанні з сучасними технологіями є ефективним механізмом забезпечення безпеки вебдодатків.

Ключові слова: автентифікація, авторизація, ASP.NET Core Identity, модель безпеки, вебдодаток, управління доступом, безпека вебдодатків, ролі користувачів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AAA	–	Authentication, Authorization, and Accounting
ABAC	–	контроль доступу на основі атрибутів
ACL	–	Access Control List
API	–	Application Programming Interface
AS	–	Authentication Server
AWS	–	Amazon Web Services
CHAP	–	Challenge-Handshake Authentication Protocol
CSRF	–	Cross-Site Request Forgery
DAC	–	дискреційний контроль доступу
DES	–	Data Encryption Standard
DoS	–	Denial of Service
EAP	–	Extensible Authentication Protocol
IAM	–	управління ідентифікацією та доступом
IdP	–	Identity Provider
IP	–	Internet Protocol
JSON	–	JavaScript Object Notation
JWT	–	JSON Web Token
KDC	–	Key Distribution Center
LDAP	–	Lightweight Directory Access Protocol
LDAPS	–	LDAP over SSL
MAC	–	мандатний контроль доступу
MS-CHAP	–	Microsoft Challenge Handshake Authentication Protocol
NAS	–	Network Access Server
ODIC	–	OpenID Connect
OAuth	–	Open Authorization
OTP	–	одноразовий пароль
PAM	–	управління привілейованим доступом

PAP	–	Password Authentication Protocol
PBAC	–	контроль доступу на основі політик
PCBC	–	Propagating Cipher Block Chaining
PIN	–	Personal Identification Number
RADIUS	–	Remote Authentication Dial-In User Service
RBAC	–	контроль доступу на основі ролей
SAML	–	Security Assertion Markup Language
SASL	–	Simple Authentication and Security Layer
SK1	–	Session Key 1
SK2	–	Session Key 2
SP	–	Service Provider
SSL	–	Secure Sockets Layer
SSO	–	Single Sign-On
TCP	–	Transmission Control Protocol
TGS	–	Ticket Granting Server
TGT	–	Ticket Granting Ticket
TLS	–	Transport Layer Security
TOTP	–	Time-based One-time Password
UDP	–	User Datagram Protocol
XML	–	Extensible Markup Language

ВСТУП

У сучасному світі, де цифрові технології проникають у всі сфери життя, безпека даних в Інтернеті стає все більш важливою. З постійним розвитком Інтернету та веб-технологій, потреба в безпечних системах автентифікації та авторизації стає все більш актуальною. Вебдодатки стають все більш складними та функціональними, що вимагає відповідного рівня захисту.

Зі зростанням кіберзлочинності, важливість надійних систем автентифікації та авторизації не може бути недооціненою. Вони є першою лінією захисту від несанкціонованого доступу до даних користувачів та систем.

ASP.NET Core Identity є одним з найбільш популярних фреймворків для розробки вебдодатків на .NET. Вивчення та розробка моделей автентифікації та авторизації на основі цього фреймворку допоможе стандартизувати підходи до забезпечення безпеки в вебдодатках.

Вивчення та розробка моделей автентифікації та авторизації в вебдодатках за допомогою ASP.NET Core Identity є актуальною та важливою темою для дослідження, що відповідає сучасним тенденціям розвитку індустрії інформаційних технологій.

Метою даної кваліфікаційної роботи є розробка моделі автентифікації та авторизації в вебдодатках за допомогою ASP.NET Core Identity.

Для досягнення поставленої мети необхідно виконати такі *завдання*:

- дослідити теоретичні основи автентифікації та авторизації;
- дослідити особливості та можливості ASP.NET Core Identity;
- удосконалити та реалізувати модель автентифікації та авторизації.

Науковою новизною є удосконалення моделі автентифікації та авторизації для найпростіших вебдодатків за допомогою технології ASP.NET Core Identity.

Об'єктом дослідження є процес автентифікації та авторизації в вебдодатках.

Предметом дослідження є модель автентифікації та авторизації в вебдодатках, реалізована за допомогою ASP.NET Core Identity.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ

1.1 Визначення та основні принципи автентифікації та авторизації

Автентифікація та авторизація є двома важливими процесами управління інформаційною безпекою, що забезпечують безпеку інформації та систем. Автентифікація перевіряє особу користувача або служби, а авторизація визначає права доступу, які вони мають. Незважаючи на те, що ці два терміни звучать схоже, вони відіграють різні, але однаково важливі ролі в захисті програм і даних. Їх поєднання дає змогу отримати безпечне рішення, що вимагає правильного налаштування автентифікації та авторизації [1].

Автентифікація користувачів є основою для більшості типів контролю доступу та підзвітності користувачів. Автентифікація користувача – це процес перевірки вказаної ідентичності, яку заявляє або підтверджує системний об’єкт. Цей процес складається з двох етапів: ідентифікації та верифікації. Ідентифікація являє собою представлення ідентифікатора системі безпеки, а верифікація, в свою чергу, надає або генерує автентифікаційну інформацію, яка підтверджує зв’язок між об’єктом та ідентифікатором [2].

Процес автентифікації ставить на меті перевірити, чи є хтось або щось тим, за кого себе видає. Існує багато форм автентифікації, що наявні не лише в інформаційних технологіях, а і, наприклад, у сфері мистецтва, де застосовується процес, що підтверджує, що картина чи скульптура є роботою конкретного художника; іншим прикладом може слугувати використання урядом певних процесів автентифікації валюти для захисту її від підробок. Можна підсумувати та визначити, що автентифікація захищає цінні речі, що в інформаційну епоху становлять системи та дані [1].

Авторизація являє собою процес надання користувачеві дозволу на доступ до певного ресурсу або функції. Цей термін часто використовується як взаємозаміна

контролю доступу або привілей користувача. Найпростішим прикладом авторизації можна назвати надання доступу до завантаження певного файлу на сервері. У безпечному інформаційному середовищі процес авторизації повинен завжди слідувати за процесом автентифікації, оскільки користувач повинен спочатку підтвердити справжність та достовірність особи, і лише після цього адміністративна складова організації повинна надати доступ до запитуваних ресурсів [3].

Для обов'язково впровадження та використання авторизації можна назвати дві причини, а саме: захист даних та дотримання законодавства. Система безпеки організації повинна бути направлена не лише на захист від зовнішніх порушників, а й передбачати правила захисту для внутрішніх працівників. Якщо працівник загубив або викрав дані, важливо контролювати, які дані можуть бути вразливими, тому звідси можна сформулювати одну з цілей авторизації – гарантування того, що кожен працівник має доступ лише до тих систем та інформації, які йому потрібні для виконання своєї роботи, а не до всіх бізнес-даних, іншими словами повинен бути забезпечений принцип мінімальних привілеїв. Таким чином, якщо злодіям вдасться викрасти облікові дані, вони будуть обмежені в обсязі інформації, до якої зможуть отримати доступ. Інша причина, що пов'язана з дотриманням норм та вимог законодавства є не менш важливою, оскільки уряд ставить певний ряд вимог для обробки та зберігання конфіденційної чи таємної інформації. Одним з прикладів можна навести медичні дані пацієнтів, оскільки не всі працівники медичної установи повинні мати доступ до інформації про стан здоров'я чи інших персональних даних пацієнтів [4].

Як було зазначено вище, автентифікація та авторизація можуть звучати однаково, але кожен з них відіграє різну роль у захисті систем і даних. На жаль, люди часто використовують обидва терміни як взаємозамінні, оскільки вони обидва стосуються доступу до системи, однак, це різні процеси. Підсумовуючи можна сказати, що один з них перевіряє особу користувача або сервіс перед тим, як надати йому доступ, а інший визначає, що він може робити після отримання доступу [1].

Незважаючи на те, що автентифікація та авторизація абсолютно різні поняття, дещо подібне між ними все-таки є. Ця подібність проявляється в тому, що вони є

двома частинами одного базового процесу, що забезпечує доступ до певного ресурсу, тому ці два терміни часто плутають через однакову абревіатуру «auth». Автентифікація та авторизація також подібні тим, що використовують ідентичність. У першому випадку перевіряється особа (ідентичність) перед наданням доступу, а у другому – використання цієї перевіреної особи (ідентичності) для управління доступом [1].

Отже, підсумовуючи можна визначити наступні головні відмінності між автентифікацією та авторизацією [5, 6]:

- в процесі автентифікації ідентичність користувача перевіряється для надання доступу до системи, а в процесі авторизації перевіряються повноваження користувача для доступу до ресурсів;
- головна ціль автентифікації – це перевірити особу користувача, що гарантує доступ до системи лише ідентифікованим користувачам;
- авторизація користувачів гарантує, що тільки авторизовані користувачі можуть отримати доступ до необхідних їм активів і тільки в тому обсязі, що дозволений системою;
- під час процесу автентифікації користувачі перевіряються, а під час процесу авторизації – підтверджуються;
- автентифікація визначає чи є особа користувачем системи чи ні, а авторизація визначає який доступ має користувач;
- процес авторизації завжди відбувається після процесу автентифікації;
- для процесу автентифікації зазвичай потрібні дані користувача, а для процесу авторизації потрібні привілеї або рівні безпеки користувача;
- автентифікація користувача здійснюється за допомогою імені користувача, паролю, розпізнавання обличчя або відбитка пальця та інших методів, в той час як авторизація користувачів відбувається за допомогою прав доступу до ресурсів використовуючи попередньо визначені ролі;
- облікові дані для автентифікації можуть бути частково змінні за бажанням користувача, а права доступу, що використовуються для авторизації не можуть бути

змінені самостійно користувачами, оскільки вони надаються власником системи і тільки він має доступ до їх зміни.

1.2 Методи автентифікації та авторизації

Методи автентифікації та авторизації є важливими компонентами безпеки інформації. Вони відіграють ключову роль у забезпеченні того, що тільки відповідні користувачі мають доступ до конкретних ресурсів та систем. Методи автентифікації та авторизації можуть варіюватися від простих до складних, в залежності від рівня безпеки, який потрібен для конкретної системи або додатка.

Основними методами автентифікації, які можуть використовуватися як самостійно так і в комбінації є: автентифікація на основі того, що відомо користувачу; автентифікація на основі того чим володіє користувач; автентифікація на основі унікальних біологічних характеристиках людини; багатофакторна автентифікація; автентифікація на основі сертифікатів [2, 7, 8].

Автентифікація на основі того, що відомо користувачеві включає в себе пароль, персональний ідентифікаційний номер (Personal Identification Number, PIN) або відповіді на заздалегідь підготовлений набір запитань. Найпоширенішим варіантом є пароль. Цей тип вимагає, щоб користувач створив пароль для свого облікового запису, який потім хешується за допомогою алгоритмів хешування, таких як SHA-1, Bcrypt тощо, і в такому вигляді зберігається в базі даних. Хешування паролю – це додатковий захист ідентифікатора користувача у разі компрометації бази даних. Цей метод має певні особливості використання [2, 9]:

- використання ідентифікаторів (логін та пароль), що відомі лише користувачу;
- безпечне зберігання паролів (хешування паролів);
- встановлення вимог щодо довжини паролю та його складності (кількість символів, використання літер різного регістру, використання спеціальних символів та цифр);

- встановлення ліміту невдалих спроб і блокування акаунту після використаної кількості невдалих спроб;
- використання серії запитань для відновлення доступу у випадку, якщо користувач не пам’ятає вірний пароль;
- створення і керування користувацькими сесіями після успішної автентифікації.

Цей метод не лише захищає від неавторизованого доступу, але й є найпростішим у використанні та впровадженні, проте має ряд недоліків, а саме: використання слабких паролів, оскільки користувачам важко запам’ятати складний пароль; підвищений ризик фішингу, соціальної інженерії, атак перебору та перехоплення паролю. Також системи, які використовують цей тип автентифікації повинні підлягати регулярному аудиту.

Наступним методом автентифікації є автентифікація на основі того, чим володіє користувач. До цього типу належать криптографічні ключі, електронні ключі, смарт-картки та фізичні ключі. Цей метод також відомий як автентифікація на основі токенів. За допомогою цього типу автентифікації користувачі можуть один раз ввести свої облікові дані та отримати у відповідь унікальний зашифрований рядок випадкових символів. Метою механізму токенів є унеможливлення повторного використання паролів за допомогою генерації нового пароля при кожному використанні. Токени можна розглядати як міст між статичною парольною автентифікацією та більш досконалим методом автентифікації. Це полегшує міграцію застарілих додатків, які були розроблені, щоб покладатися лише на паролі. Цей метод має наступні особливості використання [2, 7, 9, 10]:

- користувач повинен мати фізичний об’єкт або інформацію для доступу;
- користувач повинен мати фізичний доступ до об’єкту для автентифікації;
- включає аспект вимірювання, коли система перевіряє, чи є у користувача потрібний об'єкт.

Головною перевагою цього методу є те, що немає необхідності використовувати статичний пароль, що забезпечує високу безпеку системи.

Недоліками даного методу є наступні: втрата, крадіжка та можливість підроблення об’єкту; висока вартість впровадження та підтримки даного методу.

Ще одним методом автентифікації є автентифікація на основі унікальних біологічних характеристик людини. Цей метод поділяється на статичну та динамічну біометрію. До статичної біометрії належить розпізнавання за відбитками пальців, сітківкою ока та обличчям. Інший тип, динамічна біометрія, включає розпізнавання за зразком голосу, характеристиками почерку та ритмом друку. До особливостей використання цього методу можна віднести наступне [2, 7, 9]:

- унікальність ідентифікатора;
- вимагає фізичної присутності користувача для проведення автентифікації.

Однією з головних переваг є висока безпека, оскільки біологічні характеристики людини є унікальними та невідтворними. Також цей метод забезпечує надає можливість використання у різних сферах, захист від фішингу, широкий вибір ідентифікаторів, зручність та швидкість. Системи з біометричним типом автентифікації вимагають високої ціни та складності впровадження, а також захисту та зберігання біометричних даних, що підвищує вартість та підтримку обладнання для системи.

Багатофакторна автентифікація – це розширена і вдосконалена версія базової автентифікації. Вона має кілька рівнів та етапів автентифікації, які базуються на паролі, одноразовому паролі (One-Time Password, OTP), перевірці електронної пошти, перевірці унікальної ідентичності або навіть іноді використовується спеціальне запитанні. Ці рівні додаються для формування єдиної системи автентифікації, і користувач повинен пройти їх усі, щоб підтвердити свою автентичність. Особливостями використання цього методу є [7, 8, 9]:

- використання декількох факторів для автентифікації користувача, такі як пароль, біометрія, смарт-картка, розпізнавання за голосом і т.ін.;
- обов’язкове представлення додаткового фактору автентифікації;
- можливість вибирати різноманітні автентифікаційні методи в залежності від вимог безпеки та специфікації системи.

Цей метод також забезпечує високий рівень безпеки системи, оскільки дозволяє використовувати різноманітні фактори автентифікації, зменшує ризик використання атак перебору та соціальної інженерії. Проте цей метод може виглядати дещо складним з боку користувача, тому необхідно буде витратити додаткові зусилля для навчання користувачів. Також є можливість втрати додаткового фактору. Інфраструктура системи, що підтримує такий тип автентифікації вимагає кошти на створення та якісне підтримання середовища.

Ще одним методом автентифікації є автентифікація на основі сертифікатів. Такі сертифікати містять цифрову ідентифікацію користувача, включаючи відкритий ключ, і цифровий підпис центру сертифікації. Тільки центр сертифікації може видавати цифрові сертифікати для підтвердження права власності на відкритий ключ. Основними особливостями використання цього методу є [7, 9]:

- процес генерування та видачі сертифікатів повинен проводиться надійно та централізовано;
- забезпечення безпечного зберігання та керування закритими ключами;
- розповсюдження відкритих ключів для перевірки сертифікатів та підписів;
- сертифікати мають обмежений термін дії і повинні періодично поновлюватися.

Перевагами даного методу є: високий рівень безпеки; централізоване керування; захист від фішингу та атак типу людина по-середині. Цей метод має ряд недоліків, а саме: складність впровадження, вартість управління, втрата ключа або сертифікату, сумісність та вимоги до зберігання ключів.

Існують різні стратегії авторизації, що використовуються в системах під час розгортання додатків. Найвідоміші з них – це контроль доступу на основі ролей (Role-Based Access Control, RBAC) та контроль доступу на основі атрибутів (Attribute-Based Access Control, ABAC). Існує багато інших альтернатив, а саме дискреційний контроль доступу (Discretionary Access Control, DAC), мандатний контроль доступу (Mandatory Access Control, MAC), контроль доступу на основі політик (Policy-based access control, PBAC), управління привілейованим доступом (Privileged Access

Management, PAM). Кожен з цих методів допоможе розробникам додатків впоратися з різними вимогами до авторизації та сервісами авторизації [11, 12, 13].

Контроль доступу на основі ролей (RBAC) – це метод авторизації, який визначає доступ користувачів на основі їхньої ролі в організації. Метою даного методу є запобігання доступу неавторизованих користувачів до конфіденційної інформації, яка їм не потрібна або яку вони не повинні бачити, незалежно від того, чи є вона локальною або віддаленою. Після автентифікації користувача RBAC визначає, до чого він може отримати доступ, виходячи з його ролі в організації або системі. Ця роль може визначатися посадою, відділом, місцезнаходженням або конкретними обов'язками користувача. Завдяки цьому методу користувачам системи надається доступ лише до тієї інформації, яка безпосередньо стосується їхньої ролі в системі. Доступ надається на основі таких факторів, як повноваження, відповідальність і компетенція. Використовуючи цей метод, працівники можуть отримати доступ лише до тієї інформації, яка необхідна для ефективного виконання їх обов'язків. Усі моделі контролю доступу на основі ролей містять такі основні елементи: адміністратори (користувачі, які визначають ролі та надають дозволи); ролі (користувачі, об'єднані в групи на основі роботи, яку вони виконують) та дозволи (дії та доступ, надані кожній ролі, які визначають, що ці ролі можуть робити). Цей метод дозволяє адміністраторам створювати, призначати та контролювати дозволи доступу для кожної ролі в системі [11].

До переваг контролю доступу на основі ролей відноситься[11, 14]:

- дозволяє встановлювати дозволи на основі ролі користувача замість того, щоб керувати дозволами для кожного користувача окремо;
- спрощене налаштування, оскільки заздалегідь визначені ролі роблять цей метод практично готовим до використання інформаційно–технічним відділом, що зменшує роботу при адаптації нових співробітників;
- швидкі зміни та звільнення, коли працівник залишає компанію або змінює роботу, його доступ до ресурсів може бути відкликаний або змінений на нову роль;
- відповідність нормативним вимогам, оскільки коли доступ базується на ролях, адміністратори з меншою ймовірністю припускаються помилок, надаючи не

тим людям доступ до конфіденційних даних, що може покращити дотримання нормативних вимог;

- централізованість.

Недоліками даного методу є [11, 14]:

- комплексне розгортання, оскільки мережа обов'язків та відносин у великих підприємствах робить визначення ролей настільки складним, що це спричинило нову необхідність – розробку ролей;
- більше ролей і більш деталізовані ролі забезпечують більший захист, але адміністрування системи навпаки ускладняється; призначення користувачам занадто великої кількості ролей також підвищує ризик надання користувачам надмірних прав.

Контроль доступу на основі атрибутів (ABAC) – це метод авторизації, що використовує атрибути або характеристики для динамічного визначення привілеїв доступу користувача. Цей метод зосереджується на конкретних атрибутах, які впливають на політики. Він використовує атрибути, які можна прикріпити до користувача, ресурсу, об'єкта або всього середовища. Об'єкт авторизується, якщо система автентифікації виявляє, що всі атрибути, визначені в політиці, відповідають дійсності [11].

До переваг контролю доступу на основі атрибутів можна віднести [11, 15]:

- деталізацію, оскільки для визначення зв'язків між користувачами та ресурсами використовуються атрибути, а не ролі, адміністратори можуть створювати точно націлені правила без необхідності створювати додаткові ролі;
- гнучкість, оскільки замість того, щоб змінювати правила чи створювати нові ролі, адміністраторам потрібно лише призначити відповідні атрибути новим користувачам або ресурсам;
- адміністратори можуть змінювати атрибути та створювати контекстно-залежні правила відповідно до своїх потреб.

Недоліками даного методу є [11, 15]:

- підвищена складність, оскільки адміністраторам потрібен час і ресурси, щоб вручну визначити та призначити атрибути, окрім створення механізмів супутніх політик;

- проблеми масштабованості, це пов’язано з вищезазначеними проблемами конфігурації, його великим цифровим слідом і великою кількістю користувачів, якими потрібно керувати;
- велика кількість дозволів ускладнює комплексний аудит системи.

Дискреційний контроль доступу (DAC) – це метод, який надає користувачам контроль над правами доступу до своїх ресурсів, дозволяючи власникам інформаційних систем і даних вирішувати, хто може отримати доступ до відповідних ресурсів і який рівень доступу вони можуть мати. Цей підхід підтримує принцип найменших привілеїв – концепцію, яка передбачає надання користувачам найменшого обсягу доступу, необхідного для виконання їхньої роботи. Оскільки цей метод покладається на людські рішення щодо надання доступу, власник інформаційної системи або даних повинен ретельно перевіряти запити на доступ, щоб не допустити надання надмірно широких прав [16].

Дискреційний контроль доступу децентралізує прийняття рішень щодо безпеки на користь власників ресурсів. Такі системи використовують списки контролю доступу (Access control lists, ACLs), щоб визначити, хто може отримати доступ до ресурсу. Ці таблиці пов’язують індивідуальні та групові ідентифікатори з їхніми привілеями доступу. Списки ACL містять або користувачів, або попередньо визначені групи користувачів і відповідні їм рівні доступу. Ці рівні зазвичай включають доступ на читання, запис і виконання, що дозволяє користувачеві переглядати, змінювати або запускати процес чи програму відповідно. У більшості операційних систем функція спільного доступу є різновидом цього методу. Для кожного документа, яким ви володієте, ви можете встановити привілеї на читання/запис і вимоги до паролів у таблиці окремих осіб і груп користувачів [14, 16].

До переваг дискреційного контролю доступу можна віднести: гнучкість, оскільки дозволяє власникам ресурсів контролювати, хто може отримати доступ до їхніх ресурсів і які дії вони можуть виконувати; простий та ефективний спосіб керування доступом; можливість налаштувати дозволи доступу для конкретних осіб або груп відповідно до різних потреб організації; простота впровадження; підходить для невеликих середовищ [17].

Недоліками дискреційного контролю доступу є: небезпечне успадкування даних, наприклад, якщо батьківська папка має слабкі дозволи, ці дозволи можуть бути успадковані дочірніми папками, що може призвести до витоку конфіденційних даних; конфліктуючі дозволи можуть надати користувачеві надмірні або недостатні привілеї, оскільки користувач може бути членом кількох вкладених робочих груп; адміністратори безпеки не можуть легко бачити, як ресурси розподіляються в організації; за відсутності належної процедури реєстрації та моніторингу може бути складно виявити несанкціонований доступ або зміни до ресурсів; соціальна інженерія; довірені користувачі з правами доступу можуть навмисно або випадково зловживати своїми привілеями, що призведе до порушень безпеки [14, 17].

Мандатний контроль доступу (MAC) – це централізований метод забезпечення безпеки. Недискреційна система залишає контроль над політиками доступу за централізованою адміністрацією безпеки. Цей метод працює, застосовуючи мітки безпеки до ресурсів та осіб. Ці мітки безпеки складаються з двох елементів: класифікація та допуск; відділення. Мандатний контроль доступу спирається на систему класифікації (обмежений, секретний, надсекретний тощо), яка описує чутливість ресурсу. Допуски користувачів визначають, до яких видів ресурсів вони можуть отримати доступ. Відділення ресурсу описує групу людей (відділ, команда проекту тощо), яким дозволено доступ. Відділення користувача визначає групу або групи, в яких він бере участь. Користувач може отримати доступ до ресурсу, тільки якщо його мітка безпеки збігається з міткою безпеки ресурсу. Цей метод найкраще підходить для захисту найбільш чутливих ресурсів організації [14].

До переваг мандатного контролю доступу можна віднести: розмежування, мітки безпеки обмежують доступ до кожного ресурсу лише підмножиною користувачів; адміністратори системи встановлюють загальноорганізаційні політики, які користувачі не можуть скасувати, що полегшує їх застосування. Недоліками даного методу є: обмеження співпраці, оскільки метод працює обмежуючи комунікацію, тому організаціям з високим рівнем колаборації може знадобитися менш обмежувальний підхід; спеціальна організаційна структура повинна керувати створенням і підтримкою міток безпеки [14].

Контроль доступу на основі політик (PBAC) – це метод авторизації, що використовує політики для визначення привілеїв доступу користувачів. Політики можуть бути засновані на різних факторах, включаючи атрибути користувача, атрибути ресурсу, атрибути дії та атрибути середовища. Адміністратори створюють політики доступу на основі ролей і атрибутів користувачів і встановлюють правила щодо цих ролей і атрибутів, які динамічно визначають доступ. Рішення приймаються відповідно до контексту і ризику, коли запитується доступ. Ці політики також визначають, які дозволи мають користувачі після отримання доступу до ресурсу. Вони можуть визначати, чи користувачі мають доступ лише для читання, чи можуть вони вносити зміни до об’єкта або ділитися ним з іншими [11].

До переваг контролю доступу на основі політик можна віднести: гнучкість та швидкість, оскільки адміністратори системи більший контроль над рівнем доступу і можуть додавати, видаляти або редагувати дозволи для великої кількості користувачів одночасно; політики можуть враховувати широкий спектр динамічних атрибутів і контекстних елементів управління, таких як обмеження доступу, прив’язані до часу або місцезнаходження, тому цей метод є адаптивним; політики є зрозумілими для людини і полегшують перегляд взаємозв’язку між ідентичностями та ресурсами. Недоліками цього методу є складність та вартість впровадження системи на основі цього методу [11].

Управління привілейованим доступом (PAM) використовує комбінацію людей, процесів і технологій для захисту можливостей адміністраторів і досвідчених користувачів, а також для захисту від тих, хто може саботувати роботу системи за допомогою привілейованого облікового запису. Адміністратори мають право вносити значні зміни в загальне середовище, наприклад, додавати або видаляти користувачів, оновлювати і встановлювати апаратне і програмне забезпечення, виконувати діагностику несправностей, створювати резервні копії даних і керувати мережевою безпекою, оскільки адміністратори мають повноваження суттєво змінювати мережеве середовище, доступ до цих типів облікових записів повинен бути додатково перевірений. Цей метод є різновидом контролю доступу на основі ролей (RBAC) і важливим компонентом загального протоколу безпеки управління ідентифікацією та

доступом (Identity and Access Management, IAM). Користувачі з привілейованим доступом мають доступ до дуже чутливих і обмежених частин технологічної системи, які закриті для звичайних користувачів. Цей метод допомагає захистити систему від ризиків компрометації системи за допомогою привілейованих користувачів, додаючи додаткові рівні захисту до привілейованих облікових записів, і існує кілька різних способів її використання: застосування багатофакторної автентифікації і додаткових рівнів автентифікації для привілейованих користувачів; часта перевірка дій привілейованих користувачів з боку інших облікових записів адміністраторів; зберігання облікових даних привілейованих користувачів у високо захищених сховищах паролів; надійна система моніторингу, реєстрації, аудиту та звітності про сеанси користувачів для автоматичного виявлення аномальної активності [11].

До переваг управління привілейованим доступом можна віднести: моніторинг привілейованих користувачів на предмет незвичної поведінки стає простішим; спрощує відкликання привілеїв, коли вони більше не потрібні користувачам, таким чином запобігаючи колекціонуванню привілеїв доступу; зменшення ризику загроз. Недоліками даного методу є складність та вартість впровадження [14].

1.3 Протоколи автентифікації та авторизації

Протоколи автентифікації визначають спосіб взаємодії заявника (користувач, що намагається отримати доступ до ресурсу) та верифікатора (суб’єкта, що автентифікує заявника). Ці протоколи включають обмін повідомленнями, що перевіряють дійсність служби автентифікації та перевіряють чи володіє заявник відповідним маркером для автентифікації своєї особи. Інформація, яку надає користувач для автентифікації визначається відповідним методом автентифікації системи. Протоколи автентифікації – це стандарти, за якими верифікатори розуміють ці методи, і можуть діяти різними способами для підтвердження автентифікації. Протоколи авторизації – це набір правил і процедур, які використовуються для визначення та контролю доступу до ресурсів або послуг, вони визначають, які користувачі мають право отримувати доступ до певних ресурсів і як цей доступ

контролюється. Протоколи авторизації використовуються для перевірки ідентичності користувача та його прав на доступ до ресурсів. Вони забезпечують механізми для перевірки, чи має користувач необхідні дозволи для виконання певних операцій або отримання певних даних [18].

