Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему Розробка системи авторизації на основі ASP.NET Core Identity

Виконав: студент групи ІПЗ20-2

Спеціальність 121 «Інженерія Програмного Забезпечення»

Бєлінський Андрій Васильович

(прізвище та ініціали)

Керівник кфмн. Фірсов О.Д.

(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Бєлінський А.В. Розробка системи авторизації на основі ASP.NET Core Identity. Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справи та фінансів, Дніпро, 2024.

Дана дипломна робота створена для розробки системи авторизації для веб-сайту. Розглянуті методи та технічні засоби, необхідні для реалізації системи авторизації, сформульовані вимоги до програмного забезпечення. Було проведено проектування та розробку системи авторизації, а також її тестування. Отримані результати мають практичне значення, оскільки сприяють підвищенню безпеки та захисту даних, забезпечуючи контроль доступу до критичних функцій системи та конфіденційної інформації.

Розроблена система є надійною та ефективною, використовує передові технології, такі як багатофакторна аутентифікація та політики доступу, і є важливим кроком у покращенні управління безпекою у різних галузях.

Розроблена система авторизації має серверну та клієнтську частини. Система забезпечує перевірку облікових даних користувачів та контроль доступу до різних рівнів функціональності. Згідно з налаштуваннями ролей та політиками доступу, система надає відповідні права користувачам. Цей додаток має сучасний та зручний дизайн, який сприяє зручності користування та ефективному управлінню безпекою.

Ключові слова: авторизація, безпека, багатофакторна аутентифікація, політики доступу, ASP.NET Core Identity.

ABSTRACT

Belinsky A.V. Development of an Authorization System Based on ASP.NET Core Identity. Qualification work for obtaining a bachelor's degree in specialty 121 "Software Engineering". - University of Customs and Finance, Dnipro, 2024.

This diploma work is created for the development of an authorization system for a website. The methods and technical means necessary for the implementation of the authorization system were considered, and the software requirements were formulated. The design and development of the authorization system were carried out, as well as its testing. The obtained results are of practical importance as they contribute to enhancing security and data protection, ensuring access control to critical system functions and confidential information.

The developed system is reliable and efficient, using advanced technologies such as multi-factor authentication and access policies, and represents an important step in improving security management in various industries.

The developed authorization system has server-side and client-side components. The system ensures the verification of user credentials and access control to different levels of functionality. According to role settings and access policies, the system grants appropriate rights to users. This application features a modern and user-friendly design, which facilitates ease of use and effective security management.

Keywords: authorization, security, multi-factor authentication, access policies, ASP.NET Core Identity.

ЗМІСТ

ВСТУП

Тема цієї дипломної роботи є актуальною і важливою в сучасному світі, де інформаційні технології відіграють ключову роль. Система авторизації на ASP.NET Core з використанням сервісів Google, Microsoft, GitHub та JWT токенів є важливим елементом безпеки даних і користувацького досвіду.

Оскільки цифровий світ продовжує розширюватися, забезпечення безпеки веб-додатків і захист даних користувачів стає більш важливим, ніж будь-коли. Одним з ключових аспектів безпеки веб-розробки є автентифікація, яка передбачає перевірку особи користувачів, що отримують доступ до веб-додатків.

Для реалізації задачі буде використано ASP.NET Core - це є однією з передових технологій у сфері веб-розробки, яка пропонує широкий спектр можливостей та переваг для розробників. Обрання ASP.NET Core для розробки веб-додатків обґрунтовується кількома ключовими аспектами, які включають в себе технічні переваги, продуктивність, масштабованість, підтримку спільноти та безпеку.

Мета цієї роботи полягає в розробці надійної і ефективної системи авторизації, щоб допомогти убезпечити роботу в інтернеті та ефективно захистити дані користувачів, яка б використовувала передові технології та сервіси. Це завдання має велике значення для предметної області, оскільки воно допоможе підвищити рівень безпеки та зручності для користувачів.

Методи дослідження включають теорію інформації, обробку даних, проектування програмного забезпечення, проектування баз даних.

У відповідності до поставленої мети в кваліфікаційній роботі ставились та вирішувались наступні завдання дослідження:

1. Проаналізувати методи та технічні засоби, що застосовуються для розробки систем авторизації, та обрати необхідні для реалізації технології.
2. Розробити вимоги до програмного забезпечення для системи авторизації, враховуючи особливості безпеки, ефективності та зручності для користувачів.