Протоколи автентифікації визначають тип інформації, необхідний службі автентифікації для перевірки користувача, і спосіб її доставки. Протоколи гарантують, що користувач та верифікатор обмінюються однаковим форматом повідомлення для забезпечення процесу автентифікації. Різні протоколи автентифікації використовують власні правила для проведення процесу. Проте є певні стандартні техніки, що використовуються більшістю з наявних протоколів, як самостійно так і в комбінації. Серед таких технік є [18]:

- шифрування: передбачає шифрування даних таким чином, щоб їх могли зрозуміти лише сторони, що мають доступ до спеціального ключа, допомагає запобігти витоку чи крадіжці конфіденційної інформації, протоколи використовують інструменти шифрування для передавання інформації між користувачами та верифікаторами;
- хешування: використовує алгоритми для зменшення рядків даних до більш керованих розмірів та уникнення передачі даних у відкритому вигляді, подібно до шифрування, лише сторони, що задіяні в процесі автентифікації, знайомі з алгоритмом і можуть зрозуміти рядок хешованих даних, захист аналогічний до шифрування;
- цифрові підписи: підписами є зашифровані штампи автентифікації, які доводять, що інформацію не було змінено в процесі передачі, у протоколах автентифікації використовуються для підтвердження того, що це справді заявник намагається отримати доступ до служби, а не зловмисник.

Найбільш відомими протоколами автентифікації та авторизації є: Lightweight Directory Access Protocol (LDAP), Remote Authentication Dial-In User Service (RADIUS), Kerberos, OAuth, OpenID Connect, Security Assertion Markup Language (SAML), HTTP Basic Authentication.

1.3.1 HTTP Basic Authentication

HTTP Basic Authentication є одним з найпростіших протоколів автентифікації. Цей протокол є простим та широко використовується для автентифікації користувачів в вебдодатках. Він заснований на HTTP протоколі і передбачає зв’язок між клієнтом і сервером за допомогою заголовка HTTP, де сервер запитує облікові дані користувача для автентифікації, клієнт у відповідь надає відповідну інформацію в заголовку. Ця концепція базується на веб-автентифікації за допомогою стандартів HTTP для забезпечення безпеки інформації користувачів. Більш захищеною версією є HTTPS, де S означає рівень безпеки (Security Socket Layer, SSL) для використання шифрування під час зв’язку [19].

HTTP представляє загальну структуру для контролю доступу користувачів до веб-ресурсів. Ця структура залежить від заголовків автентифікації, які поділяються на два типи: WWW-Authenticate та Proxy-Authenticate. Нижче представлений процес того, як HTTP Authentication працює з двома типами запитів та забезпечує виконання процесу автентифікації [19]:

1) клієнт робить запит на доступ до ресурсу як анонімна особа, сервер не має жодної інформації про клієнта, що надав цей запит;
2) після виявлення відвідувача сервер відповідає клієнту статусом відповіді 401 (неавторизований) як виклик для перевірки особи та інструкцій щодо перевірки в заголовку (наприклад, заголовок WWW-Authenticate);
3) клієнт відповідає серверу, використовуючи необхідні облікові дані, ім’я користувача та пароль, для автентифікації та доступу до ресурсу;
4) якщо використовується проксі-сервер, то проксі-сервер проводить автентифікацію від імені клієнта для доступу до ресурсу;
5) після отримання облікових даних клієнта, сервер перевіряє їх і повідомляє клієнта про статус автентифікації.

Перевагами даного протоколу є: простота реалізації; один з найбільш поширених методів автентифікації і підтримується більшістю веб-серверів та

клієнтських програм; сумісність використання, оскільки працює з усіма сучасними веб-браузерами, що робить його зручним для використання в вебдодатках.

До недоліків даного протоколу можна віднести наступне: відсутність шифрування, оскільки облікові дані передаються у відкритому текстовому вигляді без застосування методів шифрування, що робить їх вразливими до перехоплення; відсутність вбудованого механізму відновлення сесій, що означає, що кожен запит повинен містити облікові дані користувача, що призводить до неефективного використання пропускної здатності мережі; не надає гнучкості в управлінні правами доступу; не надає можливості зміни пароля без повторної автентифікації; вразливість до атак перехоплення.

1.3.2 Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) – це протокол служби каталогів, який працює на рівні над стеком TCP/IP, надає механізм, який використовується для підключення, пошуку та зміни каталогів Інтернету. Служба каталогів LDAP базується на моделі клієнт-сервер. Функцією даного протоколу є надання доступу до існуючого каталогу. Модель даних (дані та простір імен) LDAP подібна до моделі служби каталогів X.500 OSI, але з меншими вимогами до ресурсів [20].

Даний протокол може використовуватися для досягнення цілей у наступних сферах [21]:

- автентифікація: LDAP зазвичай використовується як централізований механізм автентифікації у великих організаціях, а саме замість того, щоб мати окремі облікові записи користувачів у кожній системі, LDAP дозволяє використовувати єдиний вхід для всієї організації;
- авторизація: LDAP надає центральну базу даних, яка містить інформацію про користувачів, групи та засоби контролю доступу, цю інформацію можна використовувати для контролю доступу до різноманітних ресурсів в організації;

- служби адресної книги та каталогу: LDAP часто використовується для зберігання та доступу до контактної інформації, такої як адреси електронної пошти, номери телефонів та інша інформація про людей в організації;
- інтеграція додатків: LDAP може використовуватися додатками для зберігання та доступу до конфігураційної інформації, налаштувань користувача та інших даних, які можуть спільно використовувати кількома системами.

LDAP надає можливість використання наступних механізмів безпеки [22, 23]:

- автентифікація за допомогою Bind: забезпечує простий метод, який підтримує анонімні, неавтентифіковані механізми та механізми ім’я/пароль, а також метод Simple Authentication and Security Layer (SASL), який підтримує широкий спектр механізмів автентифікації, використовуються для автентифікації клієнтів (і користувачів або програм, що стоять за ними) на сервері каталогів, для встановлення ідентичності авторизації, яка використовуватиметься для наступних операцій, оброблених у цьому з’єднанні, і для визначення версії протоколу LDAP, яку використовуватиме клієнт;
- механізми для підтримки спеціальних засобів контролю доступу постачальника (LDAP не пропонує стандартний засіб контролю доступу);
- сервіс цілісності даних за допомогою рівнів безпеки в Transport Layer Security (TLS) або SASL;
- служба конфіденційності даних за допомогою рівнів безпеки в TLS або механізми SASL;
- обмеження використання ресурсів сервера за допомогою налаштованих адміністратором обмежень, налаштованих на сервері;
- автентифікація сервера за допомогою протоколу TLS або SASL механізму.

Для взаємодії з сервером LDAP використовуються протоколи, такі як LDAP, LDAPS (LDAP over SSL) та StartTLS для забезпечення безпеки передачі даних. Як було зазначено для операції Bind може використовуватися проста (базова) або SASL-автентифікація. У випадку використання простої автентифікації обліковий запис, який потрібно автентифікувати, ідентифікується доменним ім’ям запису для цього облікового запису, а підтвердження ідентичності надається у вигляді пароля, що

передається без будь-якої форми обфускації, тому рекоменлується використовувати просту автентифікацію лише через зашифроване з’єднання (наприклад, захищене SSL/TLS або розширеною операцією StartTLS). SASL-автентифікація надає можливість підключити будь-який тип автентифікації до протоколу LDAP, виконується за допомогою імені механізму SASL і закодованого набору облікових даних. Деякі механізми SASL можуть вимагати, щоб клієнт і сервер обмінювалися інформацією кілька разів (через кілька запитів на зв’язування та відповідей), для успішного завершення процесу автентифікації [23].

Даний протокол володіє певними перевагами використання, а саме: централізоване керування ідентифікацією, оскільки дозволяє зберігати облікові дані користувачів, груп та інших об’єктів у централізованому директорії, що спрощує управління обліковими записами користувачів та забезпечує однаковий доступ до інформації для всіх додатків і систем; масштабованість та гнучкість, оскільки розроблений для роботи в розподілених системах та здатний обробляти великі обсяги даних та велику кількість запитів, також дозволяє визначати власні схеми даних та атрибути, що надає можливість налаштовувати директорію відповідно до конкретних потреб організації або системи; стандартизація та сумісність, оскільки є стандартним протоколом доступу до директорій, що підтримується багатьма різними вендорами, що забезпечує сумісність та інтеграцію з різними системами, які підтримують даний протокол; можливість підвищення безпеки за рахунок шифрування даних, використовуючи протоколи LDAPS (LDAP over SSL) або StartTLS.

Проте незважаючи на гнучкість використання протоколу та його інших особливостей, LDAPє вразливим до певних типів активних та пасивних атак. Серед таких атак є: несанкціонований доступ до даних директорії через операцію пошуку даних; несанкціонований доступ до даних каталогу шляхом моніторингу доступу інших користувачів; несанкціонований доступ до багаторазової автентифікаційної інформації клієнта шляхом моніторингу доступу інших осіб; несанкціонована модифікація даних каталогів; несанкціонована модифікація конфігураційної інформації; відмова в обслуговуванні (DoS); спуфінг, що має на меті заволодіти інформацією користувача або клієнта видаючи себе за сервер каталогу, але не є,

змусити сервер каталогів повірити, що інформація надійшла від певного клієнта, коли насправді вона надійшла від ворожої організації; захоплення контроль над встановленим сеансом протоколу. До інших недоліків можна віднести: складність налаштування та керування; повільна швидкість доступу до даних; залежність від мережевого з'єднання [22].

1.3.3 Remote Authentication Dial-In User Service

Remote Authentication Dial-In User Service (RADIUS) – це стандартний Інтернет-протокол, який надає централізовану автентифікацію, облік і служби керування IP для користувачів віддаленого доступу в розподіленій комутованій мережі, що захищає мережу від несанкціонованого доступу. Розподілений протокол обміну інформацією використовує клієнт-серверну модель. Також до цього протоколу часто застосовується абревіатура AAA, що означає автентифікація, авторизація та облік (Authentication, Authorization, and Accounting). RADIUS – це стандартний протокол, який підтримується багатьма пристроями, що забезпечує високу сумісність, а також є найбільш широко використовуваним протоколом AAA в мережах, використовує протокол дейтаграм користувача (User Datagram Protocol, UDP) як протокол передачі. Також підтримує багатопотоковість, що забезпечує можливість одночасної автентифікації великої кількості користувачів [24].

Згідно з RFC 2865, що визначає протокол RADIUS, основними особливостями даного протоколу є [24]:

- клієнт-серверна модель: сервер мережевого доступу (Network Access Server, NAS) працює як клієнт RADIUS та відповідає за передачу інформації про користувача визначеним серверам RADIUS, а згодом діє на основі отриманої відповіді від серверів. В свою чергу, сервери RADIUS відповідають за отримання запитів на підключення користувача, автентифікацію користувача, а потім повертають інформацію про конфігурацію, що необхідна клієнту для надання послуг користувачеві;

- мережева безпека: транзакції між клієнтом і сервером RADIUS автентифікуються за допомогою спільного секрету, що ніколи не передається через мережу, також будь-які паролі користувачів передаються між клієнтом та сервером лише у зашифрованому вигляді, що унеможливлює визначення паролю користувача третьою особою в незахищеній мережі;
- сервер RADIUS може підтримувати різні механізми автентифікації користувача;
- розширюваність протоколу: всі транзакції складаються з 3-кортежів Attribute-Length-Value змінної довжини, нові значення атрибутів можуть бути додані без порушення наявних реалізацій протоколу.

Однією з особливостей даного протоколу є підтримка різних механізмів автентифікації, якщо серверу надано ім’я користувача та вихідний пароль, надані користувачем, він може підтримувати PAP (Password Authentication Protocol), CHAP (Challenge-Handshake Authentication Protocol), MS-CHAP (Microsoft Challenge Handshake Authentication Protocol), EAP (Extensible Authentication Protocol), вхід до UNIX та інші механізми автентифікації [25].

PAP – це простий протокол автентифікації, який використовується для перевірки ідентифікації користувача та його пароля під час встановлення з’єднання. У PAP, користувач надає своє ім’я користувача та пароль, які передаються в мережу у відкритому тексті. Сервер автентифікації перевіряє ці дані і повертає результат успішності автентифікації. В рамках протоколу RADIUS, він може використовуватися сервером RADIUS для перевірки облікових даних наданих користувачем [24, 25].

CHAP – протокол випробування автентичності користувача є іншим механізмом автентифікації, який може бути використаний у протоколі RADIUS. У процесі автентифікації за допомогою CHAP, щойно користувач намагається отримати доступ до мережевого ресурсу, сервер RADIUS надсилає випадковий запит на випробування автентичності (відомий як «виклик CHAP») клієнту, в свою чергу клієнт відповідає на цей виклик, використовуючи хеш свого пароля та інші дані, щоб створити відповідь на виклик, яку потім він передає на сервер. На сервері

відбувається перевірка отриманої відповіді від клієнта, якщо вона вірна, сервер надає користувачеві доступ до мережевих ресурсів. Одним з переваг CHAP є те, що пароль ніколи не передається через мережу у відкритому вигляді, замість цього, відбувається обмін хешами, що підвищує рівень безпеки. Також, CHAP використовується для багатопотокового випробування автентичності, що дозволяє підтримувати стабільне з'єднання навіть під час зміни пароля користувача або інших змін в його ідентифікаційних даних [24, 26].

MS-CHAP – це протокол автентифікації, що розроблений компанією Microsoft для використання в мережах з використанням точки доступу PPP (Point-to-Point Protocol). MS-CHAP є розширенням протоколу CHAP і має деякі відмінності та покращення порівняно з оригінальним CHAP. Основна мета MS-CHAP – це забезпечити безпечну автентифікацію користувачів на мережевому рівні шляхом використання взаємного виклику та відповіді. Протокол використовує аналогічний метод випробування, що і CHAP, але з додатковими можливостями та застосованими заходами безпеки. Основні характеристики MS-CHAP включають використання шифрування, автентифікацію взаємного виклику та відповіді, використання паролю або іншого обмеженого маркера доступу [27].

EAP – це протокол розширеної автентифікації, що дозволяє реалізувати різноманітні методи автентифікації користувачів у мережевих з'єднаннях. До основних переваг можна віднести розширюваність і здатність підтримувати різні механізми автентифікації, такі як пароль, сертифікати, біометричні дані тощо. EAP використовується в бездротових мережах (Wi-Fi), VPN-з'єднаннях, мобільних мережах і будь-яких інших сценаріях, де потрібна автентифікація користувачів. Протокол EAP не передбачає специфічного методу автентифікації, але надає загальну структуру для обміну повідомленнями між клієнтом і сервером автентифікації. Основні компоненти EAP включають в себе ініціалізацію, обмін повідомленнями автентифікації і завершення. Під час процесу автентифікації використовуються різні методи, наприклад, EAP-TLS (EAP Transport Layer Security), EAP-TTLS (Tunneled TLS), PEAP (Protected Extensible Authentication Protocol) тощо. [28]

Загальний процес протоколу можна описати наступним чином: пристрій, який працює як клієнт RADIUS, збирає інформацію про користувачів (наприклад, імена користувачів і паролі) і надсилає цю інформацію на сервер RADIUS, сервер RADIUS автентифікує користувача відповідно до інформації, а потім виконує авторизацію та облік для користувача [29].

Більш детально процес автентифікації, авторизації та обліку за допомогою RADIUS можна поділити на такі етапи [29]:

1) користувач хоче отримати доступ до мережі та надсилає клієнту RADIUS запит на підключення, що містить ім’я користувача та пароль;

2) клієнт RADIUS надсилає пакет Access-Request, що містить ім’я користувача та пароль, на сервер RADIUS;

3) сервер RADIUS перевіряє особу користувача: якщо ідентифікатор користувача є дійсним, сервер RADIUS відправляє клієнту RADIUS пакет Access-Accept, що дає дозвіл користувачеві на виконання подальших дій, цей пакет містить інформацію про авторизацію, оскільки протокол забезпечує як функції автентифікації, так і авторизації; якщо ідентифікатор користувача є недійсним, сервер RADIUS надсилає відповідь клієнту RADIUS пакетом Access-Reject, відхиляючи запит на доступ користувача;

4) клієнт RADIUS повідомляє користувача про результат автентифікації;

5) клієнт RADIUS приймає або відхиляє запит на доступ користувача відповідно до результату автентифікації, якщо запит на доступ користувача прийнято, клієнт RADIUS надсилає на сервер RADIUS пакет Accounting-Request (Start);

6) сервер RADIUS відповідає пакетом Accounting-Response (Start) і починає облік;

7) користувач отримує доступ до ресурсів мережі;

8) якщо ввімкнено облік у реальному часі, клієнт RADIUS періодично надсилає пакет Accounting-Request (Interim-Update) на сервер RADIUS, що запобігає невиправданому проведенню обліку, якщо користувач несподівано виходить із системи через невдачу надсилання пакета Accounting-Request (Stop), інтервал, через

який надсилаються пакети Accounting-Request (Interim-Update), можна встановити на клієнті;

9) сервер RADIUS відповідає пакетом Accounting-Response (Interim-update) і проводить облік у реальному часі;

10) користувач надсилає запит на вихід із системи, щоб припинити доступ до мережевих ресурсів;

11) клієнт RADIUS надсилає пакет Accounting-Request (Stop) серверу RADIUS;

12) сервер RADIUS відповідає пакетом Accounting-Response (Stop) і припиняє облік;

13) клієнт RADIUS сповіщає користувача про те, що доступ до мережі закінчується, і завершує доступ до мережевих ресурсів.

Варто відзначити, що етап 8 та 9 є необов'язковим і залежить від налаштування мережі та вимог системи.

Можливість обліку в протоколі RADIUS може використовуватися окремо від процесів автентифікації та авторизації RADIUS. Ця можливість дозволяє забезпечити збір та реєстрацію наступної інформації: мережевий трафік, час підключення, обсяг передачі даних тощо. Сервер RADIUS отримує повідомлення про використання ресурсу та реєструє відповідну інформацію для виставлення рахунків, аудиту та керування мережею. За допомогою цієї інформації адміністратори можуть отримати детальну інформацію про поведінку користувачів мережі та використання ресурсів, що в свою чергу допоможе краще управляти та підтримувати мережеві ресурси [30].

Перевагами даного протоколу є: простота реалізації, оскільки він має просту структуру, що сприяє швидкій та легкій реалізації в різних системах, також це зменшує витрати часу та ресурсів на розгортання та підтримку систем; даний протокол є розширюваним, оскільки дозволяє динамічно додавати нові атрибути до вже наявних пакетів без необхідності змінювати структуру, це також забезпечує гнучкість системи; підтримка багатопотоковості дозволяє обслуговувати одночасно велику кількість користувачів, що забезпечує доступність та надійність ресурсу, ця перевага також підтверджує широке використання даного протоколу, а особливо для

великих корпоративних мереж, що вимагають складної системи авторизації; безпека, що полягає в можливості шифрування інформації, це перешкоджає можливості перехоплення паролів під час передачі через мережу; підтримка обліку користувачів та використання ресурсів, ця функція є корисною для адміністраторів, оскільки допомагає управляти використанням ресурсів, а також для власників ресурсів ця інформація може бути корисною для подальших бухгалтерських обліків та розвитку.

Недоліками даного протоколу є: обмеженість безпеки, оскільки протокол може використовувати PAP, що передає дані у відкритому вигляді, це ставить під загрозу конфіденційні дані користувачів; незважаючи на те, що протокол підтримує багатопотоковість, при надмірній кількості користувачів і недостатній кількості ресурсів це може спричинити проблеми з продуктивністю та масштабованістю; RADIUS не має вбудованих засобів захисту від різноманітних атак, що може стати проблемою у сучасних середовищах; RADIUS хоч і надає основні функції автентифікації та авторизації, його можливості все одно можуть бути обмеженими для деяких сучасних вимог та сценаріїв використання.

1.3.4 Kerberos

Kerberos – це протокол для автентифікації запитів на обслуговування між надійними хостами в ненадійній мережі, наприклад в мережі Інтернеті. Kerberos був започаткований в рамках дослідницького проєкту Athena Массачусетського технологічного інституту на початку 1980-х років [31, 32].

Kerberos був розроблений задля вирішення проблеми, що полягає в наступній ситуації: у відкритому розподіленому середовищі, у якому користувачі робочих станцій бажають отримати доступ до сервісів на серверах мережі, які повинні мати можливість обмежувати доступ авторизованим користувачам і забезпечувати автентифікацію запитів на обслуговування, проте в такому середовищі не можна довіряти робочій станції у правильній ідентифікації її користувачів для доступу до мережевих служб. Це зокрема спричиняє наступні загрози з боку неавторизованого користувача: отримання доступу до певної робочої станції і видати себе за іншого

користувача, що працює з цієї робочої станції; зміна мережевої адреси робочої станції так, що запити, надіслані зі зміненої робочої станції, будуть сприйняті як ті, що надходять від підміненої станції; підслуховування телефонних розмов та використання атаки з відтворенням з метою отримання доступу до сервера та/або порушення його роботи. У будь-якому з цих випадків неавторизований користувач може отримати доступ до сервісів і даних, до яких він не повинен мати доступу. Замість того, щоб вбудовувати складні протоколи автентифікації на кожному сервері, Kerberos надає централізований сервер автентифікації, функція якого полягає в автентифікації користувачів на серверах і серверів для користувачів. Kerberos покладається виключно на симетричне шифрування, не використовуючи шифрування з відкритим ключем [2].

Kerberos є надійною службою взаємної автентифікації третьої сторони з єдиним входом. Це означає наступне [32]:

- безпека даного протоколу полягає в тому, що він ніколи не передає паролі через мережу у відкритому вигляді, також використовує унікальні квитки, що є обмеженими у часі криптографічними повідомленнями, що використовуються для підтвердження особи користувача певному серверу без надсилання паролів через мереже або їх кешування на жорсткому диску локального користувача;
- єдиний вхід означає, що кінцевим користувачам не потрібно щоразу проходити процес автентифікації до мережевих ресурсів, що підтримують Kerboros, оскільки після того, як користувач пройшов початкову автентифікацію в Kerboros, його облікові дані прозоро передаються на всі інші підтримувані ресурси, до яких надається доступ протягом дня;
- Kerberos працює через централізований сервер автентифікації, якому всі системи в мережі за своєю суттю довіряють, усі запити на автентифікацію проходять через цей сервер, тому до визначення застосовується термін довірена третя сторона;
- взаємна автентифікація визначається як процес, за якого як користувач, так і сервер підтверджують свою ідентичність один одному, цей механізм не лише перевіряє, що користувач є тим, за кого він себе видає, але також підтверджує, що сервер, із яким він взаємодіє, є справжнім, це гарантує конфіденційність інформації,

оскільки він забезпечує, що обидва учасники мають правомірний доступ один до одного, а отже, неможливість підміни ідентичності або середовища.

Kerberos працює через набір централізованих центрів розподілу ключів (Key Distribution Center, KDC). Кожен центр розподілу ключів у мережі містить базу облікових даних користувачів як для користувачів, так і для служб із підтримкою Kerberos. Централізація цієї інформації полегшує навантаження на адміністраторів, оскільки тепер їм потрібно лише підтримувати єдину базу даних імен користувачів та паролів. Також наявність централізованої системи спрощує забезпечення та підтримку безпеки [32].

Для автентифікації користувачів Kerberos використовує зашифровані квитки для підтвердження ідентичності кінцевих користувачів та мережевих серверів. Зашифровані квитки генеруються централізованими центрами розподілу ключів від імені користувачів, що прагнуть пройти процедуру автентифікації в мережі. Виходячи з цього, при використанні даного протоколу паролі користувачів ніколи не передаються в відкритому вигляді, що забезпечує додатковий рівень захисту системи [32].

Kerberos не надає в пряму значенні функції авторизації та аудиту, проте використання даного протоколу надає здатність точно ідентифікувати користувачів і служби, що дозволяє розробникам та адміністраторам в подальшому реалізовувати авторизацію та аудит, а також ця інформація буде корисною для покращення системи безпеки мережі [32].

Основними об’єктами, що беруть участь у процесі Kerberos є наступними [33]:

- клієнт – діє від імені користувача та ініціює зв’язок для отримання запиту на обслуговування;
- сервер – розміщується служба, до якої користувач хоче отримати доступ;
- сервер автентифікації (Authentication Server, AS) – виконує автентифікацію клієнта, якщо автентифікація успішна, то сервер видає клієнту квиток, що має назву TGT (Ticket Granting Ticket) та запевняє інші сервери, що клієнт успішно пройшов автентифікацію;

- сервер видачі квитків (Ticket Granting Server, TGS) – це сервер додатків, що видає службові квитки як послугу;
- центр розподілу ключів (KDC) – це сервер, що складається з трьох логічно розділених частин: бази даних, серверу автентифікації та серверу видачі квитків.

Під час процесу Kerberos важливу участь беруть три унікальні секретні ключі, а саме [33]:

- клієнт/користувач – хеш паролю користувача;
- секретний ключ TGS – хеш паролю, використаний для визначення TGS;
- секретний ключ сервера – хеш пароля, що використовується для визначення сервера, що надає послугу.

У загальному процес протоколу Kerberos складається з наступних етапів [32]:

1) початковий запит автентифікації клієнта – користувач запитує квиток від сервера автентифікації, цей запит містить ідентифікатор клієнта;
2) центр розподілу ключів перевіряє облікові дані клієнта, сервер автентифікації перевіряє базу даних на наявність клієнта та серверу видачі квитків, якщо сервер автентифікації знаходить обидва значення, то генерує секретний ключ клієнта/користувача, при цьому використовуючи хеш паролю користувача, потім сервер автентифікації обчислює секретний ключ TGS і створює ключ сеансу (SK1), що зашифрований за допомогою секретного ключа клієнта/користувача, далі відбувається генерація квитка, що містить ідентифікатор клієнта, мережеву адресу клієнта, мітку часу та SK1, секретний ключ TGS шифрує квиток;
3) клієнт використовуючи секретний ключ клієнта/користувача розшифровує повідомлення та вилучає SK1 і квиток, генеруючи автентифікатор, який перевіряє TGS клієнта;
4) клієнт запитує квиток у сервера, який пропонує послугу, надсилаючи витягнутий TGT і створений автентифікатор до TGS;
5) центр розподілу ключів створює квиток для файлового сервера, сервер видачі квитків використовує секретний ключ TGS для розшифрування квитка, отриманого від клієнта і отримує значення SK1, сервер видачі квитків розшифровує автентифікатор і перевіряє чи відповідає ідентифікатору та мережевій адресі клієнта,

також перевіряє мітку часу, щоб переконатися, що квиток дійсний, якщо все виконано успішно, то центр розподілу ключів генерує ключ сеансу служби (SK2) для спільного використання клієнтом та цільовим сервером, також центр створює службовий квиток з ідентифікаторами та ключем сеансу служби, згодом цей квиток шифрується за допомогою секретного ключа сервера та клієнт отримує повідомлення, що містить службовий квиток та SK2, зашифровані за допомогою SK1;

6) клієнт розшифровує повідомлення за допомогою SK1 і витягує SK2, цей процес генерує новий автентифікатор, що містить мережеву адресу клієнта, ідентифікатор клієнта та мітку часу, зашифровані за допомогою SK2, і надсилає його разом із службовим квитком на цільовий сервер;

7) цільовий сервер використовує секретний ключ сервера для розшифрування службового квитка та вилучення SK2, далі використовує SK2 для розшифрування автентифікатора, виконуючи перевірки, щоб переконатися, що ідентифікатор клієнта та мережева адреса клієнта з автентифікатора і службового квитка збігаються і чи не закінчився термін дії службового квитка;

8) після виконання перевірок цільовий сервер надсилає клієнту повідомлення, яке підтверджує, що клієнт і сервер автентифікували один одного і тепер користувач може брати участь у безпечному сеансі.

Kerberos пройшов декілька етапів до становлення у сучасному вигляді, тому існують певні версії даного протоколу, які зазнавали змін для удосконалення. Ранні версії Kerberos (v1, v2, v3) були створені та використані всередині проєкту для цілей тестування, версії 4 та 5 зазнали покращень та були імплементовані в системи, проте зараз Kerberos версії 4 не рекомендується до використання [32].

Kerberos версії 4 був розроблений для використання в середовищі проєкту Athena, тому він не відповідав потребам загального призначення, що призвело до наступних недоліків даної версії [2]:

- залежність системи шифрування: 4 версія протоколу вимагає використання алгоритму шифрування DES, що викликало занепокоєння в зв’язку з обмеженням на експорт DES, а також сумнівами щодо стійкості DES;

- залежність від Інтернет протоколу: дана версія вимагає використання адрес Інтернет протоколу (IP), а інші типи адрес не підтримуються;
- впорядкування байтів повідомлення: у даній версії відправник повідомлення використовує власний порядок байтів і позначає повідомлення, вказуючи молодший байт у найнижчій адресі або старший байт у найнижчій адресі, цей метод працює, але не відповідає встановленим угодам;
- термін дії квитка: значення тривалості квитка у цій версії кодуються у 8-бітній кількості в одиницях 5 хвилин, таким чином максимальний термін дії становить 1280 хвилин, що може бути недостатнім для деяких систем;
- переадресація автентифікації: дана версія не дозволяє пересилати облікові дані, видані одному клієнту, на інший хост і використовувати їх іншим клієнтом, ця можливість дозволила б клієнту отримати доступ до сервера і доручити цьому серверу отримати доступ до іншого сервера від імені клієнта, наприклад, клієнт надсилає запит до сервера друку, який потім отримує доступ до клієнтського файлу з файлового сервера, використовуючи облікові дані клієнта для доступу.

Окрім обмежень протоколу Kerberos версії 4, також наявні певні технічні недоліки, які версія 5 намагається усунути, а саме [2]:

- подвійне шифрування – квитки, що надані клієнтам двічі шифруються, один раз секретним ключем цільового сервера, а потім ще раз секретним ключем, що відомий клієнту;
- режим блокового шифрування – 4 версія використовує нестандартний режим DES, відомий як ланцюг шифроблоків з поширенням (Propagating Cipher Block Chaining, PCBC), що є вразливим до атак та передбачає обмін блоками зашифрованого тексту;
- ключі сеансу – кожен квиток містить ключ сеансу, який використовується клієнтом для шифрування автентифікатора, надісланого до служби, пов’язаної з цим квитком, також ключ сеансу може згодом використовуватися клієнтом і сервером для захисту повідомлень, переданих під час цього сеансу, проте оскільки один і той самий квиток може використовуватися неодноразово для отримання обслуговування від

певного сервера, існує ризик того, що опонент відтворить повідомлення зі старого сеансу клієнта або сервера;

- атаки на пароль – обидві версії є вразливими до даного типу атаки, проте версія 5 надає механізм, відомий як попередня автентифікація, який має ускладнити атаки на пароль, але не запобігає їм.

До переваг протоколу Kerberos можна віднести наступне: висока безпека, оскільки протокол використовує шифрування та хешування для повідомлень та паролів, не передає дані у відкритому вигляді; використання централізованого сервера автентифікації, що спрощує керування процесом автентифікації та доступом до ресурсів у розподільних мережах; використання одноразових квитків зменшує ризик перехоплення автентифікаційних даних кінцевих користувачів та повторного використання; підтримка єдиного входу до ресурсів.

Недоліками даного протоколу є: складність налаштування та підтримки системи, оскільки впровадження може вимагати додаткових ресурсів та знань, що може ускладнювати використання в малих організаціях; залежність від центрального сервера, оскільки це підвищує ризик атак та навантажень на цей сервер, що ставить під загрозу роботу всієї системи; потреба у синхронізації часу, оскільки протокол використовує тривалість часу життя квитків та це потребує, щоб всі пристрої у мережі були синхронізовані з однією часовою базою даних; не є універсальним протоколом, оскільки не зможе забезпечити в сучасному середовищі гнучкість та простоту для деяких типів додатків.

1.3.5 Open Authorization

Open Authorization (OAuth) – відкритий стандартний протокол авторизації, швидко отримав визнання та може бути названий стандартом де-факто в сфері авторизації. Даний протокол дозволяє користувачам надавати стороннім додаткам доступ до захищених ресурсів користувача, що зберігаються на визначеному сервері, при цьому не розкриваючи облікові дані користувача. Протокол пройшов через дві

основні версії, причому остання версія OAuth 2.0 не має зворотної сумісності зі своєю попередницею OAuth1.0 [34].

OAuth 1.0 – це перша версія OAuth, що стала важливим кроком на шляху до більш зручної та безпечної авторизації, проте розробникам було дещо складно використовувати та налаштовувати його через вимоги до криптографічних підписів. Наступна версія, OAuth 2.0, зосереджена на спрощенні роботи для розробників, зберігаючи високі стандарти безпеки. OAuth 2.0 відмовився від криптографічних підписів на користь SSL/TLS, тим самим спростивши протокол, також представив області дії та маркери, забезпечуючи більш точний контроль доступу. Незважаючи на те, що OAuth 2.0 є вдосконаленням початкової версії, він викликає деяку критику з боку безпеки, оскільки вважається, що використання SSL/TLS робить його більш сприйнятливим до певних типів атак [35].

Оскільки було надано відмінності між версіями протоколів та остання версія є широко застосованою, тому принцип роботи та інші властивості даного протоколу будуть розглянуті на основі версії OAuth 2.0.

В роботі даного протоколу задіяні наступні ролі, що приймають важливу участь у процесі [34]:

- власник ресурсу – суб’єкт, що володіє правом надавати дозвіл іншим на доступ до ресурсів, що знаходяться в його власності, простіше кажучи, це кінцевий користувач програми, користувач-агент (браузер);
- сервер ресурсів – об’єкт, де знаходяться захищені ресурси та мають власника, має можливість відповідати на запити доступу до ресурсу за допомогою токенів доступу;
- клієнт – додаток, що може виконувати запити до сервера ресурсів від імені власника ресурсу після успішної авторизації;
- сервер авторизації – сервер, що надає токени доступу клієнтам після успішної автентифікації.

Сервер авторизації та сервер ресурсів в багатьох випадках виконує один суб’єкт [33].

Протокол використовує два типи токенів, що дозволяють клієнту отримати доступ до ресурсу [35]:

- токен доступу – використовується клієнтом для отримання доступу до даних ресурсу, сервер авторизації надає токен після належної автентифікації та авторизації, клієнт використовує даний токен доступу для запиту до сервера ресурсів;
- токен поновлення – замінює попередній токен доступу, коли його термін дії минає, при цьому не вимагає від власника ресурсу повторної автентифікації, зазвичай надається разом з токеном доступу і використовується, коли токен доступу стає недійсним.

Загальний процес протоколу виглядає наступним чином [34]:

1) клієнт запитує дозвіл у власника ресурсу на використання його захищеного ресурсу, зазвичай це відбувається через сервер авторизації, що виступає в ролі посередника;

2) клієнт отримує дозвіл наданий власником ресурсу, існує чотири різних типи авторизації, кожен з яких повинен підтримуватися як клієнтом, так і сервером авторизації;

3) клієнт звертається до сервера авторизації з метою отримання токена доступу, що використовується для доступу до захищених ресурсів, під час цього процесу клієнт надає свої облікові дані та дозвіл на автентифікацію на сервері авторизації;

4) сервер авторизації підтверджує дійсність отриманих облікових даних і надає клієнту токен доступу;

5) клієнт запитує захищені ресурси, за допомогою отриманого токена доступу, які розміщені на сервері ресурсів;

6) власник ресурсу перевіряє дійсність токена доступу і, якщо він дійсний, то успішно обслуговує запит.

Як було вказано існує чотири різних типи авторизації що призводить до чотирьох різних можливих послідовностей взаємодії повідомлень. Під час імплементації та моделювання протоколу слід врахувати кожен тип авторизації, оскільки це забезпечить можливість підтримки різних учасників протоколу та різні виконання даного протоколу. Ці типи є наступними [34]:

- код авторизації – отримується за допомогою сервера авторизації як посередника між клієнтом і власником ресурсу, власник ресурсу перенаправляється на сервер авторизації та надає свої облікові дані, перевагою даного методу є можливість автентифікації та видання токена доступу безпосередньо клієнту, що дозволяє не розкривати клієнта іншим особам, у тому числі власника ресурсу;
- неявне надання – ситуація при якій дозвіл власника ресурсу безпосередньо виражається у вигляді токена доступу, що виключає необхідність введення коду авторизації, це покращує швидкість реагування та ефективність клієнтів;
- облікові дані власника ресурсу – пароль власника ресурсу може бути використаний як авторизаційний грант на отримання токена доступу, використовується у випадках, коли між власником та клієнтом присутній високий рівень довіри;
- облікові дані клієнта – ситуація при якій для отримання токена доступу потрібно надати лише облікові дані клієнта, використовується коли обсяг повноважень, що доступні клієнту, є обмежені захищеними ресурсами, що знаходяться під контролем клієнта, або якщо доступ до захищених ресурсів вже узгоджено з сервером авторизації, у такому випадку роль власника ресурсу і роль клієнта виконує один і той самий суб’єкт.

Переваги використання протоколу OAuth полягають у його високому рівні безпеки, спрощеному користувацькому досвіді, масштабованості та гнучкості, а також в стандартизації та взаємодії. Перш за все, OAuth пропонує підвищену безпеку за рахунок використання токенів доступу, що зменшує ризик компрометації паролів користувачів. Користувачі можуть контролювати рівень доступу, що забезпечує додатковий рівень безпеки. Крім того, OAuth спрощує життя користувачів, оскільки вони можуть уникнути необхідності запам’ятовувати багато паролів і дозволити додаткам отримувати доступ до їх даних. Його масштабована та гнучка архітектура дозволяє легко впоратися з великою кількістю користувачів та адаптуватися до різноманітних сценаріїв використання. Крім того, як відкритий стандарт, OAuth широко прийнятий та використовується в галузі, що сприяє стандартизації та взаємодії між різними платформами та технологіями [36, 37, 38].

Незважаючи на переваги, протокол OAuth також має певні недоліки. Складність реалізації може стати перешкодою для розробників, оскільки протокол включає в себе різні потоки для різних сценаріїв, кожен з яких потребує навичок реалізації. Брак прямого методу автентифікації користувачів може призвести до неодноразових реалізацій і збільшити складність інтеграції з різними сервісами. Також протокол містить наступні вразливості: незахищене зберігання токенів; недостатня перевірка токенів; міжсайтова підробка запитів (Cross-Site Request Forgery, CSRF); недостатнє відкликання токенів; фішингові атаки; незахищена передача токенів; витік коду авторизації; відсутність ентропії токенів; внутрішні загрози [36, 37, 38].

1.3.6 OpenID Connect

OpenID Connect (OIDC) – це протокол делегованої автентифікації, розроблений OpenID Foundation у листопаді 2014 року, побудований на створенні базового рівня ідентифікації поверх протоколу авторизації OAuth 2.0 [39, 40].

У даному протоколі наявні наступні головні учасники: постачальник OpenID, користувач та довірена сторона. У даному контексті довірена сторона виступає додатком, сервісом, веб-сайтом, що використовує стороннього постачальника послуг для перевірки ідентичності користувачів для доступу до ресурсу. Постачальник OIDC сам є постачальником ідентифікаційних даних, що підтримує OIDC [39, 41].

Протокол використовує три токени, що призначені для виконання різних функцій. Цими токенами є: код авторизації, токен доступу та ідентифікаційний токен. Код авторизації – це тимчасовий код, що використовується для надання довіреній стороні дозволу на отримання токенів від постачальника ідентифікаційних даних для подальшого отримання токену доступу, даний код є одноразовим та має короткий термін діє, максимум 10 хвилин, задля унеможливлення витоку інформації. Токен доступу використовується для авторизації довіреної сторони для доступу до захищених ресурсів користувача, що зберігаються на сервері ідентифікаційних даних, одним з найпоширеніших токенів доступу є JWT (JSON Web Token). JWT токен – це відкритий стандарт, який використовується для безпечного обміну інформацією між

клієнтом і сервером, зазвичай це закодований JSON, що містить набір заяв і підпис, і використовується в контексті інших механізмів автентифікації, таких як OAuth і OpenID, для обміну інформацією, пов'язаною з користувачем. Ідентифікаційний токен є основним розширенням, що надає ODIC до протоколу OAuth, що робить можливим автентифікацію користувачів довіреною стороною, що реалізований як JWT токен і містить інформацію про особу кінцевого користувача зареєстрованого на ідентифікаційному сервері [39, 42].

Процес автентифікації та авторизації в загальному вигляді є наступним: користувач реєструється на постачальнику ODIC, заповнюючи реєстраційні дані і отримує обліковий запис для входу до нього; коли користувач хоче отримати доступ до послуги в довірчому ресурсі, його перенаправляють до постачальника ODIC для автентифікації; після успішної автентифікації користувач перенаправляється на довірчу сторону з підтвердженням особи та підписаним постачальником ODIC; довірча сторона перевіряє ідентифікаційні дані і надає користувачеві відповідний доступ [41].

Перевагами даного протоколу є: простота та широке застосування, оскільки протокол спрощує процес автентифікації та авторизації, а також він сумісний з багатьма сучасними програмами та платформами; забезпечує високий рівень безпеки, оскільки використовує JWT токени та базується на протоколі OAuth; завдяки його сумісності, протокол легко інтегрувати в будь-який проєкт.

Недоліками є: залежність від інших сервісів таких як ідентифікаційні провайдери; існує ризик безпеки через обмін даними між різними сервісами та системами.

1.3.7 Security Assertion Markup Language

Security Assertion Markup Language (SAML) – це протокол для обміну даними автентифікації та авторизації між доменами безпеки, розроблений Технічним комітетом служб безпеки Організації з розвитку стандартів структурованої інформації (Organization for the Advancement of Structured Information Standards,

OASIS). Протокол SAML 2.0 наразі в основному використовується для корпоративних та державних програм, що потребують безперебійну систему єдиного входу між компаніями та підприємствами. Для роботи використовує структуру розширюваної мови розмітки (eXtensible Markup Language, XML) для передачі інформації про автентифікацію, права та атрибути користувача. Є три версії даного протоколу: SAML 1.0, SAML 1.1 та SAML 2.0 [43, 44, 45].

Основними компонентами для забезпечення процесу даного протоколу є [44]:

- постачальник ідентифікаційної інформації (Identity Provider, IdP) – це системна сутність, яка створює, підтримує та керує ідентифікаційною інформацією, пропонує автентифікацію користувача як послугу;
- постачальник послуг (Service Provider, SP) – сайт до якого користувач намагається отримати доступ;
- постачальник ідентифікаційних даних профілю SSO веб-браузера – це спеціальний профіль, у якому користувач отримує доступ до постачальника послуг через веб-браузер.

Загальний процес включає наступні етапи [43]:

- запит на автентифікацію – користувач намагається отримати доступ до ресурсів, що захищені сервісом, який підтримує SAML, сервіс переадресовує користувача на постачальника ідентифікаційної інформації для проведення автентифікації;
- автентифікація та створення тверджень – перевірка ідентичності та створення твердження, які містять інформацію про користувача, таку як ім’я, роль, або додаткові атрибути;
- передача тверджень до постачальника послуг;
- авторизація – постачальник послуг перевіряє твердження, щоб підтвердити автентичність користувача, і приймає рішення про надання доступу до ресурсів на основі цих тверджень;
- доступ до ресурсі – після успішної автентифікації та авторизації постачальник послуг надає доступ до визначених ресурсів.

Протокол SAML 2.0 має певні переваги та покращення в порівнянні з своїми попередниками, а саме [43]:

- псевдоніми – є ключовою технологією, що забезпечує конфіденційність, оскільки вони перешкоджають змові між кількома провайдерами, що забезпечується завдяки псевдовипадковому ідентифікатору, що не має видимого зв’язку зі значущими ідентифікаторами (наприклад, електронною поштою або іменами облікових записів);
- управління ідентифікаторами - визначає як два провайдери можуть встановлювати та згодом керувати псевдонімами;
- шифрування;
- профілі атрибутів;
- управління сесіями;
- розширення підтримуваних пристроїв.

До переваг використання SAML можна віднести наступне: безпека, оскільки забезпечує шифрування та інші механізми безпеки для користувачів та ресурсів; підтримка єдиного входу.

Недоліками даного протоколу є: складність налаштування; менш ефективний при масштабуванні в великих організаціях.

1.4 Проблеми та виклики автентифікації та авторизації

Автентифікація та авторизація є критичними складовими для забезпечення безпеки в інформаційних системах, тому важливим є безперервне дослідження з метою досягнення вищого рівня безпеки. Незважаючи на постійні удосконалення методів, протоколів та технологій автентифікації та авторизації, досі залишаються певні прогалини в безпеці систем. Процес автентифікації та авторизації має наступні відомі проблеми та вразливості [46, 47, 48]:

- слабкі політики паролів;
- покладання на однофакторну автентифікацію;
- надпривілейовані облікові записи;

- відсутність централізованого керування ідентифікацією;
- некоректний облік неактивних облікових записів;
- недостатнє управління токенами.

Поширеними проблемами автентифікації є: парольна слабкість; фішингові атаки; перехоплення сесій; атаки типу брутфорс; обробка помилок автентифікації.

Поширені проблеми авторизації: перевищення привілеїв; проблеми з сегрегацією обов'язків; використання застарілих або недостатніх механізмів авторизації.

Висновки за розділом 1

У першому розділі дипломної роботи було подано основні визначення та принципи автентифікації та авторизації, а саме: автентифікація користувача, що складається з двох етапів, а саме ідентифікації та верифікації; авторизації; послідовність процесів автентифікації та авторизації. Також було подано основні методи автентифікації, що можуть використовуватися як самостійно так і в комбінації. Цими методами є: автентифікація на основі того, що відомо користувачу; автентифікація на основі того чим володіє користувач; автентифікація на основі унікальних біологічних характеристиках людини; багатофакторна автентифікація; автентифікація на основі сертифікатів. Щодо методів авторизації, то було розглянуто: контроль доступу на основі ролей; контроль доступу на основі атрибутів; дискреційний контроль доступу; мандатний контроль доступу; контроль доступу на основі політик; управління привілейованим доступом.

Протоколи автентифікації та авторизації відіграють важливу роль в процесі, тому в кваліфікаційній роботі було розглянуто наступні протоколи автентифікації та авторизації: LDAP, RADIUS, Kerberos, OAuth, OpenID Connect, SAML, HTTP Basic Authentication.

Також було надано перелік найбільш поширених проблем автентифікації та авторизації.

РОЗДІЛ 2
ТЕХНОЛОГІЇ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ. МОЖЛИВОСТІ ТЕХНОЛОГІЇ ASP.NET CORE IDENTITY

2.1 Технологій автентифікації та авторизації

Сучасні технологій автентифікації та авторизації пропонують широкий функціонал для досягнення конфіденційності та цілісності інформації. Правильний підбір до потреб організації та використання переваг технологій дозволить створити безпечне середовище для функціонування вебдодатків.

Технології повинні забезпечувати безпеку, конфіденційність та цілісність даних, а також гарантувати, що лише ідентифіковані користувачі мають доступ до системи та обмежений доступ до ресурсів всередині системи. Такими технологіями є: ASP.NET Core Identity, Firebase, AWS Cognito, AWS Identity and Access Management, Auth0, Azure Active Directory, Google Cloud Identity.

2.1.1 ASP.NET Core Identity

ASP.NET Core Identity – це технологія, що підтримує функції входу в інтерфейс користувача, а також керує користувачами, паролями, даними профілю, ролями, претензіями, маркерами, підтвердженням електронної пошти тощо. Користувачі можуть створити обліковий запис за допомогою облікових даних для входу або вони можуть пройти реєстрацію використовуючи зовнішні постачальнику входу, до яких належать Facebook, Google, Microsoft [49].

Фреймворк підтримує різні протоколи для автентифікації, такі як cookies, JWT, OAuth. Надає можливості для визначення ролей користувачів та надання їм відповідних прав доступу до функцій та ресурсів додатку [49].

До переваг даної технології можна віднести наступне: підтримка зовнішніх постачальників послуг, ця можливість спрощує користувачам процес реєстрації та

автентифікації, оскільки за допомогою вже наявного акаунту вони можуть ідентифікуватися до запитуваного ресурсу; технологія використовує та зберігає паролі в хешованому вигляді, що підвищує рівень безпеки системи [50].

Проте в технології є певні недоліки, а саме: залежність від платформи ASP.NET Core; вимоги до інфраструктури та складність налаштування, яка потребує певних навичок з боку розробника [50].

Дана технологія найкраще підійде для інтеграції з платформою ASP.NET Core та вимагають гнучкої системи ідентифікації та авторизації з високим рівнем безпеки [50].

2.1.2 Firebase

Технологія Firebase являє собою рішення бекенд як сервіс (Backend as a Service, BaaS), що надає розробникам набір інструментів та скорочує час і важкість конфігурації та налаштування. Для автентифікації користувачів використовується JSON веб-токен. Технологія підтримує такі механізми автентифікації як: комбінації електронної пошти/паролю та електронної пошти/посилання, дозволяє автентифікувати номер телефону та підтримує вхід із соціальних мереж. Firebase також надає можливості авторизації користувачів, а саме захищає та контролює доступ до кожної серверної служби використовуючи правила безпеки, що дозволяють використовувати мову виразів та працюють, зіставляючи шаблон із шляхами бази даних або сегментів, а потім застосовують спеціальні умови, щоб дозволити доступ для читання та запису до даних у цих шляхах [51].

До переваг даної технології можна віднести: простоту та швидкість інтеграції, що допомагає малим організаціям швидко та ефективно впровадити рішення автентифікації та авторизації, що не вимагає затрат зусиль на налаштування та впровадження; різноманітність підтримуваних методів автентифікації – це дозволить забезпечити безпеку в тих випадках де це необхідно, а також надає зручність користувачам; гнучкість налаштування – дозволяє налаштувати функції під потреби системи [50].

Недоліки даної технології можна виділити наступні: обмежені можливості налаштування прав доступу; можливість обмеження функціоналу під час інтеграції з іншими платформами [50].

Це рішення найкраще пасує для проєктів, які активно використовують інші сервіси Google та потребують простого та швидкого рішення для автентифікації та авторизації [50].

2.1.3 AWS Cognito

AWS Cognito – це рішення, розроблене Amazon Web Services (AWS), яке забезпечує безпечний процес реєстрації та входу в обліковий запис користувача. Сервіс забезпечує синхронізацію даних між пристроями користувачів, що дозволяє уникнути збоїв у роботі програми незалежно від використовуваного пристрою. AWS Cognito включає такі функції: реєстрацію, вхід та керування обліковим записом. Технологія підтримує використання сторонніх постачальників послуг, до них відносяться соціальні мережі, відомі провайдери, а також постачальники ідентифікаційних даних корпоративного класу. AWS Cognito використовує стандартні сервіси автентифікації та авторизації, а саме Oauth 2.0, SAML 2.0 та OpenID Connect [52].

До переваг даного сервісу можна віднести: масштабованість - дозволяє здатний обробляти великий обсяг автентифікаційних та авторизаційних запитів, що робить його ідеальним рішенням для проєктів будь-якого масштабу; гнучкість, оскільки пропонує різні налаштування та опції конфігурації, які дозволяють налаштовувати автентифікаційні потоки та політики безпеки відповідно до потреб і вимог; швидкість розробки, оскільки завдяки готовому набору функцій і інструментів технологія значно спрощує процес розробки автентифікаційних та авторизаційних функцій в додатку; підтримка різних методів автентифікації та шифрування [50].

До недоліків даної технологія можна віднести складність налаштування, вартість рішення; залежність від AWS [50].

Найкраще дана технологія підходить для організацій, що використовують інфраструктуру AWS та потребують розширеного функціоналу для управління ідентичністю та доступом [50].

2.1.4 AWS Identity and Access Management

AWS Identity and Access Management (IAM) – дозволяє контролювати доступ до сервісів і ресурсів AWS, надає можливість створювати користувачів і групи користувачів, керувати ними, а також дозволяти або обмежувати права доступу користувачів і їх груп до певного ресурсу [52].

Перевагами даного рішення є: гнучкість управління доступом, оскільки надає широкі можливості для налаштування прав доступу до різних ресурсів в обліковому записі, надає можливість створювати користувачів, групи користувачів та ролі з різними рівнями доступу, що дозволяє точно контролювати, хто має доступ до яких ресурсів і яким чином; інтеграція з іншими AWS-сервісами; можливість аудиту, оскільки забезпечує можливість ведення журналу подій, який дозволяє відстежувати всі дії користувачів.

Недоліками даної технології є: складність конфігурації, необхідність розуміння концепцій AWS.

Як і технологія AWS Cognito, AWS IAM найкраще підходить для організації, що вже використовують AWS та вимагають розширеного функціоналу для управління.

2.1.5 Auth0

Технологія Auth0 надає інструменти для додавання потоків автентифікації та авторизації до програм, забезпечує гнучке керування ідентифікацією та параметрами автентифікації. Технологія містить такі функції: єдиний вхід (Single Sign-On, SSO), багатофакторна автентифікація, автентифікація без використання пароля (використання інших методів як біометрія, тощо), функції керування користувачам.

Auth0 використовує протокол OpenID Connect і систему авторизації OAuth 2.0 [53, 54].

До переваг даного рішення можна віднести наступне: простота інтеграції, оскільки надає простий та інтуїтивно зрозумілий інтерфейс для інтеграції з різноманітними додатками та платформам; масштабованість – система здатна відповісти на зростаючу кількість користувачів та об’єм трафіку без втрати продуктивності; багатофакторна автентифікація, що підвищує рівень безпеки; захист від атак; готові рішення та аналітика, оскільки пропонує готові рішення для різних сценаріїв автентифікації, такі як управління користувачами, соціальна автентифікація та багато іншого [50].

Недоліками даної технології є: вартість; залежність від стороннього сервісу; необхідність налаштування, що може вимагати значних зусиль та часу для правильного конфігурування для забезпечення потреб організації [50].

Рішення найкраще підходить для проєктів, які вимагають різноманітних методів автентифікації та авторизації, включаючи соціальні мережі [50].

2.1.6 Azure Active Directory

Azure Active Directory надає хмарний корпоративний каталог і службу керування ідентифікацією. Технологія представляє функції, що надають користувачам безперебійний доступ до всіх типів внутрішніх та зовнішніх ресурсів, використовуючи традиційні методи автентифікації користувачів, а саме за допомогою імені користувача та пароля. Також надає можливість керування ролями та дозволами, щоб забезпечити користувачам доступ до необхідних ресурсів. Окрім базових методів автентифікації та авторизації надає можливість використання функцій багатофакторної автентифікації та SSO. Використовує протоколи OAuth 2.0 та OpenID Connect [55].

До переваг можна віднести наступне: інтеграція з іншими послугами Azure; масштабованість; гнучкість управління доступом, оскільки надає широкі можливості

для налаштування прав доступу, ролей та політик, що дозволяє точно налаштувати управління доступом до ресурсів та даних [50].

Недоліками технології є: залежність від інфраструктури Microsoft, оскільки підтримується та обслуговується інфраструктурою Microsoft, що може створювати обмеження або проблеми у разі відмови чи недоступності сервісів Microsoft; складність налаштування, може бути викликом для організацій з обмеженими технічними ресурсами або навичками; вартість [50].

Ця технологія найкраще підходить для великих підприємств, які вимагають високого рівня безпеки та гнучкості в управлінні ідентичністю та доступом [50].

2.1.7 Google Cloud Identity

Google Cloud Identity – це рішення ідентифікації як сервіс (Identity as a Service, IDaaS), яка надає централізоване керування ідентифікаційними даними та забезпечує безпечну автентифікацію та авторизацію для програм і пристроїв. Вона підтримує використання токенів, SAML та OpenID Connect, а також надає можливості багатофакторної автентифікації, SSO та налаштування доступів до ресурсів за допомогою Cloud Identity Groups API [56, 57].

До переваг даного рішення можна віднести наступне: простота управління, оскільки надає простий та інтуїтивно зрозумілий інтерфейс для управління доступом та ідентифікацією користувачів, що дозволяє швидко і легко налаштовувати права доступу та контролювати безпеку віддалених ресурсів; один вхід для всіх сервісів Google; інтеграція з екосистемою Google, оскільки легко інтегрується з іншими продуктами та сервісами в екосистемі Google, що робить його ідеальним вибором для компаній, які вже використовують Google Workspace або інші рішення Google [50].

Недоліками даної технології є: залежність від інфраструктури Google, оскільки передбачає повну залежність від інфраструктури та сервісів Google, що може створювати проблеми у разі відмови сервісів Google або обмеження доступу до них; складність налаштування та інтеграції з сторонніми сервісами [50].

Найкраще підходить для проєктів, які використовують інфраструктуру Google та потребують інтегрованого рішення для автентифікації та авторизації [50].

2.2 Основні характеристики, архітектура та компоненти ASP.NET Core Identity

ASP.NET Core Identity – це фреймворк, що підтримує функцію входу в інтерфейс користувача, а також керує користувачами, паролями, обліковими даними, ролями, токенами та іншим функціоналом, що забезпечує безпечну автентифікацію та авторизацію в вебдодатках [49].

Основними характеристиками даної технології є наступні [58]:

- управління користувачами – надає можливість управляти створенням, видаленням та змінами в обліковому записі користувачів;
- управління ролями – підтримує авторизацію на основі ролей, що в свою чергу надає можливість для групування користувачів для подальшого забезпечення доступу;
- автентифікація користувачів – надає функціональні можливості для реалізації автентифікації користувачів;
- авторизація користувачів – після успішної автентифікації користувачів, сервіс може надавати доступ до визначених ресурсів на основі ролей або вимог;
- автентифікація на основі тверджень – дозволяє призначати користувачам твердження, претензії, що згодом можна використовувати для ідентифікації користувача та контролю доступу;
- підтримка зовнішніх постачальників входу – забезпечує інтеграцію з сторонніми автентифікаційними постачальниками (такими як Google, Microsoft, Facebook);
- двофакторна автентифікація – підтримує даний метод автентифікації, що забезпечує вищий рівень захисту;

- зберігання даних – за замовчування використовує Entity Framework Core для зберігання даних користувачів, проте також можна налаштувати зберігання інших типів даних;
- безпека – підтримує наступний функціонал для підвищення рівню безпеки: підтвердження облікового запису, відновлення паролю, запобігання поширеним атакам.