3. Спроектувати та реалізувати нову систему авторизації на основі аналізу потреб користувачів та вимог безпеки.

4. Провести тестування системи для перевірки її працездатності, надійності та безпеки.

Об’єктом дослідження є система авторизації.

Предметом дослідження є апаратно-програмний комплекс для реалізації системи авторизації.

Практичне значення одержаних результатів – підвищення безпеки та контролю доступу до веб-сайту.

Структура роботи:

Розділ 1 Теоретичні основи. В даному розділі буде проведено теоретичний аналіз основних концепцій, пов'язаних з аутентифікацією та авторизацією, а також з веб-розробкою на платформі ASP.NET Core.

Розділ 2 Аналіз вимог. В даному розділі будуть визначені критерії вибору до системи авторизації та описаний процес авторизації.

Розділ 3 Проектування системи авторизації. В даному розділі буде проведено розробка архітектури системи, опис інтерфейсу програми та процесу авторизації. Опис процес вибору технологій та інструментів.

Розділ 4. Реалізація системи авторизації. В даному розділі буде докладно описано процес реалізації системи авторизації, включаючи розробку програмного забезпечення та його інтеграцію з базою даних.

Розділ 5. Тестування та аналіз безпеки. В даному розділі буде здійснено тестування системи авторизації, а також проведено аналіз вразливостей для виявлення потенційних проблем безпеки. Після цього проведена оцінка ефективності системи з урахуванням результатів тестування та аналізу вразливостей.

Робота складається зі вступу, 5-х розділів, висновків, списку використаних джерел з 25 найменувань. Обсяг роботи 69 сторінок, 42 рисунків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ

1.1 Основи аутентифікації та авторизації

Аутентифікація – це процес верифікації особи або системи, що намагається отримати доступ до певних ресурсів. Це робиться завдяки запиту до користувача для того, щоб отримати дані, наприклад ім'я користувача або пароль, для того щоб підтвердити тим ким, за кого він себе видає.

Авторизація, з другого боку, визначає, які ресурси або функції може використовувати користувач після успішної аутентифікації. Цей процес базується на встановленні прав доступу до конкретних ресурсів для різних категорій користувачів.

Також є різні стратегії авторизації, як наприклад сесійна автентифікація є однією з найпоширеніших стратегій автентифікації і довгий час була способом автентифікації за замовчуванням. Вона передбачає використання файлів cookie для зберігання інформації про сеанс на стороні клієнта.

Коли користувач входить в систему, сервер створює сеанс і зберігає його в базі даних. Це означає, що з'єднання має певний стан. Потім сервер надсилає ідентифікатор сеансу клієнту у вигляді файлу cookie.

Коли користувач робить запит до сервера, цей ідентифікатор сесії включається в запит, потім сервер перевіряє ідентифікатор сесії в файлі cookie і надає доступ до програми, якщо сесія дійсна. Якщо ні, користувачеві/клієнту відмовляють у доступі і, можливо, перенаправляють на головну сторінку/сторінку входу.

Також існує інша стратегія авторизації - Автентифікація на основі токенів. Вона дуже схожа на сеансову автентифікацію, але замість того, щоб зберігати інформацію про сеанс в базі даних, сервер генерує унікальний токен JWT, який надсилається клієнту і зберігається в файлі cookie. Це означає, що з'єднання не має статусу, і нам не потрібно зберігати інформацію про сеанс в базі даних.

Токен JWT - це рядок, який містить об'єкт JSON, закодований за допомогою Base64. Він підписується за допомогою секретного ключа, який використовується для перевірки токена.

Коли користувач робить запит до сервера, токен включається в запит в заголовок Authorization з префіксом «Bearer », потім сервер перевіряє токен і надає доступ до додатку, якщо токен дійсний. Якщо ні, користувачеві/клієнту відмовляють у доступі і, можливо, перенаправляють на головну сторінку/сторінку входу.