Компонентами взаємодії ASP.NET Core Identity є [58]:

- менеджер користувача (UserManager) – це клас, що є відповідальним за управління користувачами в додатку, надає функціонал для створення, видалення, модифікації облікових даних користувача в базі даних;
- менеджер ролей (RoleManager) – це клас, що є відповідальним за управління ролями в додатку, надає функціонал для створення, видалення, модифікації ролей та присвоєння їх користувачам;
- менеджер входу (SignInManager) – це клас, що є відповідальним за управління процесами входу та виходу користувачів, надає функціонал для двофакторної автентифікації та автентифікації за допомогою зовнішніх постачальників входу;
- ідентифікація користувача (IdentityUser) – це клас користувача, що включає загальні його властивості а саме пошта, номер телефону, ідентифікатор користувача тощо, надає можливість для розширення властивостей користувача;
- ідентифікація ролей (IdentityRole) – це клас ролей, що зберігає інформацію про створену роль, а саме назву та ідентифікатор, також може бути розширений додатковими атрибутами, наприклад, опис ролі;
- параметри ідентифікації – забезпечує параметри налаштувань для ідентифікації, серед них: політики паролю, сторінки забороненого доступу за замовчуванням тощо;
- перевірка користувача та паролів – забезпечує перевірку інформації користувачів та паролів;

- відправник електронної пошти та СМС (EmailSender та SmsSender) – цей сервіс використовується для підтвердження електронної пошти та двофакторної ідентифікації;
- cookies – технологія використовує автентифікацію на основі файлів cookie за замовчуванням;
- автентифікація на основі тверджень – підтримує функціонал створення та управління твердженнями, претензіями, користувачів для подальшої авторизації;
- зовнішні постачальнику входу;
- IdentityDbContext – це контекстний клас ідентифікації, що відповідає за взаємодію з базою даних ASP.NET Core Identity, що містить ідентифікаційні дані користувачів;
- ClaimsPrinciple – в ASP.NET Core Identity ідентифікатор автентифікованого користувача представлено як об’єкт ClaimsPrincipal, твердженнями є пари ключ-значення, що представляють інформацію про користувача, наприклад ідентифікатор користувача, адресу електронної пошти, ім’я, роль або інші дані, пов’язані з ідентифікацією користувача в програмі.

Загалом компоненти ASP.NEY Core Identity можна поділити на такі умовні групи: компоненти ідентифікації, що включають UserManager, RoleManager, SignInManager; моделі даних, що включають IdentityUser та IdentityRole; сервіси, що включають параметри ідентифікації, перевірка користувача та паролів, EmailSender та SmsSender; автентифікація та авторизація, що складається з cookies; автентифікації на основі тверджень; зовнішніх постачальників послуг.

Архітектура ASP.NET Core Identity складається з таких основних компонентів, що взаємодіють між собою наступним чином [58]:

- реєстрація та вхід: запити на реєстрацію та вхід обробляються контролерами ASP.NET Core; для реєстрації користувача використовується клас UserManager та зберігає дані в Identity Database; для логіну використовується клас SignInManager; після успішної автентифікації створюється файл cookie з інформацією про автентифікованого користувача;

- управління ролями та твердженнями: після успішної автентифікації, наявні ролі та твердження користувача, що забезпечують подальший доступ до обмеженого ресурсу, витягуються з бази даних за допомогою класу UserManager та надалі зберігаються в об’єкті IdentityUser;
- створення файлів cookie та квитків автентифікації: квиток автентифікації, що містить інформацію про ідентичність користувача включаючи ролі та твердження, надається після автентифікації користувача, надалі зберігається в зашифрованому вигляді в файлі cookie;
- запити автентифікації: запити користувача надсилаються браузером у вигляді файлів cookiедля автентифікації, що для доступу розшифровується і перетворюються в об’єкт ClaimsPrincipal;
- авторизація: під час запиту на отримання доступу до захищеного ресурсу, проміжне програмне забезпечення авторизації перевіряє об’єкт ClaimsPrincipal; авторизація на основі визначеного методу; перевірка тверджень користувача на те чи відповідають вони запитуваному ресурсу;
- управління та налаштування користувачів: впродовж життєвого циклу вебдодатку класи UserManager та RoleManager керують ідентифікаційними даними користувача, ролями та твердженнями;
- вихід з системи: використовується клас SignInManager;
- зберігання та налаштування даних: за замовчуванням технологія зберігає дані в базі даних SQL за допомогою Entity Framework Core, проте також можна налаштувати для інших баз даних.

2.3 Використання ASP.NET Core Identity для автентифікації та авторизації

ASP.NET Core Identity надає наступні можливості для налаштування автентифікації та авторизації у вебдодатках на платформі ASP.NET Core: управління обліковими записами користувачів; реалізація та підтримка автентифікації та авторизації; підтримка двофакторної автентифікації, підтримка зовнішніх постачальників послуг; можливість підтвердження даних користувача та паролю;

шифрування паролів; можливість налаштування політик паролю, доступу та інших; управління ролями [49, 58].

Використання ASP.NET Core Identity буде доречним при таких потребах організації [58]:

- автентифікація та авторизація користувачів – пропонує готове рішення для впровадження процесів автентифікації та авторизації в додатку, пропонуючи готові рішення, що гнучко налаштовуються для забезпечення потреб ресурсу;
- контроль доступу на основі ролей – дозволяє керувати ролями та присвоювати їх користувачам для доступу до захищених ресурсів;
- безпека – рішення пропонує хешування паролів, блокування акаунтів користувачів, захист користувацьких даних, запобігання поширеним типам вразливостей;
- інтеграція з ASP.NET Core;
- вимоги до даних користувача – технологія дозволяє розширити набір ідентифікаційних даних користувача, що будуть зберігатися в базі даних, додаючи додаткові властивості для профілю користувача, щоб забезпечити потреби додатку.

2.4 Можливості використання ASP.NET Core Identity для проєктування моделі автентифікації та авторизації у вебдодатках

Модель автентифікації та авторизації буде розроблена для найпростіших вебдодатків, наприклад, додатки призначені для допомоги ведення здорового способу життя, додатки з рекомендаціями догляду за хатніми рослинами та інші. Функціональність таких додатків полягає в статтях та рекомендаціях щодо певної тематики. Для таких додатків є характерним використання ролей, що визначають рівень доступу та можливостей користувачів. Детальна функціональність буде розглянута на прикладі додатку для допомоги ведення здорового способу життя.

Такий додаток має на меті допомогти користувачам краще зрозуміти свої потреби та покращити стан свого здоров’я. Функціональність додатку полягає в

статтях та рекомендаціях щодо харчування та тренування. Для доступу до сайту є три типи ролей: стандартний користувач, преміум користувач та адміністратор.

Стандартний користувач має доступ до обмеженої кількості порад, статей та рекомендацій. Для преміум користувача існує розширений функціонал, який полягає в додаткових статтях та розробленого плану тренування та харчування. Адміністратор має доступ до всього функціоналу сайту та до можливості побачити та відповісти на питання користувачів.

Для додаткової перевірки ідентичності адміністратора необхідне використання багатофакторної автентифікації за допомогою OTP токенів.

Для підтримання моделі автентифікації та авторизації у поданому вебдодатку будуть використовуватися наступні можливості ASP.NET Core Identity [49, 58]:

- клас IdentityUser: цей клас представляє користувача в системі, можна розширити його або створити власний клас користувача, який успадковує IdentityUser, для додавання додаткових властивостей користувача;
- клас IdentityRole: цей клас представляє роль в системі, можна використовувати його для визначення ролей, таких як «стандартний користувач», «преміум користувач» та «адміністратор»;
- клас UserManager: цей клас надає методи для управління користувачами, такі як створення, оновлення, видалення та отримання користувачів, може використовуватися для створення нових користувачів та налаштування їх ролей та дозволів;
- клас RoleManager: цей клас надає методи для управління ролями, такі як створення, оновлення, видалення та отримання ролей;
- атрибути авторизації: ASP.NET Core Identity надає різні атрибути авторизації, такі як [Authorize], які можна використовувати для обмеження доступу до контролерів та дій на основі ролей та дозволів;
- політики авторизації: може використовувати політики авторизації для визначення складних правил доступу до ресурсів на основі ролей, дозволів та інших умов;

- багатофакторна автентифікація: ASP.NET Core Identity підтримує багатофакторну автентифікацію, включаючи використання OTP токенів;
- перевірка доступу: може використовувати методи, такі як UserManager.IsInRoleAsync() та UserManager.HasClaimAsync(), для перевірки ролей та дозволів користувача та обмеження доступу до функціональності на основі цих перевірок.

Висновки за розділом 2

Важливу роль відіграють технології, що допомагають реалізувати процес автентифікації та авторизації в системі. Завдяки цим технологіям розробники та власники ресурсу, можуть швидко інтегрувати систему безпеки в свої мережі та забезпечити конфіденційність даних ресурсу та користувачів. Було розглянуто наступні технології: ASP.NET Core Identity, Firebase, AWS Cognito, AWS Identity and Access Management, Auth0, Azure Active Directory, Google Cloud Identity. Правильний підбір до потреб організації та використання переваг технологій дозволить створити безпечне середовище для функціонування вебдодатків, тому було представлено для яких систем та організацій найкраще підходять дані рішення. Вибір технології для автентифікації та авторизації повинен базуватися на специфічних потребах та вимогах організації або проекту, а також на технічних можливостях та обмеженнях кожної конкретної технології.

Технологія ASP.NET Core Identity найкраще підходить для інтеграції з вебдодатками на платформі ASP.NET Core, оскільки є повністю сумісна з даним рішенням. Окрім цього фреймворк надає широкий функціонал для забезпечення процесу автентифікації та авторизації у вебдодатках. Цими функціональними можливостями є: управління користувачами та ролями; автентифікація та авторизація користувачів; автентифікація на основі тверджень; підтримка зовнішніх постачальників входу; двофакторна автентифікація; зберігання даних; контроль доступу на основі ролей; безпека.

Компоненти ASP.NEY Core Identity поділені на такі умовні групи, що взаємодіють між собою для управління та налаштування процесу автентифікації та авторизації: компоненти ідентифікації, що включають UserManager, RoleManager, SignInManager; моделі даних, що включають IdentityUser та IdentityRole; сервіси, що включають параметри ідентифікації, перевірка користувача та паролів, EmailSender та SmsSender; автентифікація та авторизація, що складається з cookies; автентифікації на основі тверджень; зовнішніх постачальників послуг.

Для проєктування моделі автентифікації та авторизації у вебдодатках можуть бути використані наступні складові ASP.NET Core Identity: клас IdentityUser; клас IdentityRole; клас UserManager; клас RoleManager; атрибути авторизації; політики авторизації; багатофакторна автентифікація: ASP.NET Core Identity підтримує багатофакторну автентифікацію, включаючи використання OTP токенів; перевірка доступу може використовувати методи, такі як UserManager.IsInRoleAsync() та UserManager.HasClaimAsync; для налаштування реєстрації пробного періоду можна використовувати авторизації на основі тверджень (Claim-based authorization).

Модель автентифікації та авторизації буде розроблена завдяки можливостям технології ASP.NET Core Identity та для найпростіших типів вебдодатків, що наповнені статтями та рекомендаціями відповідно до тематики додатку та характеризуються наявністю наступних ролей: стандартний користувач, преміум користувач та адміністратор.

РОЗДІЛ 3
МОДЕЛЬ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ В ВЕБДОДАТКАХ ЗА ДОПОМОГОЮ ASP.NET CORE IDENTITY

3.1 Проєктування моделі автентифікації та авторизації

Модель автентифікації та авторизації повинна задовольняти наступні умови:

- перевірка ідентичності користувача використовуючи логін та пароль;
- захист ресурсу за допомогою визначених ролей користувачів;
- перевірка прав доступу до ресурсу;
- додатковий захист за допомогою двофакторної автентифікації.

Модель автентифікації та авторизації в вебдодатках полягає в перевірці ідентичності користувача та доступу до відповідних ресурсів. Як було зазначено в попередньому розділі, існує три ролі доступу: стандартний користувач, преміум користувач та адміністратор. Варто зазначити, що користувач з доступом адміністратор, повинен обов’язково використовувати двофакторну автентифікацію використовуючи OTP, оскільки цей тип акаунт в вебдодатках зазвичай має більше прав доступу, серед яких можуть бути конфіденційні дані користувачів та додатку.

Використання запропонованої моделі автентифікації та авторизації в найпростіших вебдодатках забезпечить наступні переваги:

- безпека – використання доступу на основі ролей та двофакторної автентифікації забезпечить покращений захист ресурсів та даних користувачів;
- спрощення процесу адміністрування та розробки – модель дозволяє краще розуміти процес автентифікації та авторизації, також у разі виникнення проблем допоможе виявити та виправити помилки в реалізації;
- підвищення рівня довіри користувачів: впровадження моделі автентифікації та авторизації у вебдодатку сприятиме більшій довірі користувачів, оскільки вони будуть впевнені в тому, що їх дані захищені.

Згідно з умовами вище, було удосконалено модель автентифікації та авторизації, що представлена на рис. 3.1 у вигляді блок схеми.

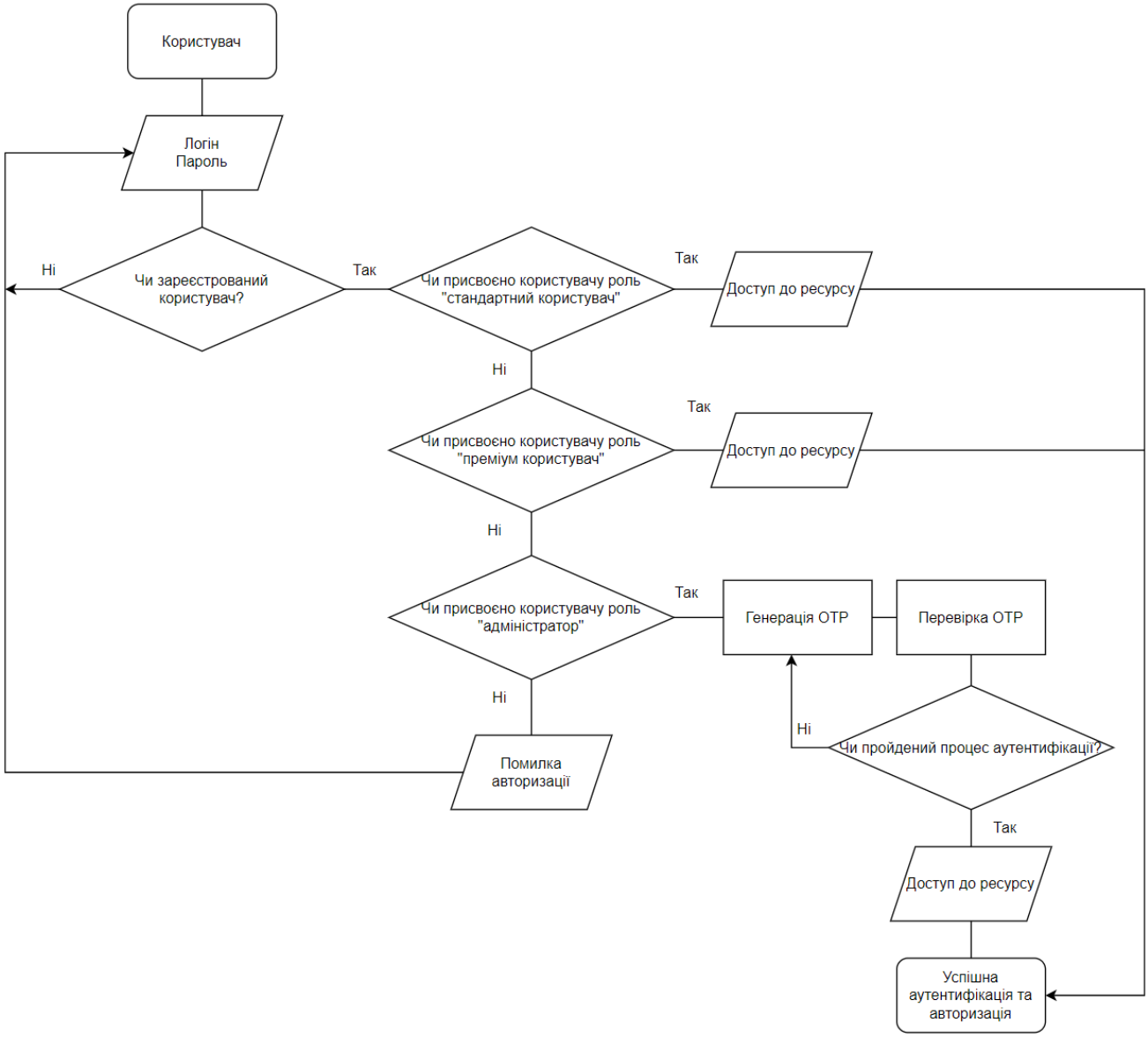

Рисунок 3.1 – Модель автентифікації та авторизації у вебдодатку у вигляді блок-схеми

Також функціональні можливості технології ASP.NET Core Identity дозволяють кожному зареєстрованому та авторизованому користувачу налаштувати двофакторну автентифікацію за допомогою додатків-автентифікаторів, серед них Google Authenticator та Microsoft Authenticator. Ці додатки використовують Time-based One-time Password Algorithm (TOTP) для підтвердження ідентичності користувача. Використання такого методу двофакторної автентифікації має ряд переваг, а саме: безпека, зручність для користувача, швидка реалізація завдяки можливостям

ASP.NET Core Identity. Модель у вигляді блок-схеми, що показує процес автентифікації та авторизації користувача при ввімкненій двофакторній автентифікації за допомогою додатку автентифікації показано на рис. 3.2.

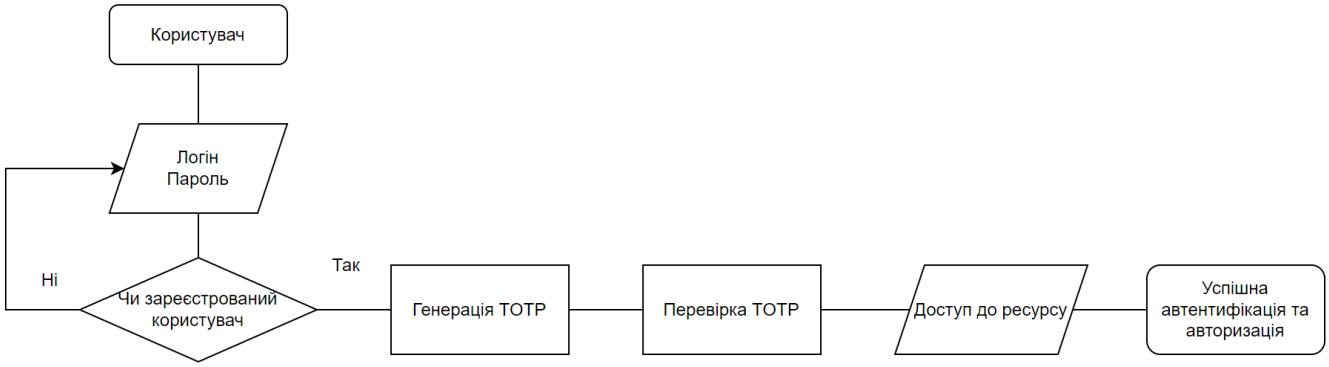

Рисунок 3.2 – Модель автентифікації та авторизації у вигляді блок-схеми при використанні додатка-автентифікатора

ASP.NET Core Identity використовує значення за замовчуванням для налаштування файлів cookie, тому модель автентифікації та авторизації, використовуючи ці файли показана на рис. 3.3.

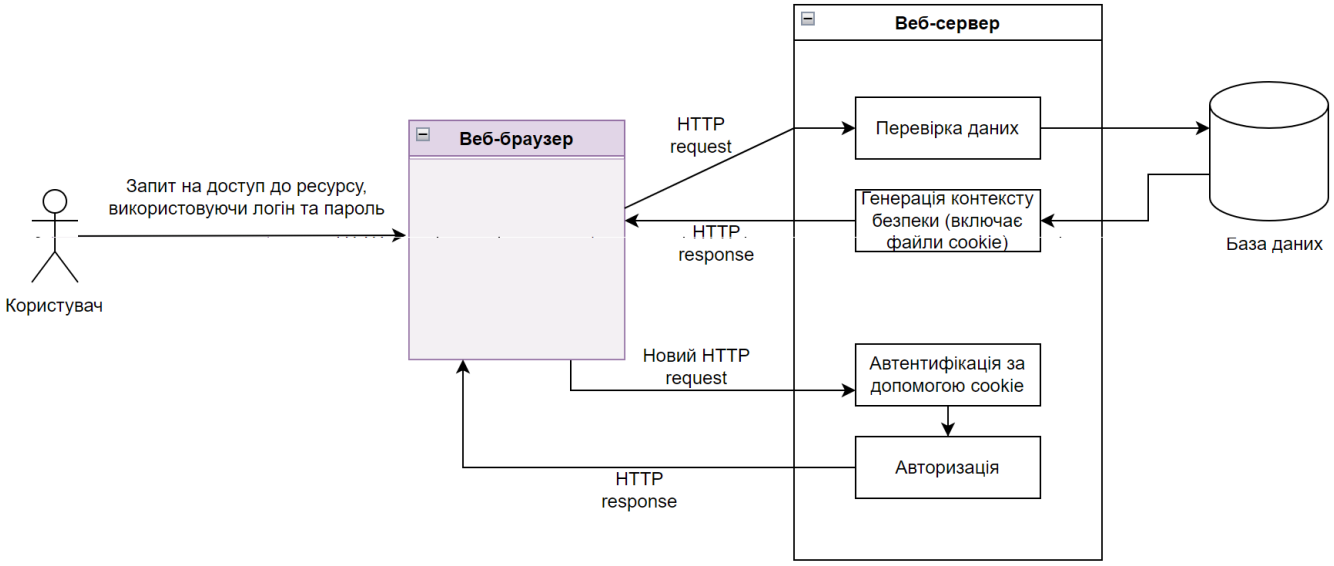

Рисунок 3.3 – Модель автентифікації та авторизації використовуючи файли cookie

Завдяки тому, що технологія ASP.NET Core Identity підтримує налаштування файлів cookie, користувачу не потрібно щоразу вводити свої дані для перевірки при кожному запиті ресурсу на той же веб-сервер. Це досягається тим, що веб-сервер формує контекст безпеки, включаючи зашифровані дані користувача, згідно інформації з бази даних.

3.2 Реалізація моделі автентифікації та авторизації

Розроблена модель авторизації та автентифікації реалізована завдяки ASP.NET Core Identity та середовища Visual Studio 2022. Для реалізації було створено новий проєкт з назвою Auth_Model використовуючи шаблон ASP.NET Core Web App (Razor Pages). Завдяки Razor Pages реалізований зовнішній вигляд моделі, оскільки призначений для спрощення реалізації типових шаблонів, що використовуються у браузерах, при створенні програми. Проєкт використовує фреймворк .NET 8.0. Також для реалізації моделі було створено базу даних SQL, що була підключена до створеного проєкту.

До проєкту було імплементовано пакети, що показані на рис. 3.3 та будуть використовуватися для подальшої реалізації.

AuthModel.csproj

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

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="8.0.4" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="8.0.4" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="8.0.4" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.4" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.4">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="8.0.2" />
    <PackageReference Include="SendGrid" Version="9.29.3" />
    <PackageReference Include="SendGrid.Extensions.DependencyInjection" Version="1.0.1" />
  </ItemGroup>

</Project>
```

Рисунок 3.3 – Інстальовані пакети

Пакет Microsoft.AspNetCore.Identity.EntityFrameworkCore розширює функціонал ASP.NET Core Identity за допомогою Entity Framework Core, дозволяючи зберігати та керувати даними користувачів такими як імена, паролі, ідентифікатори, ролі, тощо.

Пакет Microsoft.AspNetCore.Identity.UI містить готові компоненти інтерфейсу користувача, які в подальшому можна відредагувати під власні потреби.

Пакет Microsoft.EntityFrameworkCore.SqlServer є постачальником Entity Framework Core для бази даних Microsoft SQL Server, що дозволяє взаємодіяти з базою даних SQL Server за допомогою Entity Framework Core.

Пакет Microsoft.EntityFrameworkCore.Tools містить інструменти командного рядка Entity Framework Core, що будуть використовуватися для міграції даних та оновлення бази даних.

Пакет SendGrid надає бібліотеку для взаємодії з сервісом електронної пошти SendGrid, що буде використовуватися для підтвердження електронної пошти користувача та двофакторної автентифікації за допомогою OTP та електронної пошти.

Пакет SendGrid.Extensions.DependencyInjection допомагає налаштувати залежності SendGrid в проєкті, також буде використовуватися для підтвердження електронної пошти та двофакторної автентифікації за допомогою OTP.

Було створено клас ApplicationDbContext, що розширює клас контексту бази даних (DbContext) структури Entity Framework Core. Програмний код цього класу та подальших файлів налаштувань буде знаходитись в Додатку В. В файлі Program.cs було додано створений ApplicationDbContext для подальшого його налаштування та взаємодії з підключеною базою даних.

Після підключення бази даних та розширення класу було виконано початкову міграцію, використовуючи IdentityDbContext було підключено таблиці, що необхідні для автентифікації та авторизації. Ці таблиці можна налаштовувати під потреби вебдодатку. Тому було додано такі поля таблиці яг: First Name, Last Name. Це було реалізовано завдяки класу розширення ApplicationUser, що доповнює клас

IdentityUser. Після міграції було оновлено базу даних, на рис. 3.4 показано таблиці, які містить база даних.

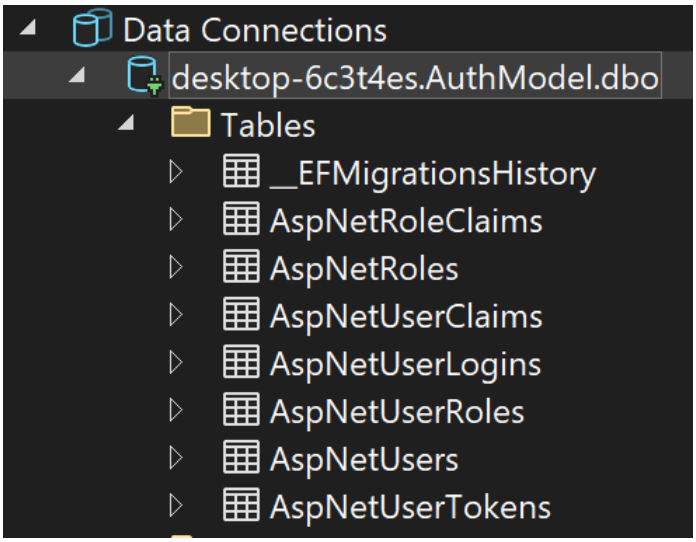

Рисунок 3.4 – Таблиці бази даних

Для реалізації авторизації на основі ролей було оновлено таблицю AspNetRoles за допомогою налаштування класу ApplicationDbContext. Було ініціалізовано три ролі: User, Admin, Premium User. Оновлена таблиця показана на рис. 3.5.

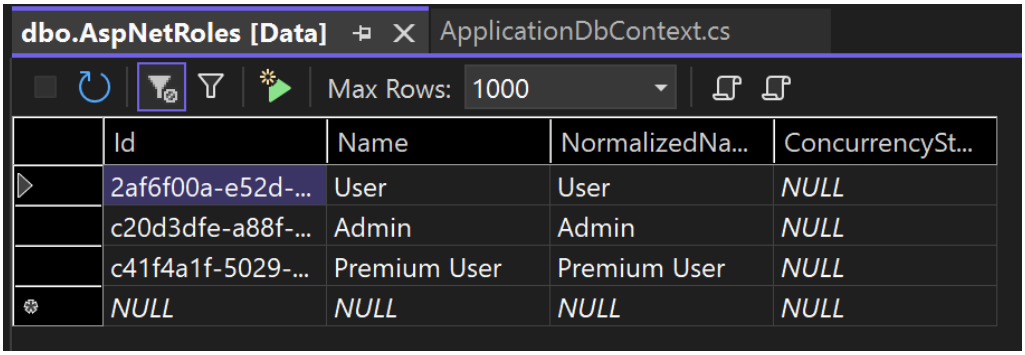

Рисунок 3.5 – Таблиця AspNetRoles

Було додано Identity Pages для подальшого налаштування вигляду та функціоналу моделі автентифікації та авторизації.

За допомогою файлів _LoginPartial.cshtml та _Loyout.cshtnl було змінено навігаційну панель. Результат налаштування інтерфейсу показано на рис. 3.6.

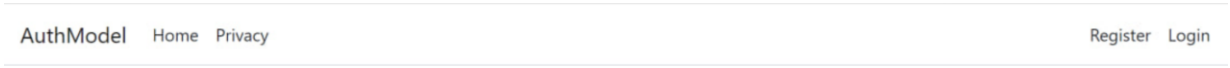

Рисунок 3.6 – Результат налаштування навігації

Було налаштовано сторінку реєстрації користувача за допомогою файлу Register.cshtml. Було додано поля реєстрації First Name та Last Name. Інтерфейс сторінки реєстрації показано на рис. 3.7. Щоб підключити додаткові додані властивості необхідно змінити налаштування файлу Register.cshtml.cs, що відповідає за функціональність вебдодатку. Для створення нового користувача було використано нову модель створення об’єкту – ApplicationUser().

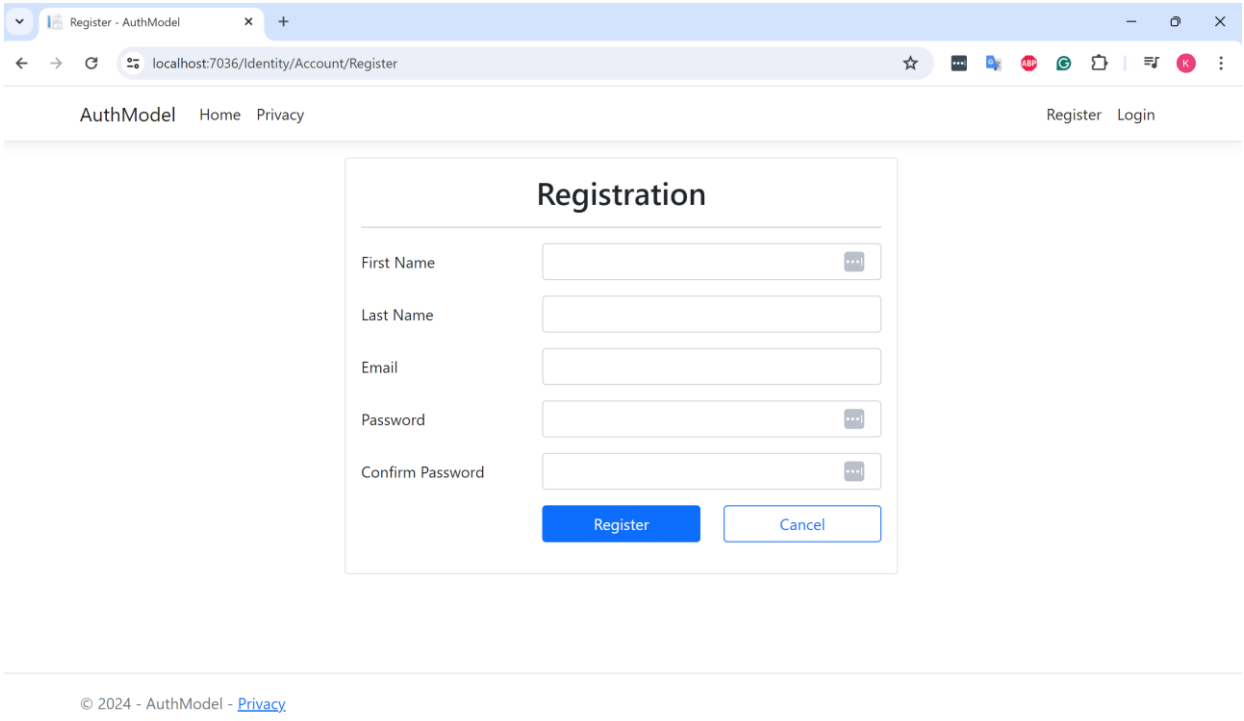

Рисунок 3.7 – Інтерфейс сторінки реєстрації

Було налаштовано сторінку входу користувача за допомогою файлу Login.cshtml. Інтерфейс сторінки входу показано на рис. 3.8.

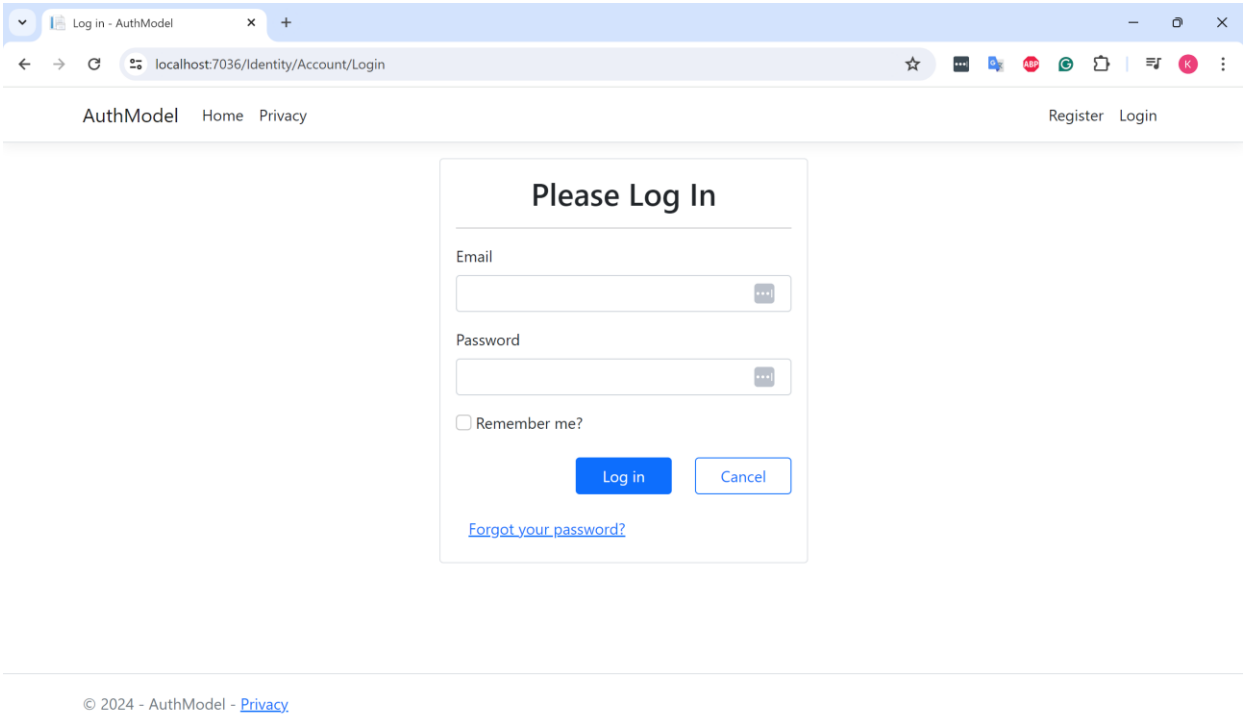

Рисунок 3.8 – Інтерфейс сторінки входу

Було налаштовано інтерфейс та функціонал виходу з акаунту за допомогою файлів Logout.cshtml та Logout.cshtml.cs. Ці файли було модифіковано з наступною метою: коли користувач виходить з акаунту, то додаток перенаправляє користувача на домашню сторінку замість сторінки виходу.

Для реалізації авторизації було створено 3 сторінки типу Razor Pages для користувача, адміністратора та преміум користувача (User.cshtml, User.cshtml.cs; Admin.cshtml, Admin.cshtml.cs; PremiumUser.cshtml, PremiumUser.cshtml.cs). Файли інтерфейсу було змінено, так щоб відображало вітальне повідомлення. Функціональні файли було модифіковано за допомогою [Authorize] для забезпечення авторизації. Для забезпечення авторизації на основі ролей необхідно змінити налаштування Program.cs, додаючи функцію .AddRoles<IdentityRole>. Головна сторінка має вигляд, що показаний на рис. 3.9.

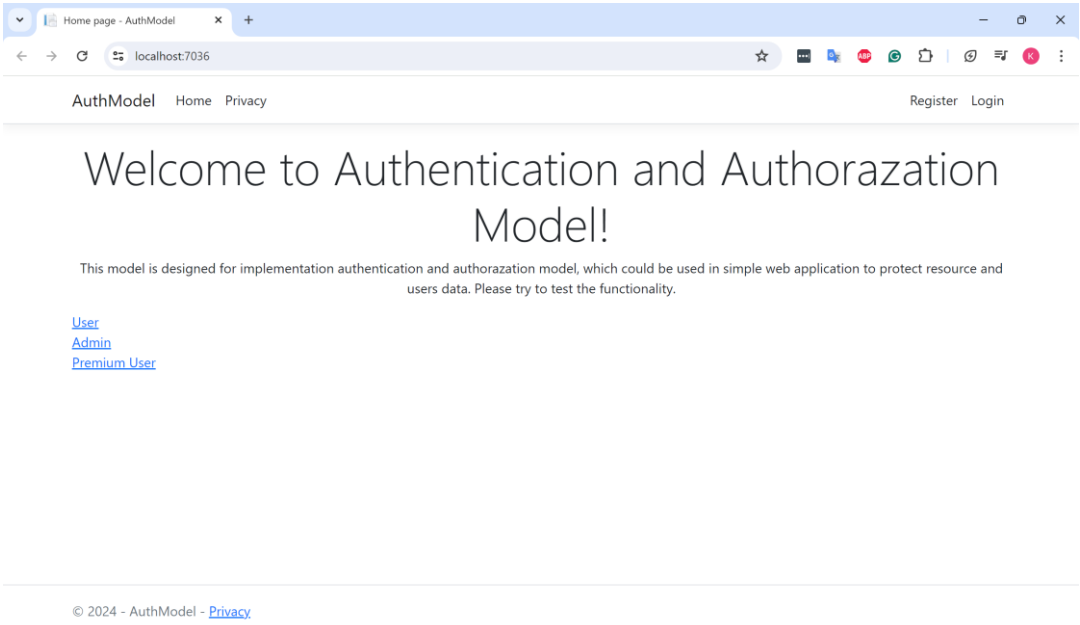

Рисунок 3.9 – Інтерфейс головної сторінки

Для автентифікованих користувачів доступна функція підключення двофакторної автентифікації за допомогою додатків-автентифікаторів, Google Authenticator та Microsoft Authenticator. Для підключення функції було імплементовано генерацію QR-коду. Ця функція була імплементована за допомогою JavaScript коду QRCode.js. Також було змінено інтерфейс EnableAuthenticator.cshtml. Також був змінений функціональний код сторінки EnableAuthenticator.cshtml.cs. Для реалізації цієї функції було створено скрипт файл qr.js. Результат впровадження генерації QR-коду показано на рис. 3.10.

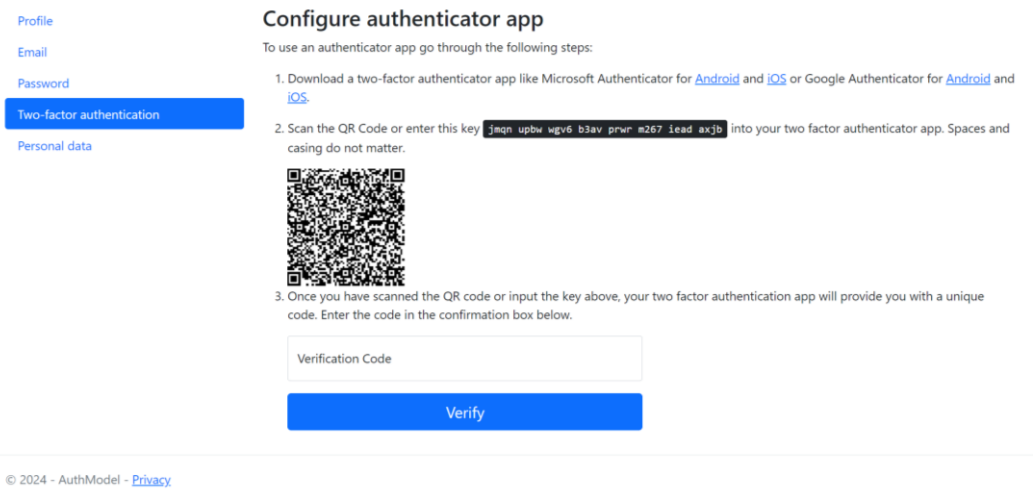

Рисунок 3.10 – Інтерфейс сторінки конфігурації додатка-автентифікатора

Для реалізації двофакторної автентифікації для адміністратора за допомогою OTP, необхідно для початку верифікувати електронну пошту користувача. Для цього була проведена реєстрація в сервісі SendGrid. За допомогою цього сервісу будуть відправлятися листи верифікації та OTP. Налаштована пошта для верифікації показана на рис. 3.11.

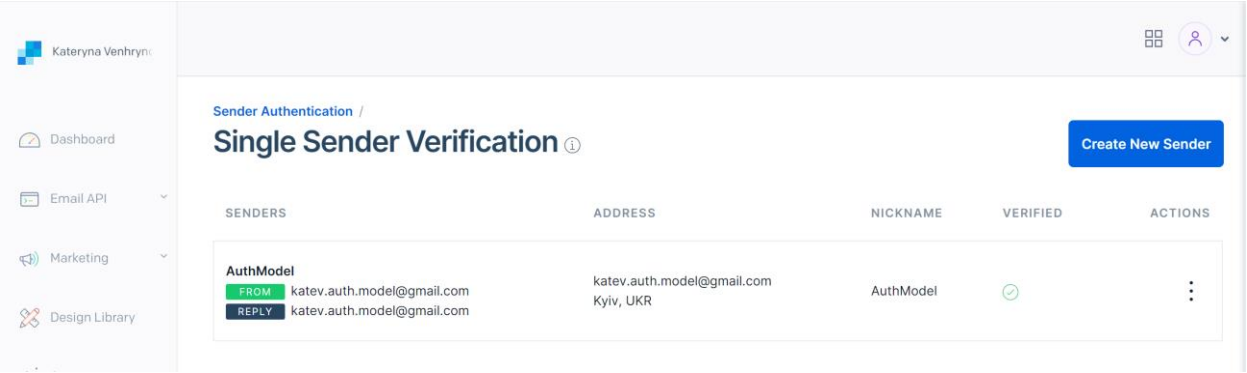

Рисунок 3.11 – Налаштований сервіс SendGrid

Для підключення даних для надсилання електронних листів було змінено файл appsettings.Development.json. Для налаштування функції надсилання листів було створено SendGridSettings.cs. В файлі Program.cs було сконфігуровано створений клас для надсилання листів. Також для реалізації було створено службу EmailSenderServices.cs.

Для реалізації двофакторної автентифікації за допомогою OTP, що надісланий через електронну пошту, необхідно налаштувати файл LoginWith2fa.cshtml.cs.

3.3 Тестування функцій моделі автентифікації та авторизації

Для тестування функціональності реалізованої моделі зареєструємо три акаунти користувачів:

- standard.user.mod@ukr.net – акаунт стандартного користувача (рис. 3.12);
- admin.user.mod@ukr.net – акаунт для адміністратора (рис. 3.13);
- premium.user.mod@ukr.net – акаунт для преміум користувача (рис. 3.14).

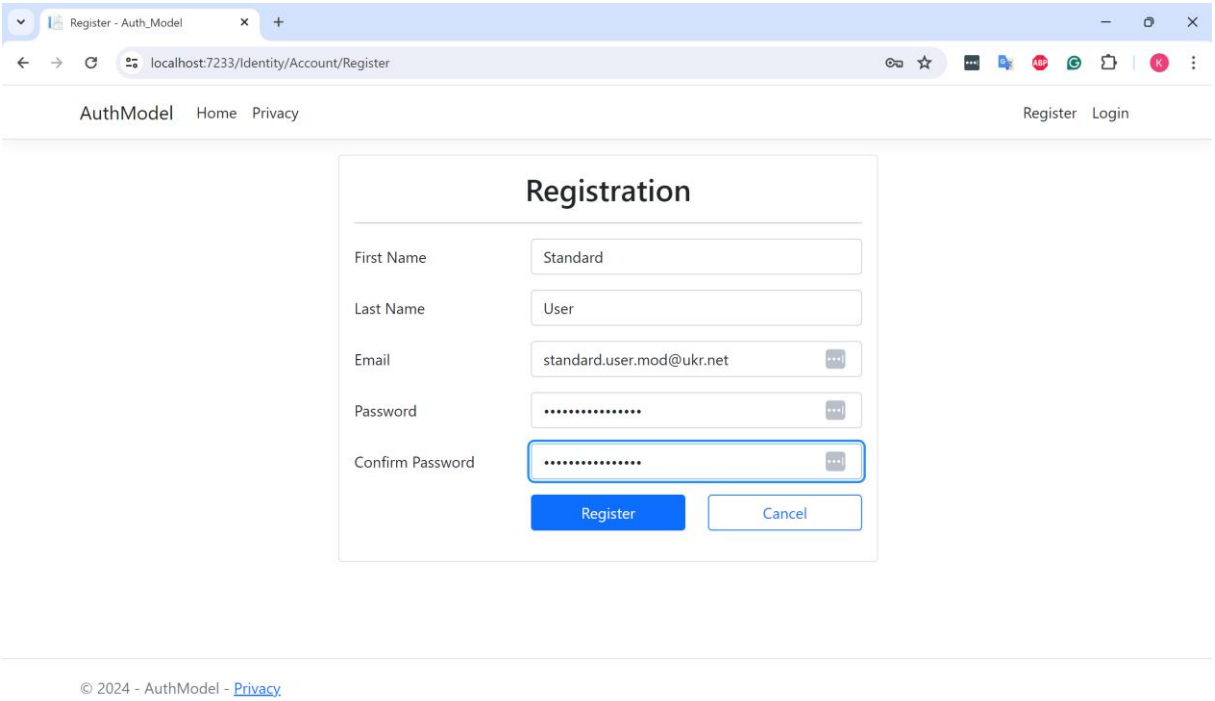

Рисунок 3.12 – Реєстрація стандартного акаунту

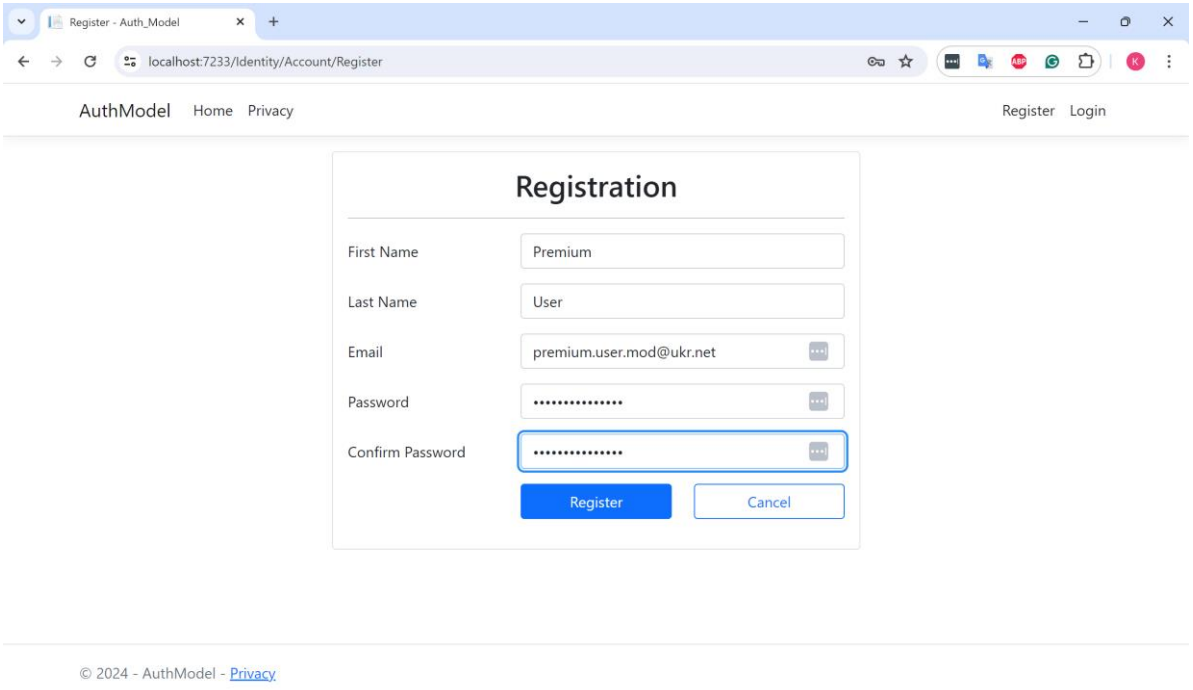

Рисунок 3.13 – Реєстрація преміум акаунту

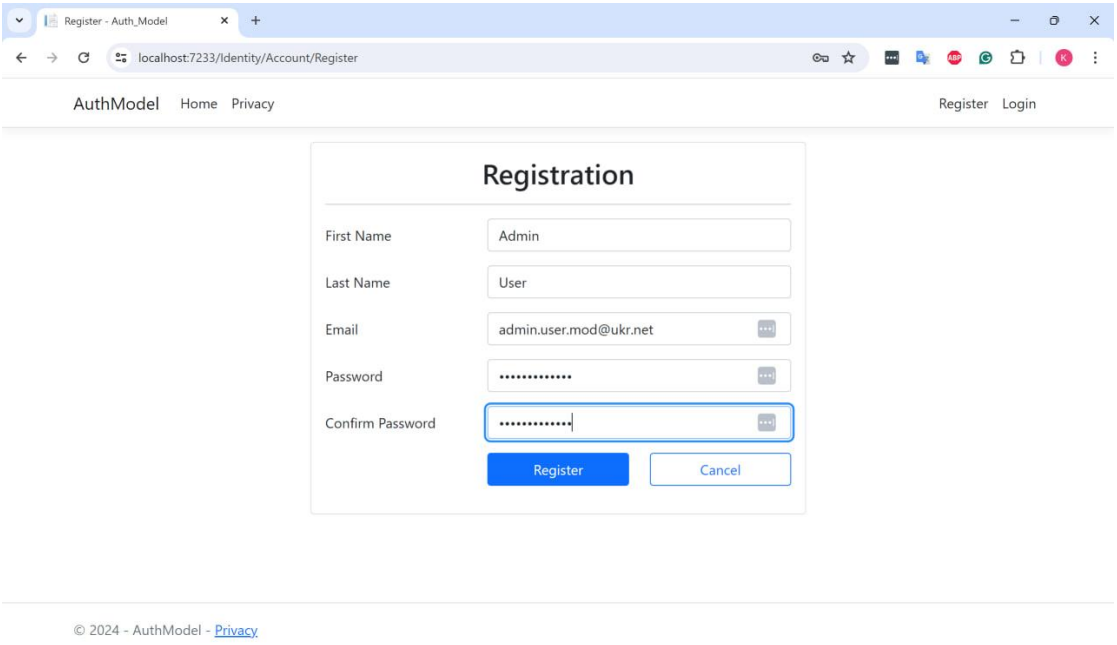

Рисунок 3.14 – Реєстрація акаунту адміністратора

Після успішної реєстрації користувачів було перевірено таблиці бази даних AspNetUsers (рис. 3.15), AspNetRoles (рис. 3.16) та AspNetUserRoles (рис. 3.17). Таблиця AspNetUsers містить дані про створених користувачів, AspNetRoles визначає створені ролі в проєкті, AspNetUserRoles виражає присвоєні ролі користувачів.

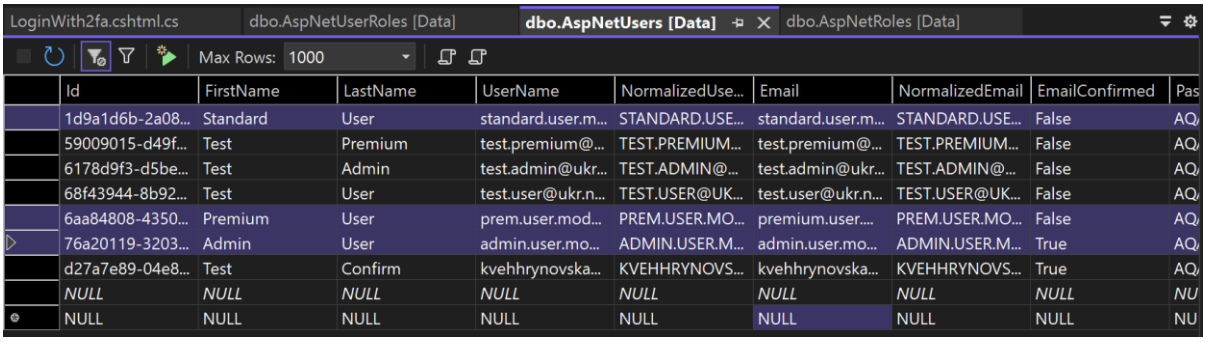

Рисунок 3.15 – Таблиця AspNetUsers

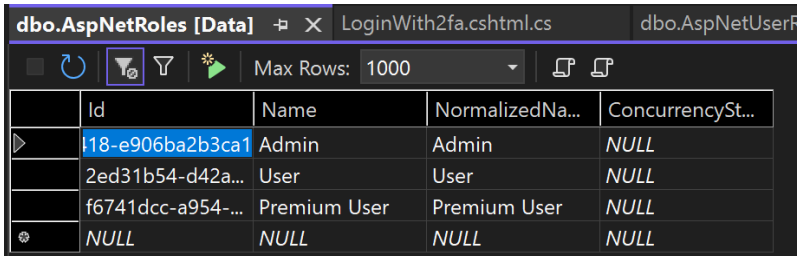

Рисунок 3.16 – Таблиця AspNetRoles

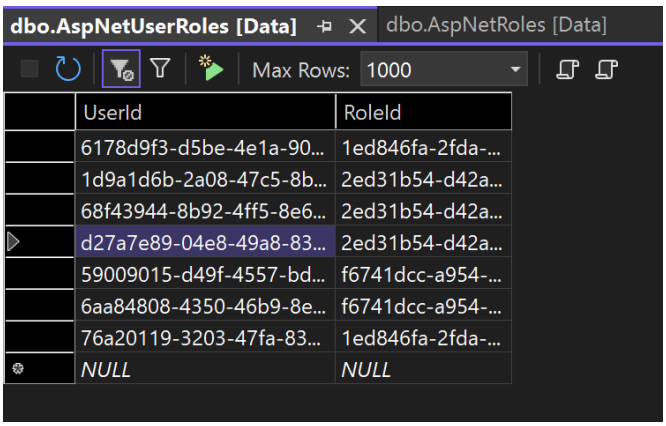

Рисунок 3.17 – Таблиця AspNetUserRoles

Було здійснено вхід в акаунт стандартного користувача, головна сторінка якого показана на рис. 3.18. При спробі доступу до ресурсу User була відображена сторінка, що показана на рис. 3.19. При спробі доступу до ресурсу Admin була відображена веб-сторінка з відмовою в доступі (рис. 3.20).

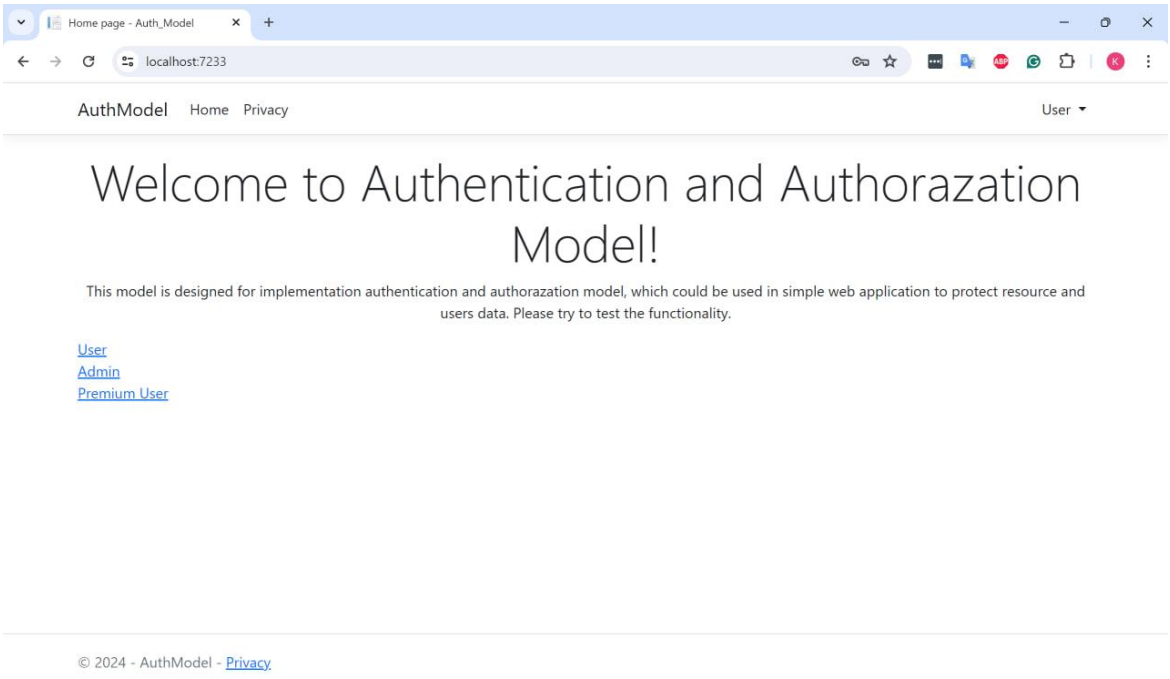

Рисунок 3.18 – Головна сторінка стандартного користувача

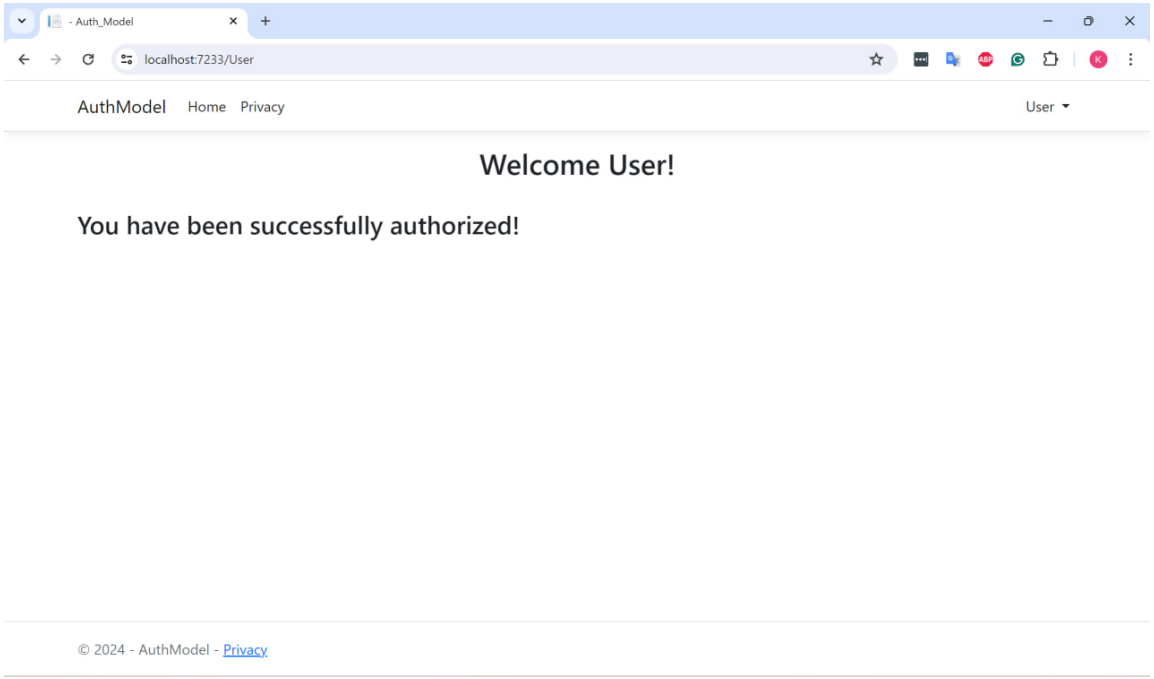

Рисунок 3.19 – Доступ до ресурсу User

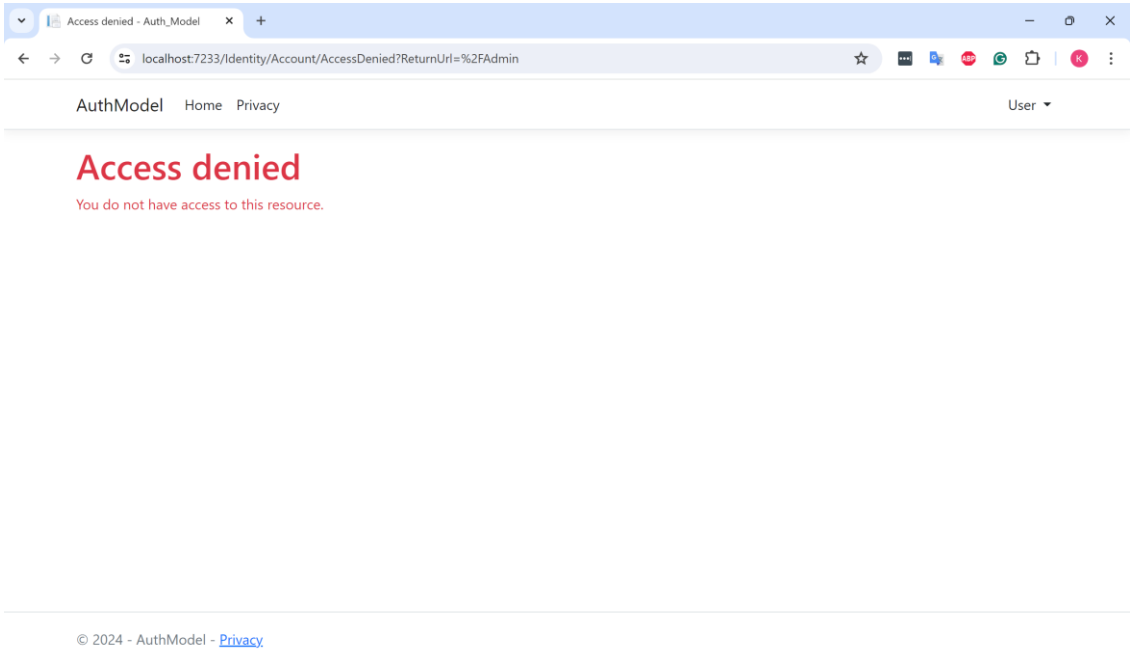

Рисунок 3.20 – Відмова в доступі до ресурсу Admin

Вхід в акаунт преміум користувача показаний на рис. 3.21. Головна сторінка преміум користувача відображена на рис. 3.22. Доступ до ресурсу Premium User показаний на рис. 3.23. Відмова в доступі до ресурсів User та Admin показана на рис. 3.24 та рис. 3.25.

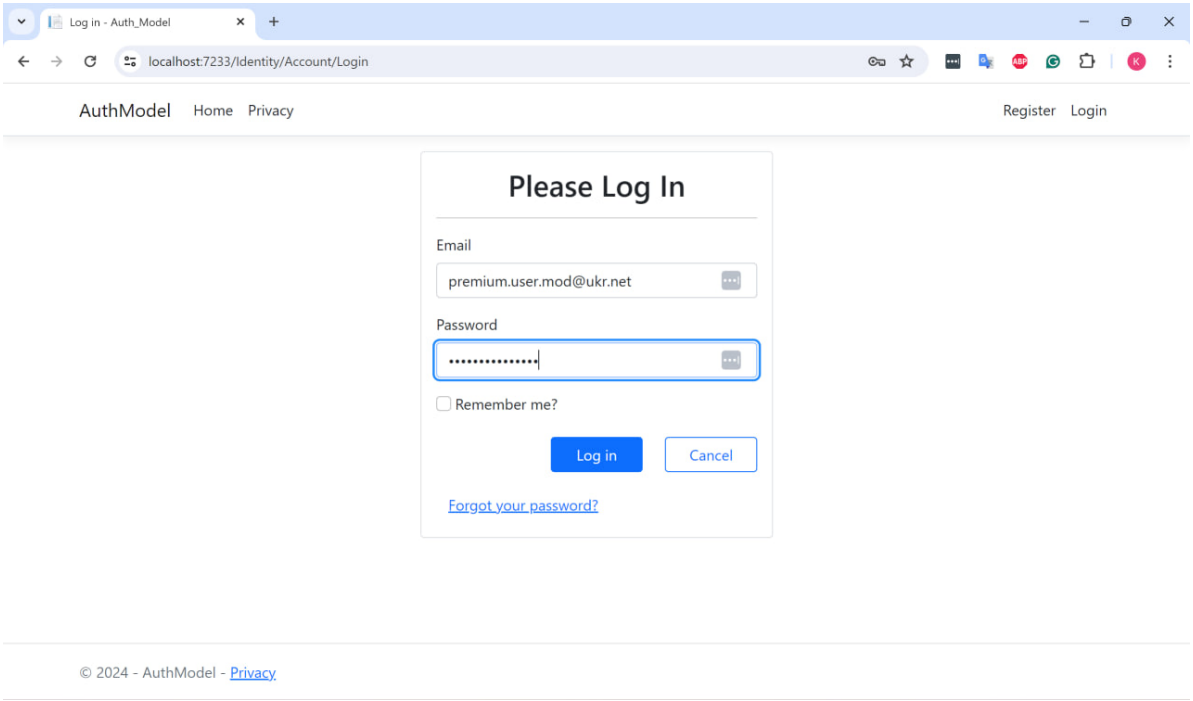

Рисунок 3.21– Вхід в акаунт преміум користувача

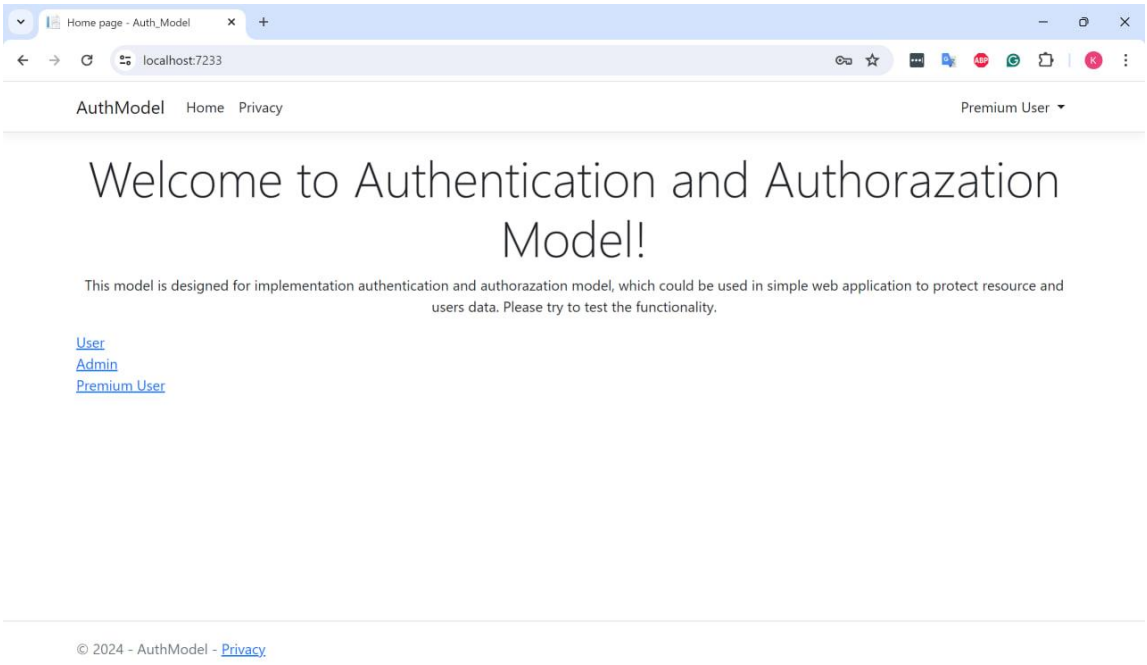

Рисунок 3.22 – Головна сторінка преміум користувача

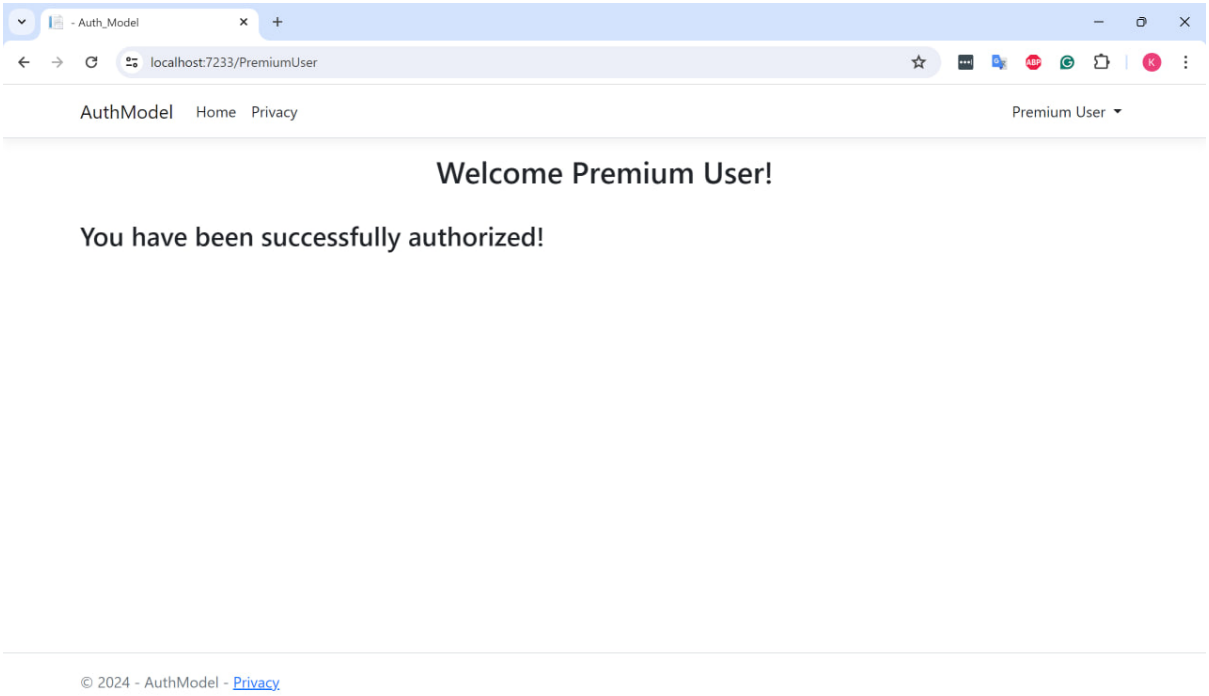

Рисунок 3.23 – Доступ до ресурсу Premium User

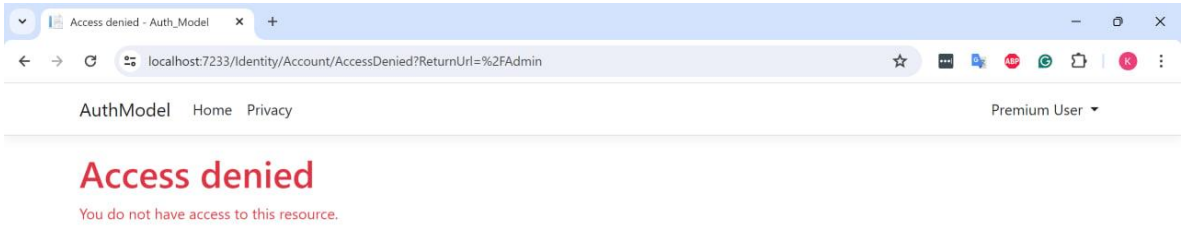

Рисунок 3.24 – Відмова в доступі до ресурсу Admin

Рисунок 3.25 – Відмова в доступі до ресурсу User

Для преміум користувача було ввімкнено двофакторну автентифікацію за допомогою Google Authenticator (рис. 3.26 – 3.28).

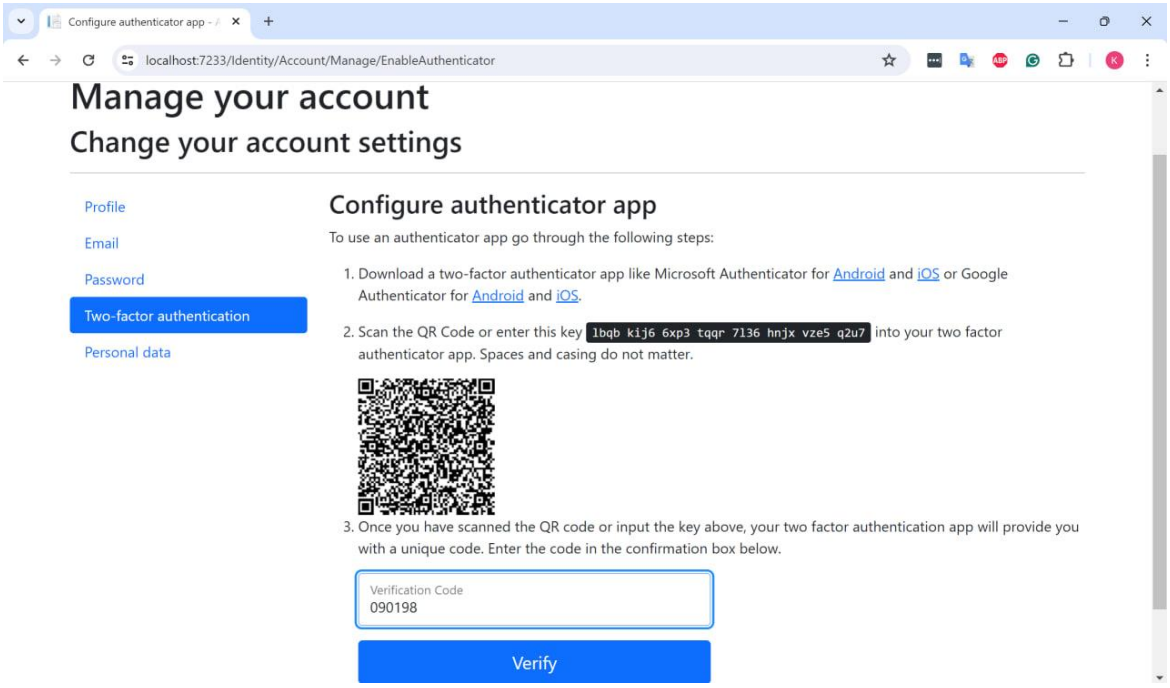

Рисунок 3.26 – Налаштування Google Authenticator

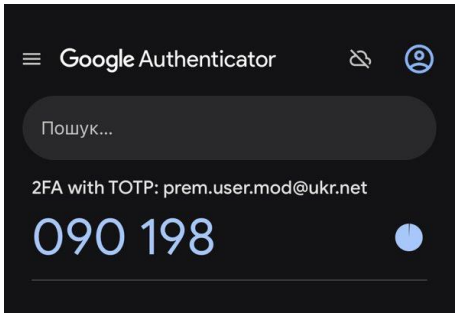

Рисунок 3.27 – TOTP для верифікації

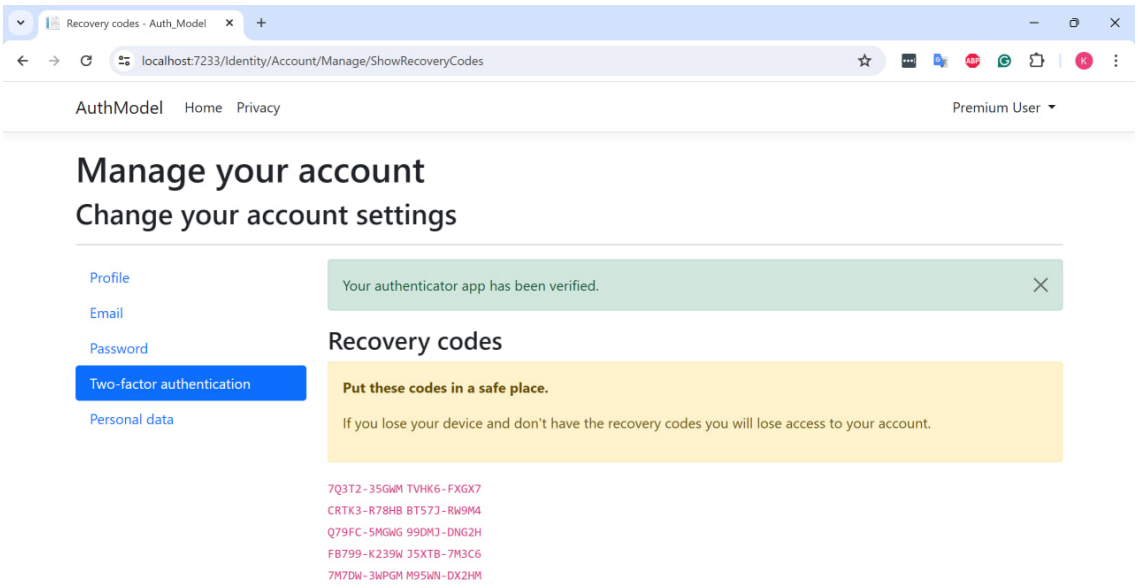

Рисунок 3.28 – Підтвердження успішного налаштування

Вхід за допомогою налаштованої двофакторної автентифікації для преміум користувача показано на рис. 3.29 та 3.30.

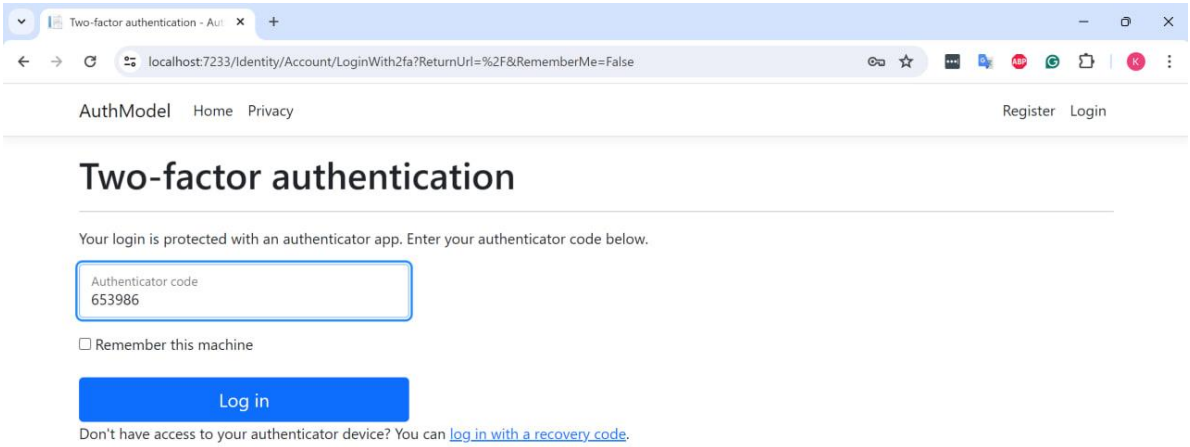

Рисунок 3.29 – Двофакторна автентифікація з Google Authenticator

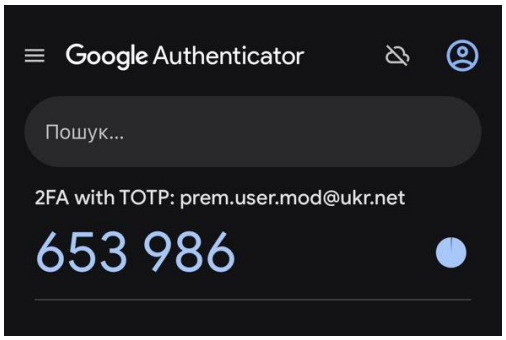

Рисунок 3.30 – TOTP для автентифікації

Підтвердження електронної пошти для акаунту адміністратора показано на рис. 3.31 – 3.33.

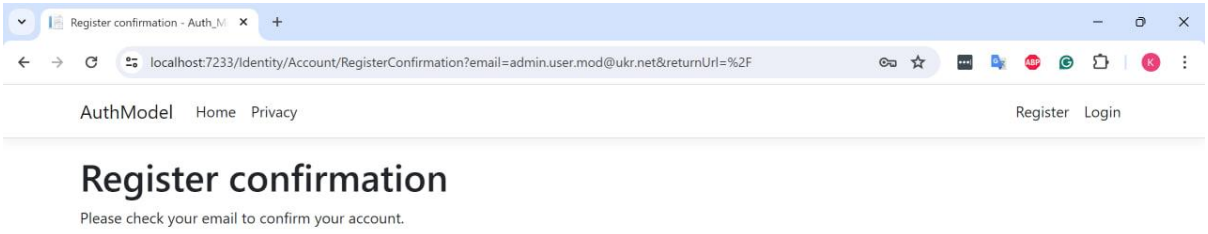

Рисунок 3.31 – Повідомлення про лист підтвердження

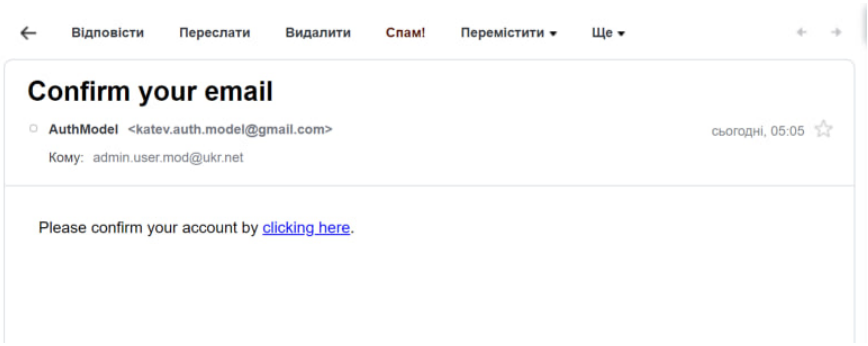

Рисунок 3.32 – Лист з посиланням для підтвердження електронної пошти

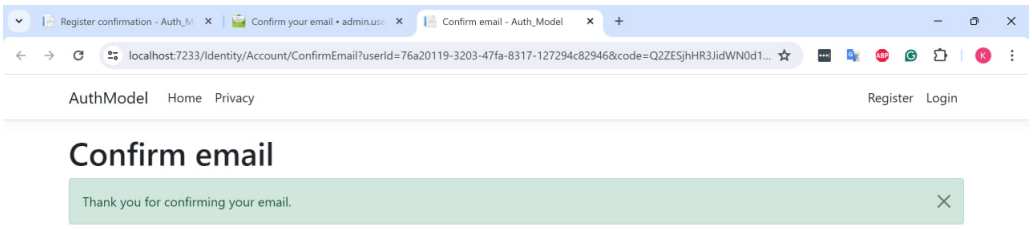

Рисунок 3.33 – Повідомлення про успішне підтвердження

Вхід в акаунт адміністратора за допомогою OTP показано на рис. 3.34 – 3.36.

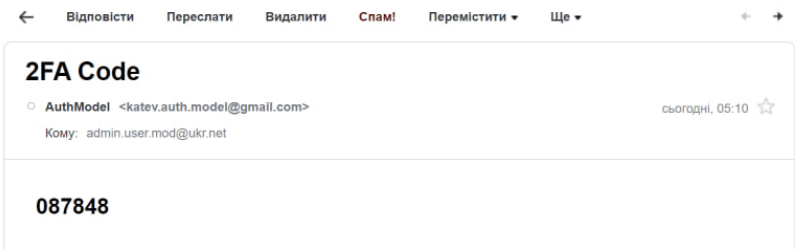

Рисунок 3.34 – OTP для двофакторної автентифікації

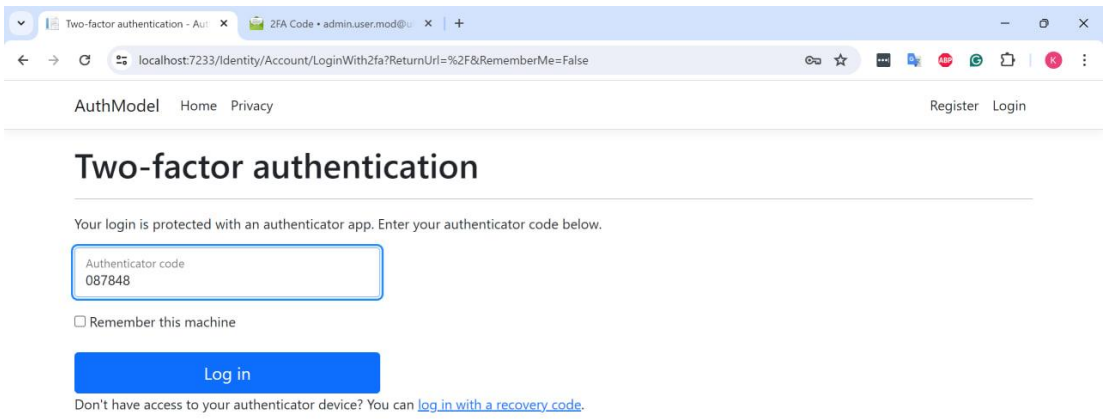

Рисунок 3.35 – Введення OTP

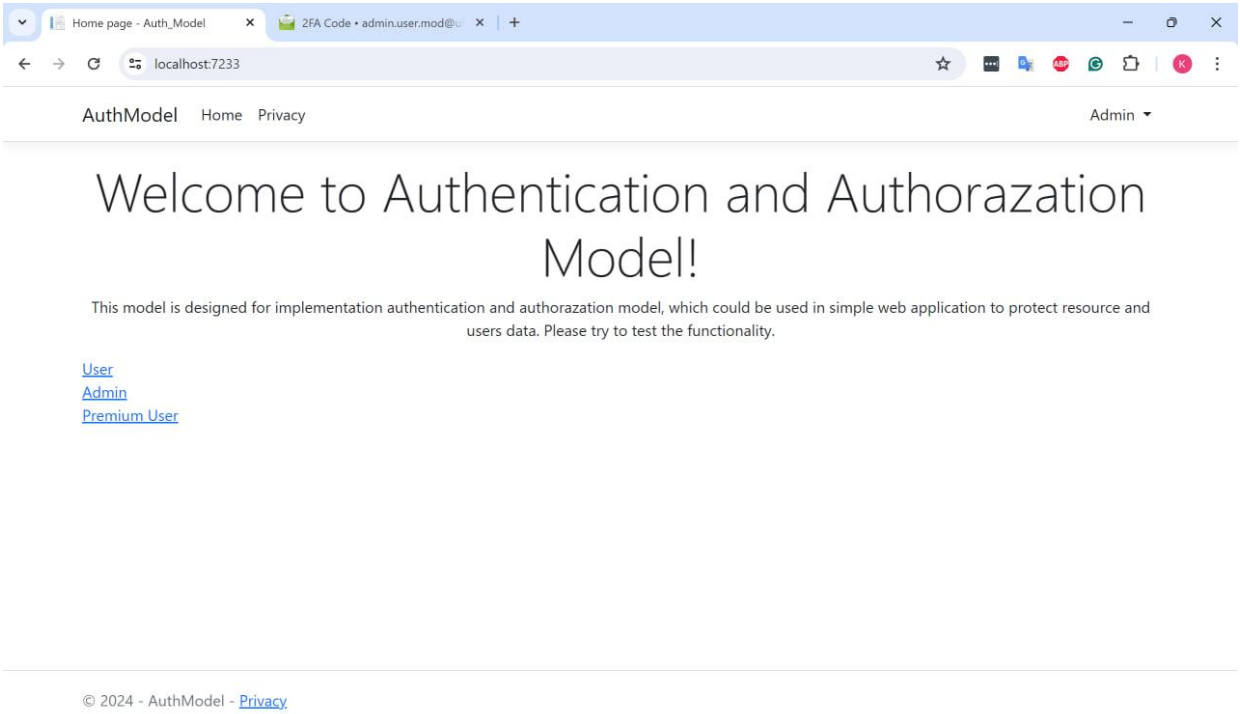

Рисунок 3.36 – Головна сторінка адміністратора

Успішний доступ користувача адміністратор до ресурсу Admin показано на рис. 3.37. Відмова в доступі до ресурсів User та Premium User показана на рис. 3.38 та 3.39 відповідно.

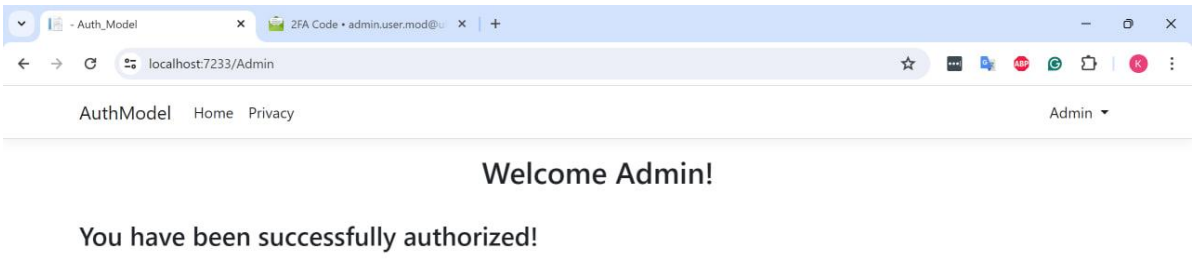

Рисунок 3.37 – Доступ до ресурсу Admin

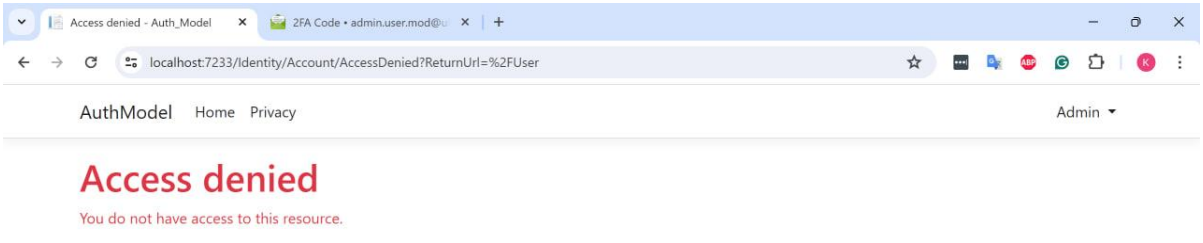

Рисунок 3.38 – Відмова в доступі до ресурсу User

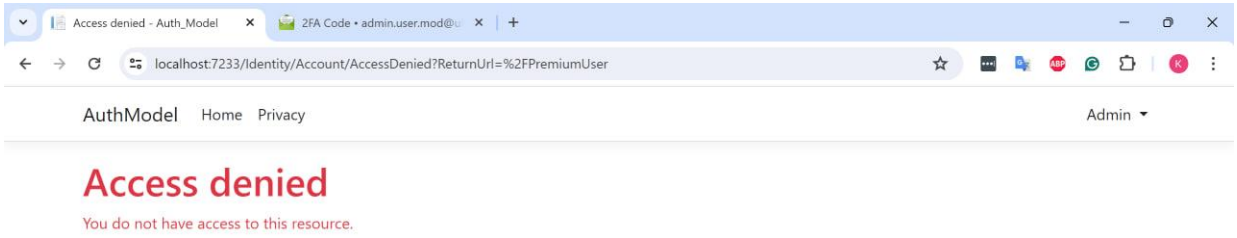

Рисунок 3.39 – Відмова в доступі до ресурсу Premium User

Отже, реалізована модель автентифікації та авторизації задовільняє функціональні потреби розробленої моделі.

Висновки за розділом 3

У третьому розділі дипломної роботи було розроблено модель автентифікації та авторизації для найпростіших вебдодатків, що забезпечує двофакторну автентифікацію та авторизацію на основі ролей.

Розроблену модель було реалізовано за допомогою ASP.NET Core Identity та середовища Visual Studio 2022. Для реалізації було інстальовано та використано наступні пакети: Microsoft.AspNetCore.Identity.EntityFrameworkCore, Microsoft.AspNetCore.Identity.UI, Microsoft.EntityFrameworkCore.SqlServer, Microsoft.EntityFrameworkCore.Tools, SendGrid, SendGrid.Extensions.DependencyInjection.

Для досягнення умов розробленої моделі автентифікації та авторизації було налаштовано наступні функції: створені додаткові атрибути користувача; налаштовано форму реєстрації та входу; можливість двофакторної автентифікації за допомогою додатка-автентифікатора та OTP надісланого на електронну пошту користувача; реалізовано рольову модель, з ролями стандартний користувач, адміністратор та преміум користувач.

Тестування функціональних можливостей реалізованої моделі показує, що умови до автентифікації та авторизації розробленої моделі були виконані.

ВИСНОВКИ

Кваліфікаційна робота пропонує рішення для захисту вебдодатків використовуючи удосконалену модель автентифікації та авторизації, що є актуальним завданням в сучасних реаліях.

Під час написання кваліфікаційної роботи було досліджено теоретичні основи автентифікації та авторизації. Було розглянуто методи та протоколи автентифікації та авторизації. Також було проаналізовано можливості сучасних технологій для реалізації авторизації та автентифікації та надано інформацію щодо їх кращого використання для досягнення найоптимальніших результатів. Правильний вибір технології автентифікації та авторизації забезпечить надійний захист конфіденційним даним ресурсу та даним користувачів.

Було розглянуто можливості ASP.NET Core Identity для реалізації автентифікації та авторизації. Було проаналізовано компоненти, архітектуру та основні особливості даної технології. Також надано перелік методів та можливостей для реалізації автентифікації та авторизації.

Було удосконалено модель автентифікації та авторизації для найпростіших вебдодатків, що забезпечує двофакторну автентифікацію та авторизацію на основі ролей. Удосконалену модель було реалізовано за допомогою ASP.NET Core Identity та середовища Visual Studio 2022. Для досягнення умов запропонованої моделі було налаштовано наступні функції: створені додаткові атрибути користувача; налаштовано форму реєстрації та входу; можливість двофакторної автентифікації за допомогою додатка-автентифікатора та OTP надісланого на електронну пошту користувача; реалізовано рольову модель, з ролями стандартний користувач, адміністратор та преміум користувач.

Отже, було досягнуто мету дипломної роботи – розроблено та реалізовано модель автентифікації та авторизації за допомогою ASP.NET Core Identity.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Authentication vs. Authorization [Electronic resource] – Access: https://www.onelogin.com/learn/authentication-vs-authorization

2. William Stallings, Cryptography and Network Security: Principles and Practice, 7th Edition, Pearson Education, 2017

3. Authentication vs. Authorization [Electronic resource] – Access: https://www.okta.com/identity-101/authentication-vs-authorization/

4. What Is Authorization? Definition & Examples [Electronic resource] – Access: https://www.netsuite.com/portal/resource/articles/erp/authorization.shtml

5. Difference between Authentication and Authorization [Electronic resource] – Access: https://www.geeksforgeeks.org/difference-between-authentication-and-authorization/

6. Authentication vs. Authorization: Key Differences [Electronic resource] – Access: https://www.fortinet.com/resources/cyberglossary/authentication-vs-authorization

7. Piyush Pant, Anand Singh Rajawat, S.B.Goyal, Pradeep Bedi, Chaman Verma, Maria Simona Raboaca, Florentina Magda Enescu, “Authentication and Authorization in Modern Web Apps for Data Security Using Nodejs and Role of Dark Web” 4th International Conference on Innovative Data Communication Technology and Application, Procedia Computer Science 215 (2022), pp. 781–790.