Аутентифікація та авторизація є ключовими аспектами забезпечення безпеки веб-сайтів, вони дозволяють ідентифікувати користувачів та контролювати їх доступ до ресурсів сайту з метою захисту конфіденційності та цілісності даних.[4]

1.2 Основи веб-розробки на ASP.NET Core

ASP.NET Core - це платформа веб-розробки, що базується на відкритому вихідному коді та розроблена корпорацією Microsoft для створення модерних та масштабованих веб-додатків і служб. Ця платформа втілює в собі ряд принципів роботи, які сприяють її ефективності, розширюваності та стабільності.[23]

Більшість традиційних .NET-додатків розгортаються як окремі одиниці, що відповідають одному виконуваному файлу або одному веб-додатку, запущеному в одному домені додатків IIS. Такий підхід є найпростішою моделлю розгортання і дуже добре підходить для багатьох внутрішніх і невеликих загальнодоступних додатків. Однак, навіть враховуючи цю єдину одиницю розгортання, більшість нетривіальних бізнес-додатків виграють від деякого логічного розділення на кілька рівнів.

Найменша можлива кількість проектів для архітектури додатку - один. У цій архітектурі вся логіка програми міститься в одному проекті, компілюється в єдину збірку і розгортається як єдине ціле.

Новий проект ASP.NET Core, створений у Visual Studio або з командного рядка, починається як простий моноліт «все в одному». Він містить усю поведінку програми,

включно з презентацією, бізнес-логікою та логікою доступу до даних. На рисунку 1.1 показано файлову структуру монолітної програми.

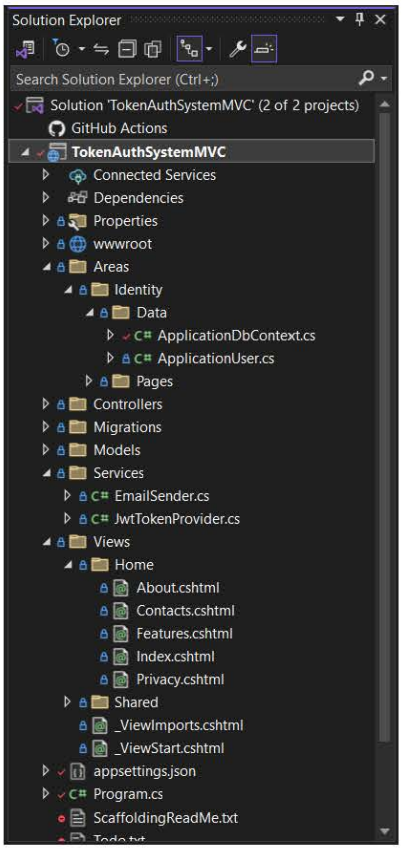


Рисунок 1.1 - Структура монолітної програми

В одному проектному сценарії розділення завдань досягається за допомогою папок. Шаблон за замовчуванням включає окремі папки для моделей, представлень і контролерів, а також додаткові папки для даних і сервісів у шаблоні MVC. При такому розташуванні деталі представлення повинні бути максимально обмежені папкою Views, а деталі реалізації доступу до даних повинні бути обмежені класами, що зберігаються в папці Data. Бізнес-логіка повинна міститися у сервісах і класах у папці Models.[21]

Використання такої структури, дозволяє розробникам легко орієнтуватися в кодовій базі та підтримувати чисту архітектуру додатка.

1.3. Висновки до першого розділу

Аутентифікація визначається як процес перевірки особи або системи, що намагається отримати доступ до ресурсів, через запит ідентифікаційних даних. Авторизація, з іншого боку, встановлює, які ресурси або функції може використовувати користувач після успішної аутентифікації. У тексті також описані дві стратегії авторизації: сесійна автентифікація, що базується на використанні файлів cookie для зберігання інформації про сеанс, та автентифікація на основі токенів, де сервер генерує унікальний токен JWT для забезпечення доступу. Обидві стратегії мають свої переваги та застосування в залежності від потреб системи. У кінці тексту наголошується на тому, що аутентифікація та авторизація є критичними для забезпечення безпеки веб-сайтів, оскільки вони дозволяють ідентифікувати користувачів та контролювати їх доступ до ресурсів з метою забезпечення конфіденційності та цілісності даних.