8. S. Subrayan, S. Mugilan, B. Sivanesan and S. Kalaivani, “Multi-factor Authentication Scheme for Shadow Attacks in Social Network”, 2017 International Conference on Technical Advancements in Computers and Communications (ICTACC), 2017, pp. 36-40.

9. Serhii Buchyk AUTHENTICATION AND AUTHORIZATION METHODS IN A WEB APPLICATION / Serhii Buchyk, Kateryna Venhrynovska // X International conference “Information Technology and Implementation”, 21 November, 2023.

10. Syed Zulkarnain Syed Idrus, Estelle Cherrier, Christophe Rosenberger, Jean-Jacques Schwartzmann. A Review on Authentication Methods. Australian Journal of Basic and Applied Sciences, 2013, 7 (5), pp.95-107.

11. Authorization Methods [Electronic resource] – Access: https://www.pingidentity.com/en/resources/identity-fundamentals/authorization/authorization-methods.html

12. What is Authorization? [Electronic resource] – Access: https://auth0.com/intro-to-iam/what-is-authorization

13. Authentication vs Authorization [Electronic resource] – Access: https://frontegg.com/blog/authentication-vs-authorization

14. Access Control Models: MAC, DAC, RBAC, & PAM Explained [Electronic resource] – Access: https://www.twingate.com/blog/other/access-control-models

15. Attribute-Based Access Control: Pros, Cons & Use Cases [Electronic resource] – Access: https://www.digitalguardian.com/blog/attribute-based-access-control#:~:text=The%20Disadvantages%20of%20Using%20ABAC%201%20Increased%20complexity%3A,harder%20for%20the%20system%20to%20be%20comprehensively%20audited.

16. Guide to Discretionary Access Control (DAC) With Examples [Electronic resource] – Access: https://builtin.com/articles/discretionary-access-control

17. What is Discretionary Access Control? Your Complete DAC Guide [Electronic resource] – Access: https://butterflymx.com/blog/discretionary-access-control/

18. A Guide to Authentication Protocols [Electronic resource] – Access: https://www.descope.com/learn/post/authentication-protocols

19. Explain HTTP authentication [Electronic resource] – Access: https://www.geeksforgeeks.org/explain-http-authentication/

20. Lightweight Directory Access Protocol [Electronic resource] – Access: https://learn.microsoft.com/en-gb/previous-versions/windows/desktop/ldap/lightweight-directory-access-protocol-ldap-api

21. Lightweight Directory Access Protocol (LDAP) [Electronic resource] – Access: https://www.portnox.com/cybersecurity-101/lightweight-directory-access-protocol-ldap/

22. Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms [Electronic resource] – Access: https://www.rfc-editor.org/rfc/rfc4513.html#section-1

23. The LDAP Bind Operation [Electronic resource] – Access: https://ldap.com/the-ldap-bind-operation/

24. Remote Authentication Dial In User Service (RADIUS) [Electronic resource] – Access: https://www.rfc-editor.org/rfc/rfc2865.html

25. What is RADIUS Server and how does RADIUS Server Authentication work? [Electronic resource] – Access: https://www.miniorange.com/blog/radius-server-authentication/

26. PPP Challenge Handshake Authentication Protocol (CHAP) [Electronic resource] – Access: https://www.ietf.org/rfc/rfc1994.txt

27. Microsoft PPP CHAP Extensions [Electronic resource] – Access: https://www.rfc-editor.org/rfc/rfc2433

28. Extensible Authentication Protocol (EAP) [Electronic resource] – Access: https://www.rfc-editor.org/rfc/rfc3748.html

29. What Is RADIUS? [Electronic resource] – Access: https://info.support.huawei.com/info-finder/encyclopedia/en/RADIUS.html

30. What Is Remote Authentication Dial In User Service(RADIUS)? [Electronic resource] – Access: https://community.fs.com/article/what-is-remote-authentication-dial-in-user-service.html

31. What is Kerberos? [Electronic resource] – Access: https://www.techtarget.com/searchsecurity/definition/Kerberos

32. Jason Garman, Kerberos: The Definitive Guide: The Definitive Guide, First Edition, O'Reilly Media, Inc, 2003

33. What Is Kerberos? How Does Kerberos Work: Everything You Need to Know [Electronic resource] – Access: https://www.simplilearn.com/what-is-kerberos-article

34. Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M. Pai, Sanjay Singh, "Formal Verification of OAuth 2.0 Using Alloy Framework", 2011 International Conference on Communication Systems and Network Technologies, 2011, pp. 655 - 659.

35. What Is OAuth? Definition, How It Works, and Tips for Success [Electronic resource] – Access: https://frontegg.com/blog/oauth

36. OAuth - Pros and Cons [Electronic resource] – Access: https://www.linkedin.com/pulse/oauth-pros-cons-digialert/

37. Oauth 2.0 Basic Understanding [Electronic resource] – Access: https://stfalcon.com/en/blog/post/oauth-2.0

38. Should You Use OAuth 2.0? Pros and Cons [Electronic resource] – Access: https://www.digitalinformationworld.com/2023/11/should-you-use-oauth-20-pros-and-cons.html

39. Belfaik Yousra, Sadqi Yassine, Maleh Yassine, Safi Said, Tawalbeh Lo'ai, Khaled Salah "A Novel Secure and Privacy-Preserving Model for OpenID Connect Based on Blockchain", IEEEAccess, Volume 11, 2023, pp. 67660 - 67678.

40. M. Schwartz and M. Machulak, Securing the Perimeter: Deploying Identity and Access Management With Free Open Source Software. Berkeley, CA, USA: Apress, 2018, pp. 151–203.

41. Akshay Rasiwasia, A Framework To Implement OpenID Connect Protocol For Federated Identity Management In Enterprises, Luleå University of Technology Department of Computer Science, Electrical and Space Engineering, 2017 [Electronic resource] – Access: https://www.diva-portal.org/smash/get/diva2:1121361/FULLTEXT01.pdf

42. What is JWT (JSON Web Token)? How does JWT Authentication work? [Electronic resource] – Access: https://www.miniorange.com/blog/what-is-jwt-json-web-token-how-does-jwt-authentication-work/

43. Paul Madsen, Eve Maler, SAML V2.0 Executive Overview, Committee Draft 01, 12 April 2005

44. What is SAML 2.0? [Electronic resource] – Access: https://auth0.com/intro-to-iam/what-is-saml

45. SAML 2.0 Protocol Overview [Electronic resource] – Access: https://community.denodo.com/kb/en/view/document/saml%202.0%20protocol%20overview

46. Common Problems with Authorization and Authentication and How to Solve Them [Electronic resource] – Access: https://www.linkedin.com/pulse/common-problems-authorization-authentication-how-solve-mateusz-cio%C5%82ek/

47. CHALLENGES OF USER AUTHENTICATION: WHAT YOU NEED TO KNOW [Electronic resource] – Access: https://securityaffairs.com/135434/security/challenges-of-user-authentication.html

48. Common authentication and authorization vulnerabilities (and how to avoid them) [Electronic resource] – Access: https://www.invicti.com/blog/web-security/how-to-avoid-authentication-and-authorization-vulnerabilities/

49. Introduction to Identity on ASP.NET Core [Electronic resource] – Access: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio

50. Сергій Бучик Аналіз технологій автентифікації та авторизації / Сергій Бучик, Катерина Венгриновська // VII Міжнародна науково-практична конференція “Проблеми кібербезпеки інформаційно-телекомунікаційних систем (PCSITS)”, 26 квітня 2024 року

51. Tram M. H. B. FIREBASE: Thesis. 2019. 58 p.

52. Lysakov V., Sievierinov O., Taran I. Security of Web Applications Using AWS Cloud Provider. Computer and Information Systems and Technologies, Kharkiv. 2021. P. 35–36.

53. Auth0: Key Features, Technical Overview, and Alternatives [Electronic resource] – Access: https://frontegg.com/guides/auth0

54. Authentication and Authorization Flows [Electronic resource] – Access: https://auth0.com/docs/get-started/authentication-and-authorization-flow

55. Zaal S. Azure Active Directory for Secure Application Development: Use Modern Authentication Techniques to Secure Applications in Azure. Packt Publishing, Limited, 2022.

56. What is Google Cloud Identity? [Electronic resource] – Access: https://www.goldyarora.com/blog/what-is-google-cloud-identity

57. Authentication and authorization [Electronic resource] – Access: https://cloud.google.com/identity/docs/concepts/auth

58. ASP.NET Core Identity Tutorials [Electronic resource] – Access: https://dotnettutorials.net/course/asp-net-core-identity-tutorials/

ДОДАТОК А

Копія наукової публікації

X Міжнародна науково-практична конференція «Інформаційні технології та впровадження» (IT&I-2023)

[1] Serhii Buchyk

Doctor of Technical Sciences, Professor at the Department of Cyber Security and Information Protection, Faculty of Information Technologies

[2] Kateryna Venhrynovska

Student at the Department of Cyber Security and Information Protection, Faculty of Information Technologies

[1] Taras Shevchenko National University of Kyiv

AUTHENTICATION AND AUTHORIZATION METHODS IN A WEB APPLICATION

Abstract. The number of web applications that provide users with various functions and services is increasing every day. This rapid development requires great attention to the security and confidentiality of information. The foundation of security is the authentication and authorization model, which ensures that only identified users with specific permissions can access resources. This article provides a general description of the existing models of authentication and authorization in web applications.

Keywords: authentication, authorization, OAuth 2.0, OpenID Connect.

Modern web applications offer users a wide range of possibilities, such as using services, exchanging data, communicating, etc. This increases the interest of hackers to gain unauthorized access to personal information. This article discusses web application security using an authentication and authorization model.

User authentication is the basis for most types of access control and user accountability. User authentication as the process of verifying an identity claimed by or for a system entity.

The process consists of two steps: identification and verification. Identification is the presentation of an identifier to the security system and verification is the presentation or generation of authentication information that confirms the binding between the entity and the identifier [1].

There are a number of common authentication methods that can be used alone or in combination [1].

1. Authentication is based on something the user knows. This type includes a password, a personal identification number (PIN), or answers to a prearranged set of questions. Passwords are the most common method. This type requires the user to create a password for their account, and then the password is hashed using hashing algorithms such as SHA-1, Bcrypt, etc. The hashed password is stored in the database, in case the database is compromised, it's an additional protection of the user's identifier [1,2].

2. Authentication is based on something the user possesses. This type includes cryptographic keys, electronic keycards, smart cards, and physical keys. This method is also known as token-based authentication. With token-based authentication technologies, users can submit their credentials once and receive a unique encrypted string of random characters in return [1, 2].

3. Authentication is based on the unique biological characteristics of an individual. This method can be divided into static and dynamic biometrics. Static biometrics includes recognition by fingerprint, retina, and face. Another type, dynamic biometrics, includes recognition by voice pattern, handwriting characteristics, and typing rhythm [1, 2].

4. Multi-factor authentication is an extended and improved version of basic authentication. It has multiple layers and stages of authentication that are password based, OTP (One Time Password), email verification, unique identity verification or even sometimes a special question. These layers are added to form a single authentication system and the user has to pass all of them to authenticate himself [2, 3].

5. Authentication is based on certificates. Certificates contain a user's digital identity, including a public key, and the digital signature of a certification authority. Only a Certification Authority can issue digital certificates to prove ownership of a public key [2].

There are a number of open specifications and protocols that define how to design an

authentication and authorization system. They define how identities should be managed, how personal data should be securely moved, and who can access applications and data. There are five main authentication and authorization protocols: Kerberos, LDAP, OAuth 2.0, RADIUS, SAML [4, 5].

The OAuth 2.0 is one of the modern protocols. It is the protocol through which a client obtains an access token from an authorization server, to access a protected resource, stored in a resource server. OAuth is an authorization framework in which a third-party application is delegated limited right to access the user information that is stored in another web service. OAuth 2.0 provides greater developer convenience and a simpler authentication process than OAuth 1.0 and OAuth 1.1. It does this by removing a complicated encryption process and using the HTTPS protocol [6, 7].

The OpenID Connect (OIDC) authentication protocol, which is compatible with OAuth 2.0, is used to simplify the way users' identities are verified based on authentication performed by the authentication server. It allows third-party applications to verify the identity of the end user and obtain basic user profile information. OIDC uses JSON Web Tokens (JWTs), which can be obtained using flows that conform to the OAuth 2.0 specification [8].

The summary results characterizing the identified authentication and authorization methods in web applications are presented in table 1.

Table 1

Authentication and authorization methods in a web application

S. No	**Authentication and authorization method**	**Features of use**	**Advantages, disadvantages**
1	Authentication based on something the user knows	- Use of an identifier (login and password) known only to the user. - Store passwords in a secure format (password hashing).	Advantages: - easy to implement and use; - protection from unauthorized access.

Table continuation 1

		- Set minimum password length and complexity requirements. - Limit the number of failed attempts and lock the account after a certain number of failed attempts. - Use a series of questions to restore access if the user forgets the password. - Change password regularly to improve security. - Create and manage user sessions after successful authentication.	Disadvantages: - use of weak passwords, as it is difficult for a user to remember a complex password; - increased risk of phishing, social engineering, brute force attacks and password interception; - the system should be audited regularly.
2	Authentication based on something the user possesses	- The user must have a physical object or information to access. - The user must have physical access to the facility to perform authentication. - Includes the measurement aspect, where the system checks that the user has the required object.	Advantages: - high security; - no need for passwords. Disadvantages: - loss, theft, and the possibility of counterfeiting; - cost and support for implementation.
3	Authentication based on the unique biological characteristics of	- Uniqueness of the identifier. - Requires the physical presence of the user for authentication.	Advantages: - high security; - convenience and speed;

Table continuation 1

3	an individual		- phishing protection; - the possibility of use in different areas; - a wide range of identifiers. Disadvantages: - cost of implementation; - biometric data storage and protection; - equipment requirements.
4	Multi-factor authentication	- Use multiple factors to identify a user, such as password, biometrics, smart cards, facial recognition, voice, etc. - Mandatory introduction of an additional factor. - The ability to choose different authentication methods depending on the specifics of the application and security requirements.	Advantages: - high security; - reduces the risk of brute force and social engineering attacks; - the possibility of using different factors. Disadvantages: - complication of the procedure by the user; - the risk of losing an additional factor; - infrastructure and support costs.
5	Authentication based on certificates	- Certificates must be generated and issued in a reliable and centralized manner.	Advantages: - high security; - centralized

Table continuation 1

5	Authentication based on certificates	- Private keys need to be stored and managed securely. - Public key distribution for certificate and signature verification. - Certificates have a limited validity period and must be renewed periodically.	management; - protection against Man-in-the-Middle and phishing attacks. Disadvantages: - complexity of implementation; - management costs; - loss of a key or certificate; - compatibility; - key storage requirements.

Thus, the authentication and authorization model in web applications is the fundamental basis for security. The above authentication methods can be combined to achieve a higher level of security. The OpenID and OAuth 2.0 authentication and authorization protocols help to create a strong authentication and authorization model. They can be implemented using ASP.NET Core Identity technology that supports these protocols.

References:

1. William Stallings, Cryptography and Network Security: Principles and Practice, 7th Edition, Pearson Education, 2017

2. Piyush Pant, Anand Singh Rajawat, S.B.Goyal, Pradeep Bedi, Chaman Verma, Maria Simona Raboaca, Florentina Magda Enescu, “Authentication and Authorization in Modern Web Apps for Data Security Using Nodejs and Role of Dark Web” 4th International Conference on Innovative Data Communication Technology and Application, Procedia Computer Science 215 (2022), pp. 781–790

3. S. Subrayan, S. Mugilan, B. Sivanesan and S. Kalaivani, “Multi-factor

Authentication Scheme for Shadow Attacks in Social Network", 2017 International Conference on Technical Advancements in Computers and Communications (ICTACC), 2017, pp. 36-40

4. Okta.com [Electronic resource]. - https://www.okta.com/identity-101/authentication-protocols/

5. GeeksforGeeks.org [Electronic resource]. - https://www.geeksforgeeks.org/types-of-authentication-protocols/

6. Nikos Fotiou, Iakovos Pittaras, Vasilios A. Siris, Spyros Voulgaris, George C. Polyzos, "OAuth 2.0 authorization using blockchain-based tokens", 28 Jan 2020

7. Seongho Hong and Heeyoul Kim, "VaultPoint: A Blockchain-Based SSI Model that Complies with OAuth 2.0", Electronics 2020, 9, 1231

8. Auth0.com [Electronic resource]. - https://auth0.com/docs/authenticate/protocols/openid-connect-protocol#what-is-openid-connect-oidc-

ДОДАТОК Б

Копія наукової публікації

VII Міжнародна науково-практична конференція “Проблеми кібербезпеки інформаційно-телекомунікаційних систем” (PCSITS)”

Аналіз технологій автентифікації та авторизації

Сергій Бучик[1], Катерина Венгриновська[2]

1. Кафедра кібербезпеки та захисту інформації, Київський національний університет імені Тараса Шевченка, УКРАЇНА, м.Київ, вул.Володимирська, 60, E-mail: buchyk@knu.ua

2. Кафедра кібербезпеки та захисту інформації, Київський національний університет імені Тараса Шевченка, УКРАЇНА, м.Київ, вул.Володимирська, 60, E-mail: katievengr2001@gmail.com

The emergence of the Internet and digital technologies has made online security a pressing issue. Authentication and authorization are key components of security systems and play a critical role in ensuring the security of data and resources on the Internet. To meet modern security standards, it is necessary to analyze and evaluate various authentication and authorization technologies. This article provides a detailed analysis of modern authentication and authorization technologies, including their advantages and disadvantages, and recommendations for their use in various areas of information security. The study results may be useful for software developers, network administrators, and anyone interested in improving security in the digital environment.

Ключові слова – автентифікація, авторизація, ASP.NET Core Identity, Firebase, AWS Cognito, Auth0, Azure Active Directory, Google Cloud Identity.

Вступ

Щодня зростає кількість кіберзагроз та ризиків безпеки в Інтернеті, аналіз технологій автентифікації та авторизації стає критично важливим завданням для забезпечення надійного захисту конфіденційної інформації та обмеження доступу до важливих ресурсів. Сучасні технологій автентифікації та авторизації пропонують

широкий функціонал для досягнення конфіденційності та цілісності інформації. Правильний підбір до потреб організації та використання переваг технологій дозволить створити безпечне середовище для функціонування вебдодатків.

Аналіз технологій автентифікації та авторизації

Сучасні технології автентифікації та авторизації повинні забезпечувати безпеку, конфіденційність та цілісність даних, а також гарантувати, що лише ідентифіковані користувачі мають доступ до системи та обмежений доступ до ресурсів всередині системи. Такими технологіями є: Firebase, AWS Cognito, Auth0, Azure Active Directory, Google Cloud Identity, ASP.NET Core Identity.

Технологія Firebase являє собою рішення бекенд як сервіс (Backend as a Service, BaaS), що надає розробникам набір інструментів, що скорочує час і важкість конфігурації та налаштування. Для автентифікації користувачів використовується JSON веб-токен (JWT), підтримує такі механізми Збережена XSS – це атака, яка передбачає введення зловмисного коду в поля форми на веб-сторінці, і зберігання введених даних в базі даних, після їх відправки [1].

автентифікації: комбінації електронної пошти/паролю та електронної пошти/посилання, дозволяє автентифікувати номер телефону та підтримує вхід із соціальних мереж. Firebase також надає можливості авторизації користувачів, а саме захищає та контролює доступ до кожної серверної служби використовуючи правила безпеки, що дозволяють використовувати мову виразів та працюють, зіставляючи шаблон із шляхами бази даних або сегментів, а потім застосовують спеціальні умови, щоб дозволити доступ для читання та запису до даних у цих шляхах [1, 2].

Технологія AWS Cognito забезпечує безпечну реєстрацію користувача та вхід. Сервіс забезпечує синхронізацію даних між пристроями користувачів. AWS Cognito включає такі функції: реєстрацію, вхід та керування обліковим записом. Технологія підтримує використання сторонніх постачальників послуг, до них відносяться соціальні мережі, відомі провайдери, а також постачальники ідентифікаційних даних корпоративного класу. AWS Cognito використовує стандартні сервіси автентифікації та авторизації, а саме Oauth 2.0, SAML 2.0 та OpenID Connect [3].

Технологія Auth0 надає інструменти для додавання потоків автентифікації та авторизації до програм, забезпечує гнучке керування ідентифікацією та параметрами автентифікації. Технологія містить такі функції: єдиний вхід (Single Sign-On, SSO), багатофакторна автентифікація, автентифікація без використання пароля (використання інших методів як біометрія, тощо), функції керування користувачам. Auth0 використовує протокол OpenID Connect (OIDC) і систему авторизації OAuth 2.0 [4, 5].

Azure Active Directory (Azure AD) надає хмарний корпоративний каталог і службу керування ідентифікацією. Технологія представляє функції, що надають користувачам безперебійний доступ до всіх типів внутрішніх та зовнішніх ресурсів, використовуючи традиційні методи автентифікації користувачів, а саме за допомогою імені користувача та пароля. Також надає можливість керування ролями та дозволами, щоб забезпечити користувачам доступ до необхідних ресурсів. Окрім базових методів автентифікації та авторизації надає можливість використання функцій багатофакторної автентифікації та SSO. Використовує протоколи OAuth 2.0 та OpenID Connect [6].

Google Cloud Identity – це рішення ідентифікації як сервіс (Identity as a Service, IDaaS), яка надає централізоване керування ідентифікаційними даними та забезпечує безпечну автентифікацію та авторизацію для програм і пристроїв. Вона підтримує використання токенів, SAML та OpenID Connect, а також надає можливості багатофакторної автентифікації, SSO та налаштування доступів до ресурсів за допомогою Cloud Identity Groups API. [7, 8].

Технологія ASP.NET Core Identity підтримує функцію входу в інтерфейс користувача; керує користувачами, паролями, даними профілю, ролями, заявками, маркерами, підтвердженням електронної пошти тощо. Користувачі можуть створити обліковий запис, використовуючи дані для входу або вони можуть використовувати зовнішні постачальники послуг входу. Фреймворк підтримує різні протоколи для автентифікації, такі як cookies, JWT (JSON Web Tokens), OAuth. Надає можливості для визначення ролей користувачів та надання їм відповідних прав доступу до функцій та ресурсів додатку [9].

В табл. 1 представлені переваги та недоліки проаналізованих технологій автентифікації та авторизації.

ТАБЛИЦЯ 1

Переваги та недоліки технологій автентифікації та авторизації

Технологія	Переваги	Недоліки
Firebase	Простота та швидкість інтеграції; різноманітність підтримуваних методів автентифікації; гнучкість налаштування.	Обмежені можливості налаштування прав доступу; можливість обмеження функціоналу під час інтеграції з іншими платформами.
AWS Cognito	Масштабованість; гнучкість; швидкість розробки; підтримка різних методів автентифікації та шифрування.	Складність налаштування; вартість; залежність від AWS.
Auth0	Простота інтеграції; масштабованість; багатофакторна автентифікація; захист від атак; готові рішення та аналітика.	Вартість; залежність від стороннього сервісу; необхідність налаштування.
Azure Active Directory	Інтеграція з іншими послугами Azure; масштабованість; гнучкість управління доступом.	Залежність від інфраструктури Microsoft; складність налаштування; вартість.
Google Cloud Identity	Простота управління; один вхід для всіх сервісів Google; інтеграція з екосистемою Google.	Залежність від інфраструктури Google; складність налаштування та інтеграції з сторонніми сервісами.
ASP.NET Core Identity	Підтримка зовнішніх постачальни-ків; використ-тання та зберігання хешованих паролів.	Залежність від платформи ASP.NET Core; вимоги до інфраструктури, складність налаштування.

Висновок

Реалізація автентифікація та авторизації за допомогою сучасних технологій надає ряд переваг: гнучкість налаштування, зручність інтеграції, підтримка різноманітних методів автентифікації та авторизації, скорочення часу розробки та безпека інформації. У роботі було проаналізовано найпопулярніші технології та представлені переваги та недоліки кожної. Кожна технологія найкраще підходить для певних типів проєктів та організацій, а саме: ASP.NET Core Identity – для інтеграції з платформою ASP.NET Core та вимагають гнучкої системи ідентифікації та авторизації з високим рівнем безпеки; Firebase – для проєктів, які активно використовують інші сервіси Google та потребують простого та швидкого рішення для автентифікації та авторизації; AWS Cognito – для організацій, що використовують інфраструктуру AWS та потребують розширеного функціоналу для управління ідентичністю та доступом; Auth0 – для проєктів, які вимагають різноманітних методів автентифікації та авторизації, включаючи соціальні мережі; Azure Active Directory – для великих підприємств, які вимагають високого рівня безпеки та гнучкості в управлінні ідентичністю та доступом; Google Cloud Identity – для проєктів, які використовують інфраструктуру Google та потребують інтегрованого рішення для автентифікації та авторизації.

Отже, вибір технології для автентифікації та авторизації повинен базуватися на специфічних потребах та вимогах організації або проекту, а також на технічних можливостях та обмеженнях кожної конкретної технології.

Література

[1] Tram M. H. B. FIREBASE : Thesis. 2019. 58 p. [Елекронний ресурс]. - https://www.theseus.fi/bitstream/handle/10024/263641/Mai_Tram.pdf?sequence=2&isAllowed=y

[2] educative.io [Елекронний ресурс]. - https://www.educative.io/answers/authentication-and-authorization-in-firebase

[3] Lysakov V., Sievierinov O., Taran I. Security of Web Applications Using AWS Cloud Provider. Computer and Information Systems and Technologies, Kharkiv. 2021. P. 35–36.

[4] frontegg.com [Елекронний ресурс]. - https://frontegg.com/guides/auth0

[5] auth0.com [Елекронний ресурс]. - https://auth0.com/docs/get-started/authentication-and-authorization-flow

[6] Zaal S. Azure Active Directory for Secure Application Development: Use Modern Authentication Techniques to Secure Applications in Azure. Packt Publishing, Limited, 2022.

[7] goldyarora.com [Елекронний ресурс]. - https://www.goldyarora.com/blog/what-is-google-cloud-identity

[8] cloud.google.com [Елекронний ресурс]. - https://cloud.google.com/identity/docs/concepts/auth

[9] learn.microsoft.com [Елекронний ресурс]. - https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual

ДОДАТОК В

Код програми

Код ApplicationDbContext.cs