Платформа дотримується ряду принципів, що сприяють її ефективності, розширюваності та стабільності. Текст розглядає різні аспекти архітектури додатків на базі ASP.NET Core, зокрема, підхід "все в одному" та монолітний підхід. Він наголошує на тому, що найменша можлива кількість проектів для архітектури додатку - один, і описує структуру монолітної програми, яка включає у себе усю поведінку програми, включно з презентацією, бізнес-логікою та логікою доступу до даних. Крім того, текст розглядає підходи до розділення завдань в монолітній програмі за допомогою папок, що дозволяє розробникам легко орієнтуватися в кодовій базі та підтримувати чисту архітектуру додатка.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ

2.1 Переваги та недоліки існуючих рішень

Системи авторизації відіграють ключову роль у забезпеченні безпеки інформаційних систем, дозволяючи контролювати доступ до ресурсів на основі ідентифікації користувача.

RBAC (Role-Based Access Control) - є однією з найпоширеніших моделей, що базується на ролях користувачів у системі. Користувачам призначаються одна або кілька ролей, а кожна роль має набір дозволів на доступ до різних ресурсів. Переваги цієї моделі включають: гнучкість (легко управляти доступом до ресурсів через ролі), зручність адміністрування (адміністраторам потрібно керувати ролями, а не окремими дозволами для кожного користувача.)

ABAC (Attribute-Based Access Control) використовує атрибути (властивості) користувачів, ресурсів і середовища для прийняття рішень про доступ. Наприклад, атрибутами можуть бути вік користувача, місцезнаходження, час доби тощо.

PBAC (Policy-Based Access Control) дозволяє створювати складні політики доступу, що базуються на поєднанні різних умов і атрибутів. Це дозволяє детально налаштовувати авторизацію відповідно до потреб організації.

Модель CBAC (Context-based access control) враховує контекстні фактори, такі як місце, час, пристрій і мережеве середовище, щоб приймати рішення про доступ. Наприклад, доступ може бути дозволений тільки з певних місць або пристроїв.

Кожна з розглянутих систем авторизації має свої переваги та недоліки, на основі цього можна створити таблицю (таблиця 2.1).

Таблиця 2.1

Порівняння моделей авторизації

Модель	Переваги	Недоліки
RBAC	Простота, легке управління	Обмежена гнучкість

ABAC	Висока гнучкість, динамічність	Складність налаштування
MFA	Висока безпека	Збільшена складність для користувачів
PBAC	Детальний контроль, гнучкість	Складність адміністрування

2.2 Сучасні підходи до авторизації

Сучасні підходи до авторизації у веб-додатках значно змінилися з появою нових стандартів та технологій, які забезпечують підвищену безпеку, гнучкість і зручність у використанні.

Основні підходи включають OAuth 2.0, який є широко використовуваним протоколом авторизації, що дозволяє стороннім додаткам отримувати обмежений доступ до ресурсів користувача без необхідності передавати його облікові дані, з такими компонентами як Authorization Server, Resource Server, Client та Resource Owner. OpenID Connect (OIDC) є розширенням OAuth 2.0, що додає функції ідентифікації користувачів, дозволяючи додаткам отримувати інформацію про автентифікованого користувача та перевіряти його особу за допомогою ID Token та UserInfo Endpoint.

В цій роботі буде використовуватися метод авторизації через Google - схема роботи якого, відображена на рисунку 2.1 .

Рисунок 2.1 - Процес автентифікації користувача через Google

JSON Web Tokens (JWT) - це компактний, URL-безпечний спосіб представлення вимог між двома сторонами, широко використовуваний для передачі інформації про авторизацію та автентифікацію через компоненти Header, Payload та Signature (рисунок 2.2).