```
using Auth_Model.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace Auth_Model.Services
{
   public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
   {
      public ApplicationDbContext(DbContextOptions options) : base(options)
      {
      }
      protected override void OnModelCreating(ModelBuilder builder)
      {
         base.OnModelCreating(builder);

         var admin = new IdentityRole("Admin");
         admin.NormalizedName = "Admin";

         var user = new IdentityRole("User");
         user.NormalizedName = "User";

         var premiumuser = new IdentityRole("Premium User");
         premiumuser.NormalizedName = "Premium User";
```

```
        builder.Entity<IdentityRole>().HasData(admin, user, premiumuser);
    }
  }
}
```

Код файлу Program.cs

```
using Auth_Model.Services;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;
using Auth_Model.Models;
using RazorPage.AspNetIdentity.Settings;
using SendGrid.Extensions.DependencyInjection;
using Microsoft.AspNetCore.Identity.UI.Services;
using RazorPage.AspNetIdentity.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationDbContext>(options =>
{
  var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
  options.UseSqlServer(connectionString);
});

builder.Services.AddDefaultIdentity<ApplicationUser>(options =>
options.SignIn.RequireConfirmedAccount = false)
  .AddRoles<IdentityRole>()
  .AddEntityFrameworkStores<ApplicationDbContext>();
```

```
builder.Services.Configure<SendGridSettings>(builder.Configuration.GetSection("Send
GridSettings"));

builder.Services.AddSendGrid(options => {
    options.ApiKey = builder.Configuration.GetSection("SendGridSettings")
    .GetValue<string>("ApiKey");
});

builder.Services.AddScoped<IEmailSender,EmailSenderService>();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();
```

Код файлу ApplicationUser.cs

```
using Microsoft.AspNetCore.Identity;

namespace Auth_Model.Models
{
    public class ApplicationUser : IdentityUser
    {
        public string FirstName { get; set; } = "";
        public string LastName { get; set; } = "";

    }
}
```

Код файлу _LoginPartial.cshtml

```
@using Microsoft.AspNetCore.Identity
@using Auth_Model.Models

@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager

<ul class="navbar-nav">
    @if (SignInManager.IsSignedIn(User))
    {
        if (User.IsInRole("Admin"))
        {
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle text-dark" href="#" role="button" data-
bs-toggle="dropdown" aria-expanded="true">
                    Admin
                </a>
```

```
            <ul class="dropdown-menu">
                <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Manage/Index">Profile</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Logout">Logout</a></li>
            </ul>
        </li>
    }
    else if (User.IsInRole("User"))
    {
        <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle text-dark" href="#" role="button" data-
bs-toggle="dropdown" aria-expanded="true">
                User
            </a>
            <ul class="dropdown-menu">
                <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Manage/Index">Profile</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Logout">Logout</a></li>
            </ul>
        </li>
    }
    else if (User.IsInRole("Premium User"))
    {
        <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle text-dark" href="#" role="button" data-
bs-toggle="dropdown" aria-expanded="true">
```

```
                Premium User
            </a>
            <ul class="dropdown-menu">
                <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Manage/Index">Profile</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Logout">Logout</a></li>
            </ul>
        </li>
    }
}
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" id="register" asp-area="Identity" asp-
page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" id="login" asp-area="Identity" asp-
page="/Account/Login">Login</a>
    </li>
}
</ul>
```

Код файлу _Loyout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Auth_Model</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/Auth_Model.styles.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-page="/Index">AuthModel</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Privacy</a>
                        </li>
                    </ul>
                    <partial name="_LoginPartial" />
```

```
            </div>
        </div>
    </nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2024 - AuthModel - <a asp-area="" asp-page="/Privacy">Privacy</a>
    </div>
</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Код файлу Register.cshtml

```
@page
@model RegisterModel
@{
    ViewData["Title"] = "Register";
}
```

```
<div class="row">
    <div class="col-lg-6 mx-auto rounded border p-3">
        <h2 class="text-center mb-3">Registration</h2>
        <hr />
        <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
        <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl"
method="post">
            <div class="row mb-3">
                <label class="col-sm-4 col-form-label">First Name</label>
                <div class="col-sm-8">
                    <input class="form-control" asp-for="Input.FirstName">
                    <span asp-validation-for="Input.FirstName" class="text-danger"></span>
                </div>
            </div>

            <div class="row mb-3">
                <label class="col-sm-4 col-form-label">Last Name</label>
                <div class="col-sm-8">
                    <input class="form-control" asp-for="Input.LastName">
                    <span asp-validation-for="Input.LastName" class="text-danger"></span>
                </div>
            </div>

            <div class="row mb-3">
                <label class="col-sm-4 col-form-label">Email</label>
                <div class="col-sm-8">
                    <input class="form-control" asp-for="Input.Email">
                    <span asp-validation-for="Input.Email" class="text-danger"></span>
                </div>
```

```
        </div>

        <div class="row mb-3">
            <label class="col-sm-4 col-form-label">Password</label>
            <div class="col-sm-8">
                <input class="form-control" asp-for="Input.Password">
                <span asp-validation-for="Input.Password" class="text-danger"></span>
            </div>
        </div>

        <div class="row mb-3">
            <label class="col-sm-4 col-form-label">Confirm Password</label>
            <div class="col-sm-8">
                <input class="form-control" asp-for="Input.ConfirmPassword">
                <span asp-validation-for="Input.ConfirmPassword" class="text-
danger"></span>
            </div>
        </div>

        <div class="row mb-3">
            <div class="offset-sm-4 col-sm-4 d-grid">
                <button type="submit" class="btn btn-primary">Register</button>
            </div>
            <div class="col-sm-4 d-grid">
                <a class="btn btn-outline-primary" href="/" role="button">Cancel</a>
            </div>
        </div>
    </form>
</div>
```

```
</div>
```

Код файлу Register.cshtml.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Text.Encodings.Web;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Auth_Model.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.WebUtilities;
using Microsoft.Extensions.Logging;

namespace Auth_Model.Areas.Identity.Pages.Account
{
    public class RegisterModel : PageModel
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly IUserStore<ApplicationUser> _userStore;
        private readonly IUserEmailStore<ApplicationUser> _emailStore;
        private readonly ILogger<RegisterModel> _logger;
```

```
    private readonly IEmailSender _emailSender;

    public RegisterModel(
        UserManager<ApplicationUser> userManager,
        IUserStore<ApplicationUser> userStore,
        SignInManager<ApplicationUser> signInManager,
        ILogger<RegisterModel> logger,
        IEmailSender emailSender)
    {
        _userManager = userManager;
        _userStore = userStore;
        _emailStore = GetEmailStore();
        _signInManager = signInManager;
        _logger = logger;
        _emailSender = emailSender;
    }

    [BindProperty]
    public InputModel Input { get; set; }

    public string ReturnUrl { get; set; }

    public IList<AuthenticationScheme> ExternalLogins { get; set; }

    public class InputModel
    {
        [Required]
        public string FirstName { get; set; }

        [Required]
```

```
        public string LastName { get; set; }
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
        public string ConfirmPassword { get; set; }
    }
    public async Task OnGetAsync(string returnUrl = null)
    {
        ReturnUrl = returnUrl;
        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    }
    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
        returnUrl ??= Url.Content("~/");
        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
```

```
        if (ModelState.IsValid)
        {
            var user = new ApplicationUser()
            {
                FirstName = Input.FirstName,
                LastName = Input.LastName,
                UserName = Input.Email,
                Email = Input.Email,
            };

            var result = await _userManager.CreateAsync(user, Input.Password);

            if (result.Succeeded)
            {
                _logger.LogInformation("User created a new account with password.");

                await _userManager.AddToRoleAsync(user, "User");

                var userId = await _userManager.GetUserIdAsync(user);
                var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
                code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
                var callbackUrl = Url.Page(
                    "/Account/ConfirmEmail",
                    pageHandler: null,
                    values: new { area = "Identity", userId = userId, code = code, returnUrl =
returnUrl },
                    protocol: Request.Scheme);

                await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
```

```
                $"Please confirm your account by <a
href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

            if (_userManager.Options.SignIn.RequireConfirmedAccount)
            {
                return RedirectToPage("RegisterConfirmation", new { email =
Input.Email, returnUrl = returnUrl });
            }
            else
            {
                await _signInManager.SignInAsync(user, isPersistent: false);
                return LocalRedirect(returnUrl);
            }
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }

    return Page();
}

private ApplicationUser CreateUser()
{
    try
    {
        return Activator.CreateInstance<ApplicationUser>();
    }
    catch
```

```
        {
            throw new InvalidOperationException($"Can't create an instance of
'{nameof(ApplicationUser)}'. " +
                $"Ensure that '{nameof(ApplicationUser)}' is not an abstract class and has a
parameterless constructor, or alternatively " +
                $"override the register page in
/Areas/Identity/Pages/Account/Register.cshtml");
        }
    }

    private IUserEmailStore<ApplicationUser> GetEmailStore()
    {
        if (!_userManager.SupportsUserEmail)
        {
            throw new NotSupportedException("The default UI requires a user store with
email support.");
        }
        return (IUserEmailStore<ApplicationUser>)_userStore;
    }
  }
}
```

Код файлу Login.cshtml

```
@page
@model LoginModel

@{
  ViewData["Title"] = "Log in";
}
```

```
<div class="row">
    <div class="col-md-4 mx-auto rounded border p-3">
        <section>
            <h2 class="text-center mb-3">Please Log In</h2>
            <hr />
            <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>

            <form id="account" method="post">

                <div class="mb-3">
                    <label class="form-label">Email</label>
                    <input asp-for="Input.Email" class="form-control" />
                    <span asp-validation-for="Input.Email" class="text-danger"></span>
                </div>
                <div class="mb-3">
                    <label class="form-label">Password</label>
                    <input asp-for="Input.Password" class="form-control" />
                    <span asp-validation-for="Input.Password" class="text-danger"></span>
                </div>
                <div class="checkbox mb-3">
                    <label asp-for="Input.RememberMe" class="form-label">
                        <input class="form-check-input" asp-for="Input.RememberMe" />
                        @Html.DisplayNameFor(m => m.Input.RememberMe)
                    </label>
                </div>
                <div class="row mb-3">
                    <div class="offset-sm-4 col-sm-4 d-grid">
                        <button type="submit" class="btn btn-primary">Log in</button>
                    </div>
```

```
                <div class="col-sm-4 d-grid">
                    <a class="btn btn-outline-primary" href="/" role="button">Cancel</a>
                </div>
            </div>
            <div>
                <a class="btn btn-link" id="forgot-password" asp-
page="./ForgotPassword">Forgot your password?</a>
            </div>
        </form>
    </section>
  </div>
</div>
```

Код файлу Logout.cshtml.cs

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Auth_Model.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace Auth_Model.Areas.Identity.Pages.Account
{
  public class LogoutModel : PageModel
  {
    private readonly SignInManager<ApplicationUser> _signInManager;
    private readonly ILogger<LogoutModel> _logger;
```

```
        public LogoutModel(SignInManager<ApplicationUser> signInManager,
ILogger<LogoutModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        public async Task<IActionResult> OnGet()
        {
            await _signInManager.SignOutAsync();
            return LocalRedirect("/");
        }

        public async Task<IActionResult> OnPost()
        {
            await _signInManager.SignOutAsync();
            return LocalRedirect("/");
        }
    }
}
```

Код файлу User.cshtml

```
@page
@model Auth_Model.Pages.UserModel
@{
}

<h2 class="text-center">Welcome User!</h2>
<br />
<h3>You have been successfully authorized!</h3>
```

Код файлу User.cshtml.cs

```
using Auth_Model.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Auth_Model.Pages
{
    [Authorize(Roles = "User")]
    public class UserModel : PageModel
    {
        private readonly UserManager<ApplicationUser> userManager;

        public ApplicationUser? appUser;

        public UserModel(UserManager<ApplicationUser> userManager)
        {
            this.userManager = userManager;
        }
        public void OnGet()
        {
            var task = userManager.GetUserAsync(User);
            task.Wait();
            appUser = task.Result;
        }
    }
}
```

Код файлу Admin.cshtml

```
@page
@model Auth_Model.Pages.AdminModel
@{
}

<h2 class="text-center">Welcome Admin!</h2>
<br />
<h3>You have been successfully authorized!</h3>
```

Код файлу Admin.cshtml.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Auth_Model.Pages
{
    [Authorize(Roles = "Admin")]
    public class AdminModel : PageModel
    {
        public void OnGet()
        {
        }
    }
}
```

Код файлу PremiumUser.cshtml

```
@page
@model Auth_Model.Pages.PremiumUserModel
```

```
@{
}

<h2 class="text-center">Welcome Premium User!</h2>
<br />
<h3>You have been successfully authorized!</h3>
```

Код файлу PremiumUser.cshtml.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Auth_Model.Pages
{
    [Authorize(Roles ="Premium User")]
    public class PremiumUserModel : PageModel
    {
        public void OnGet()
        {
        }
    }
}
```

Код файлу EnableAuthenticator.cshtml

```
@page
@model EnableAuthenticatorModel
@{
    ViewData["Title"] = "Configure authenticator app";
    ViewData["ActivePage"] = ManageNavPages.TwoFactorAuthentication;
```

```
}

<partial name="_StatusMessage" for="StatusMessage" />
<h3>@ViewData["Title"]</h3>
<div>
    <p>To use an authenticator app go through the following steps:</p>
    <ol class="list">
        <li>
            <p>
                Download a two-factor authenticator app like Microsoft Authenticator for
                <a href="https://go.microsoft.com/fwlink/?Linkid=825072">Android</a> and
                <a href="https://go.microsoft.com/fwlink/?Linkid=825073">iOS</a> or
                Google Authenticator for
                <a
href="https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&amp;hl=en">Android</a> and
                <a href="https://itunes.apple.com/us/app/google-authenticator/id388497605?mt=8">iOS</a>.
            </p>
        </li>
        <li>
            <p>Scan the QR Code or enter this key <kbd>@Model.SharedKey</kbd> into your two factor authenticator app. Spaces and casing do not matter.</p>
            <div id="qrCode"></div>
            <div id="qrCodeData" data-url="@Model.AuthenticatorUri"></div>
        </li>
        <li>
            <p>
                Once you have scanned the QR code or input the key above, your two factor authentication app will provide you
```

```
            with a unique code. Enter the code in the confirmation box below.
        </p>
        <div class="row">
            <div class="col-md-6">
                <form id="send-code" method="post">
                    <div class="form-floating mb-3">
                        <input asp-for="Input.Code" class="form-control" autocomplete="off"
placeholder="Please enter the code."/>
                        <label asp-for="Input.Code" class="control-label form-
label">Verification Code</label>
                        <span asp-validation-for="Input.Code" class="text-danger"></span>
                    </div>
                    <button type="submit" class="w-100 btn btn-lg btn-
primary">Verify</button>
                    <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
                </form>
            </div>
        </div>
    </li>
</ol>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
    <script type="text/javascript" src="~/lib/qrcode.min.js"></script>
    <script type="text/javascript" src="~/js/qr.js"></script>
}
```

Код файлу EnableAuthenticator.cshtml.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Text.Encodings.Web;
using System.Threading.Tasks;
using Auth_Model.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace Auth_Model.Areas.Identity.Pages.Account.Manage
{
    public class EnableAuthenticatorModel : PageModel
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly ILogger<EnableAuthenticatorModel> _logger;
        private readonly UrlEncoder _urlEncoder;

        private const string AuthenticatorUriFormat =
"otpauth://totp/{0}:{1}?secret={2}&issuer={0}&digits=6";

        public EnableAuthenticatorModel(
            UserManager<ApplicationUser> userManager,
            ILogger<EnableAuthenticatorModel> logger,
            UrlEncoder urlEncoder)
        {
```

```
        _userManager = userManager;
        _logger = logger;
        _urlEncoder = urlEncoder;
    }

    public string SharedKey { get; set; }
    public string AuthenticatorUri { get; set; }

    [TempData]
    public string[] RecoveryCodes { get; set; }

    [TempData]
    public string StatusMessage { get; set; }

    [BindProperty]
    public InputModel Input { get; set; }

    public class InputModel
    {
        [Required]
        [StringLength(7, ErrorMessage = "The {0} must be at least {2} and at max {1}
characters long.", MinimumLength = 6)]
        [DataType(DataType.Text)]
        [Display(Name = "Verification Code")]
        public string Code { get; set; }
    }

    public async Task<IActionResult> OnGetAsync()
    {
        var user = await _userManager.GetUserAsync(User);
```

```
            if (user == null)
            {
                return NotFound($"Unable to load user with ID
'{_userManager.GetUserId(User)}'.");
            }

            await LoadSharedKeyAndQrCodeUriAsync(user);

            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            var user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
                return NotFound($"Unable to load user with ID
'{_userManager.GetUserId(User)}'.");
            }

            if (!ModelState.IsValid)
            {
                await LoadSharedKeyAndQrCodeUriAsync(user);
                return Page();
            }


            var verificationCode = Input.Code.Replace(" ", string.Empty).Replace("-",
string.Empty);
```

```
        var is2faTokenValid = await _userManager.VerifyTwoFactorTokenAsync(
            user, _userManager.Options.Tokens.AuthenticatorTokenProvider,
verificationCode);

        if (!is2faTokenValid)
        {
            ModelState.AddModelError("Input.Code", "Verification code is invalid.");
            await LoadSharedKeyAndQrCodeUriAsync(user);
            return Page();
        }

        await _userManager.SetTwoFactorEnabledAsync(user, true);
        var userId = await _userManager.GetUserIdAsync(user);
        _logger.LogInformation("User with ID '{UserId}' has enabled 2FA with an
authenticator app.", userId);

        StatusMessage = "Your authenticator app has been verified.";

        if (await _userManager.CountRecoveryCodesAsync(user) == 0)
        {
            var recoveryCodes = await
_userManager.GenerateNewTwoFactorRecoveryCodesAsync(user, 10);
            RecoveryCodes = recoveryCodes.ToArray();
            return RedirectToPage("./ShowRecoveryCodes");
        }
        else
        {
            return RedirectToPage("./TwoFactorAuthentication");
        }
    }
```

```
private async Task LoadSharedKeyAndQrCodeUriAsync(ApplicationUser user)
{
    var unformattedKey = await _userManager.GetAuthenticatorKeyAsync(user);
    if (string.IsNullOrEmpty(unformattedKey))
    {
        await _userManager.ResetAuthenticatorKeyAsync(user);
        unformattedKey = await _userManager.GetAuthenticatorKeyAsync(user);
    }

    SharedKey = FormatKey(unformattedKey);

    var email = await _userManager.GetEmailAsync(user);
    AuthenticatorUri = GenerateQrCodeUri(email, unformattedKey);
}

private string FormatKey(string unformattedKey)
{
    var result = new StringBuilder();
    int currentPosition = 0;
    while (currentPosition + 4 < unformattedKey.Length)
    {
        result.Append(unformattedKey.AsSpan(currentPosition, 4)).Append(' ');
        currentPosition += 4;
    }
    if (currentPosition < unformattedKey.Length)
    {
        result.Append(unformattedKey.AsSpan(currentPosition));
    }
```

```
            return result.ToString().ToLowerInvariant();
        }

        private string GenerateQrCodeUri(string email, string unformattedKey)
        {
            return string.Format(
                CultureInfo.InvariantCulture,
                AuthenticatorUriFormat,
                _urlEncoder.Encode("2FA with TOTP"),
                _urlEncoder.Encode(email),
                unformattedKey);
        }
    }
}
```

Код файлу qr.js

```
window.addEventListener("load", () => {
    const uri = document.getElementById("qrCodeData").getAttribute('data-url');
    new QRCode(document.getElementById("qrCode"),
        {
            text: uri,
            width: 150,
            height: 150
        });
});
```

Код файлу appsettings.Development.json

```
  "DetailedErrors": true,
```

```
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "SendGridSettings":{
    "FromEmail": "katev.auth.model@gmail.com",
    "EmailName": "AuthModel",
    "ApiKey": "ApiKye from SendGrid"
  }
}
```

Код файлу SendGridSettings.cs

```
namespace RazorPage.AspNetIdentity.Settings;


public class SendGridSettings
{
    public string ApiKey { get; set; }
    public string FromEmail { get; set; }
    public string EmailName { get; set; }
}
```

Код файлу EmailSenderService.cs

```
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using RazorPage.AspNetIdentity.Settings;
using SendGrid;
using SendGrid.Helpers.Mail;
```

```
namespace RazorPage.AspNetIdentity.Services;

public class EmailSenderService : IEmailSender
{
   private readonly ISendGridClient  _sendGridClient;
   private readonly SendGridSettings _sendGridSettings;
   public EmailSenderService(ISendGridClient sendGridClient,
   IOptions<SendGridSettings> sendGridSettings)
   {
      _sendGridClient = sendGridClient;
      _sendGridSettings = sendGridSettings.Value;
   }
   public async Task SendEmailAsync(string email, string subject, string htmlMessage)
   {
      var msg = new SendGridMessage()
      {
         From = new EmailAddress(_sendGridSettings.FromEmail,
_sendGridSettings.EmailName),
         Subject = subject,
         HtmlContent = htmlMessage
      };
           msg.AddTo(email);
           await _sendGridClient.SendEmailAsync(msg);
   }
}
```

Код файлу LoginWith2fa.cshtml.cs

```
using System;
using System.ComponentModel.DataAnnotations;
```

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using Auth_Model.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;
using Microsoft.AspNetCore.Identity.UI.Services;

namespace Auth_Model.Areas.Identity.Pages.Account
{
    public class LoginWith2faModel : PageModel
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly ILogger<LoginWith2faModel> _logger;
        private readonly IEmailSender _emailSender;

        public LoginWith2faModel(
            SignInManager<ApplicationUser> signInManager,
            UserManager<ApplicationUser> userManager,
            ILogger<LoginWith2faModel> logger,
            IEmailSender emailSender)
        {
            _signInManager = signInManager;
            _userManager = userManager;
            _logger = logger;
            _emailSender = emailSender;
        }
```

```
        [BindProperty]
        public InputModel Input { get; set; }

        public bool RememberMe { get; set; }
        public string ReturnUrl { get; set; }

        public class InputModel
        {
            [Required]
            [StringLength(7, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]
            [DataType(DataType.Text)]
            [Display(Name = "Authenticator code")]
            public string TwoFactorCode { get; set; }

            [Display(Name = "Remember this machine")]
            public bool RememberMachine { get; set; }
        }

        public async Task<IActionResult> OnGetAsync(bool rememberMe, string returnUrl = null)
        {

            var user = await _signInManager.GetTwoFactorAuthenticationUserAsync();

            if (user == null)
            {
                throw new InvalidOperationException($"Unable to load two-factor authentication user.");
            }
```

```
        var providers =await _userManager.GetValidTwoFactorProvidersAsync(user);

        if(providers.Any(_ => _ == "Email"))
        {
            var token = await _userManager.GenerateTwoFactorTokenAsync(user,
"Email");

            await _emailSender.SendEmailAsync(user.Email, "2FA Code",
            $"<h1>{token}</h1>");
        }
        ReturnUrl = returnUrl;
        RememberMe = rememberMe;

        return Page();
    }

    public async Task<IActionResult> OnPostAsync(bool rememberMe, string
returnUrl = null)
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        returnUrl = returnUrl ?? Url.Content("~/");

        var user = await _signInManager.GetTwoFactorAuthenticationUserAsync();
        if (user == null)
        {
```

```
            throw new InvalidOperationException($"Unable to load two-factor
authentication user.");
        }

        var authenticatorCode = Input.TwoFactorCode.Replace(" ",
string.Empty).Replace("-", string.Empty);

        //var result = await
_signInManager.TwoFactorAuthenticatorSignInAsync(authenticatorCode,
rememberMe, Input.RememberMachine);

        var result = await _signInManager.TwoFactorSignInAsync("Email",
authenticatorCode, rememberMe, Input.RememberMachine);
        var userId = await _userManager.GetUserIdAsync(user);

        if (result.Succeeded)
        {
            _logger.LogInformation("User with ID '{UserId}' logged in with 2fa.",
user.Id);
            return LocalRedirect(returnUrl);
        }
        else if (result.IsLockedOut)
        {
            _logger.LogWarning("User with ID '{UserId}' account locked out.", user.Id);
            return RedirectToPage("./Lockout");
        }
        else
        {
            _logger.LogWarning("Invalid authenticator code entered for user with ID
'{UserId}'.", user.Id);
```

```
            ModelState.AddModelError(string.Empty, "Invalid authenticator code.");
            return Page();
        }
    }
  }
}
```