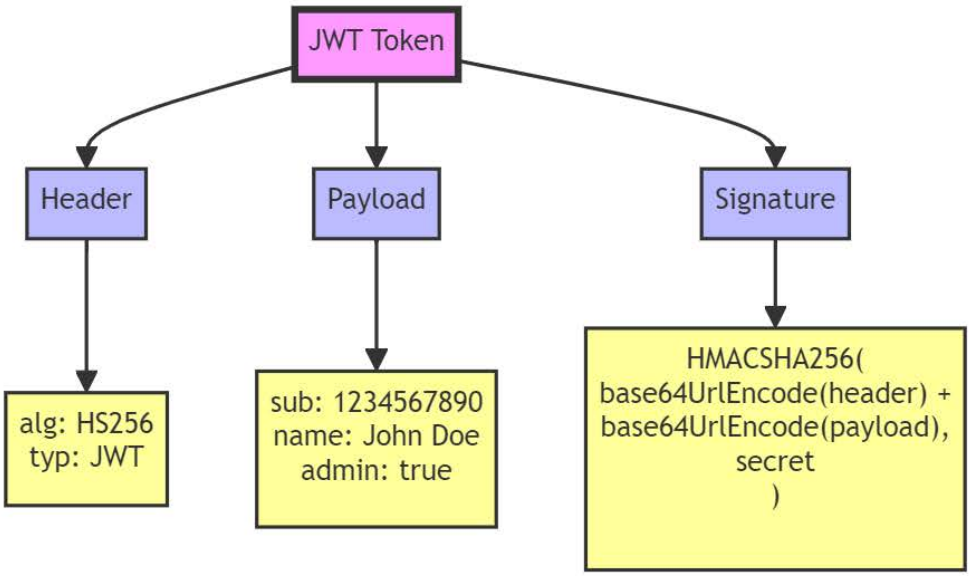


Рисунок 2.2 – Будова JWT токену

Заголовок (Header) містить метадані про тип даних і алгоритм, який буде використано для шифрування даних, що передаються. Заголовок JWT складається з трьох секцій - це метадані для токена, тип підпису та алгоритм шифрування. Він складається з двох властивостей, тобто "alg" і "typ". Хоча перша відноситься до використовуваного криптографічного алгоритму, тобто в даному випадку HS256, друга використовується для вказівки типу токена, тобто в даному випадку JWT. Корисне навантаження (Payload) зазвичай містить твердження, ідентифікаційні дані користувача, дозволені дозволи тощо. Ви можете використовувати твердження для передачі додаткової інформації. Вони також називаються JWT-запитами і бувають двох типів: Зарезервовані та Користувацькі. Ось список зарезервованих запитів: Підпис (Signature)

відповідає специфікації JSON Web Signature (JWS) і використовується для перевірки цілісності даних, що передаються каналами зв'язку. Він складається з хешу заголовка, корисного навантаження і секрету, і використовується для того, щоб переконатися, що повідомлення не було змінено під час передачі. Остаточний підписаний токен створюється відповідно до специфікації JSON Web Signature (JWS). Закодований JWT-заголовок і закодований JWT-вміст об'єднуються, а потім підписуються за допомогою стійкого алгоритму шифрування, такого як HMAC SHA 256.

Користувач надає свої облікові дані, такі як ім'я користувача та пароль, для входу на веб-сайт або додаток. Система перевіряє, чи є надані облікові дані коректними та валідними. Якщо дані правильні, користувач отримує доступ.

Після успішної аутентифікації сервер створює JWT токен. Цей токен містить інформацію про користувача та встановлює строк дії токена. Відправлення токена клієнту: JWT токен передається клієнту, як правило, у відповіді на успішний запит на аутентифікацію. Клієнт зберігає отриманий JWT токен. Зазвичай він зберігається в локальному сховищі, такому як локальне сховище браузера або пам'ять пристрою.

Використання токена при доступі до ресурсів: Кожен раз, коли клієнт намагається отримати доступ до захищених ресурсів на сервері, він включає JWT токен у заголовку запиту. Сервер перевіряє токен на валідність та авторизує користувача на основі інформації, збереженої в токені.

Оновлення токена (опціонально): Якщо токен має обмежений строк дії і він ще не закінчився, сервер може оновити токен, видавши новий токен з поновленим строком дії.

Вихід з системи або завершення сесії: При виході з системи або завершенні сесії токен видаляється зберігається на клієнтському пристрої.

Багатофакторна автентифікація (MultiFactor Authentication) (рисунок 2.3) покращує безпеку авторизації, вимагаючи від користувачів підтвердження своєї особи за допомогою кількох методів автентифікації, таких як паролі, мобільні додатки, біометричні дані тощо.

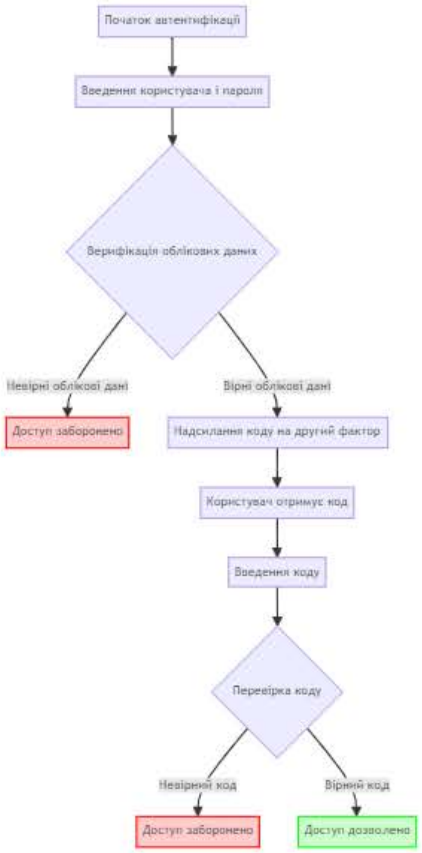


Рисунок 2.3 – Багатофакторна автентифікація

Технологія єдиного входу (Single Sign-On) (рисунок 2.4) дозволяє користувачам автентифікуватися один раз і отримувати доступ до кількох додатків без необхідності повторної автентифікації, спрощуючи користувацький досвід і зменшуючи кількість облікових даних, які потрібно запам'ятовувати.

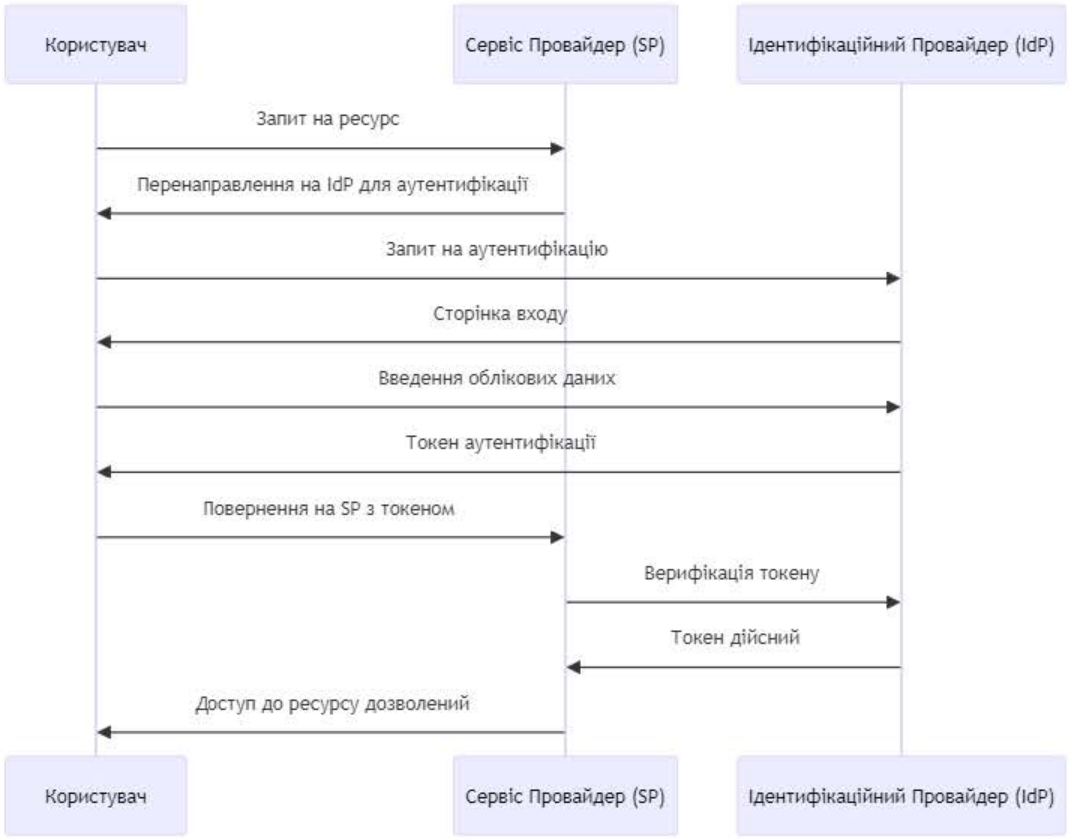


Рисунок 2.4 – Схема роботи SSO

Ці сучасні підходи до авторизації забезпечують баланс між зручністю використання та високим рівнем безпеки, відповідаючи вимогам сучасних веб-додатків.

2.3 Критерії вибору системи авторизації

Вибір системи авторизації для ASP.NET Core залежить від кількох ключових критеріїв, які включають функціональні вимоги, безпеку, продуктивність, масштабованість, зручність використання та інтеграцію з іншими системами.

Однією з ключових функціональних вимог є забезпечення механізмів аутентифікації. Це включає підтримку різноманітних протоколів аутентифікації, таких як OAuth2, OpenID Connect, SAML, і LDAP, що забезпечує гнучкість інтеграції з різними зовнішніми та внутрішніми системами аутентифікації. Додатково, багатофакторна

аутентифікація (MFA) повинна забезпечувати додатковий рівень захисту через SMS, електронну пошту або апаратні токени

Управління доступом є ще одним критичним аспектом. Система повинна підтримувати створення та управління ролями користувачів, а також детальне налаштування прав доступу для кожної ролі. Важливо також мати можливість визначення ієрархічних та динамічних політик доступу, які можуть автоматично оновлюватися на основі певних умов.

Інтеграція з іншими системами також є важливим функціональним вимогою. Це включає підтримку сервісів соціальної аутентифікації, таких як Google, Facebook, Twitter, для спрощення процесу аутентифікації користувачів, а також інтеграцію з корпоративними службами, такими як Microsoft Active Directory та Azure AD, що забезпечує єдиний вхід для співробітників.

Крім того, аудит та логування є важливими функціональними аспектами. Система повинна забезпечувати детальне логування подій аутентифікації та авторизації, включаючи спроби входу, зміни паролів та зміну прав доступу. Також важливо мати механізми моніторингу безпеки для виявлення та попередження підозрілих дій.

Однією з ключових нефункціональних вимог є безпека. Це включає захист від поширених атак, таких як SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), а також використання сучасних методів шифрування для захисту даних під час передачі та зберігання, зокрема TLS для передачі даних і AES для зберігання чутливих даних.

Продуктивність є ще одним важливим нефункціональним аспектом. Система повинна бути здатна масштабуватися горизонтально та вертикально для обробки збільшення кількості користувачів та запитів, а також забезпечувати оптимізацію швидкодії з мінімальним впливом на час відгуку додатка під час виконання аутентифікації та авторизації.

Зручність використання також є важливим нефункціональним аспектом. Система повинна мати детальну документацію та приклади коду для спрощення процесу

інтеграції системи авторизації з іншими компонентами додатка, а також забезпечувати гнучкість налаштування для легкої конфігурації та кастомізації під специфічні потреби організації.

Нарешті, надійність є критично важливим нефункціональним аспектом. Система повинна забезпечувати високу доступність з мінімальним часом простою та мати механізми для швидкого відновлення після збоїв, включаючи резервне копіювання та відновлення даних.

Для реалізації задачі буде використано вбудована система ідентифікації ASP.NET Core Identity, яка забезпечує повний набір функцій для управління користувачами, ролями, правами доступу та багатофакторною аутентифікацією.

2.4 Опис процесу авторизації

Процес авторизації (рисунок 2.5) є критично важливим етапом у забезпеченні безпеки веб-додатків. Він дозволяє контролювати доступ користувачів до різних ресурсів і функцій додатка, гарантуючи, що лише авторизовані користувачі можуть отримати доступ до певних даних або виконувати певні дії.

Процес авторизації в ASP.NET Core Identity включає кілька ключових етапів, починаючи з автентифікації користувача, яка перевіряє його облікові дані за допомогою UserManager і SignInManager. Після успішного входу користувача система використовує його ідентифікацію для визначення доступу до різних ресурсів та функцій додатка. Авторизація здійснюється за допомогою ролей і політик, налаштованих у додатку. RoleManager керує ролями, дозволяючи адміністраторам додавати нові ролі або змінювати наявні, тоді як політики авторизації визначають правила доступу до конкретних частин додатка. У контролерах та діях використовується атрибут [Authorize], який перевіряє, чи відповідає користувач необхідним вимогам доступу. Це забезпечує надійну систему контролю доступу, яка дозволяє легко керувати правами користувачів і забезпечувати безпеку даних у додатку.

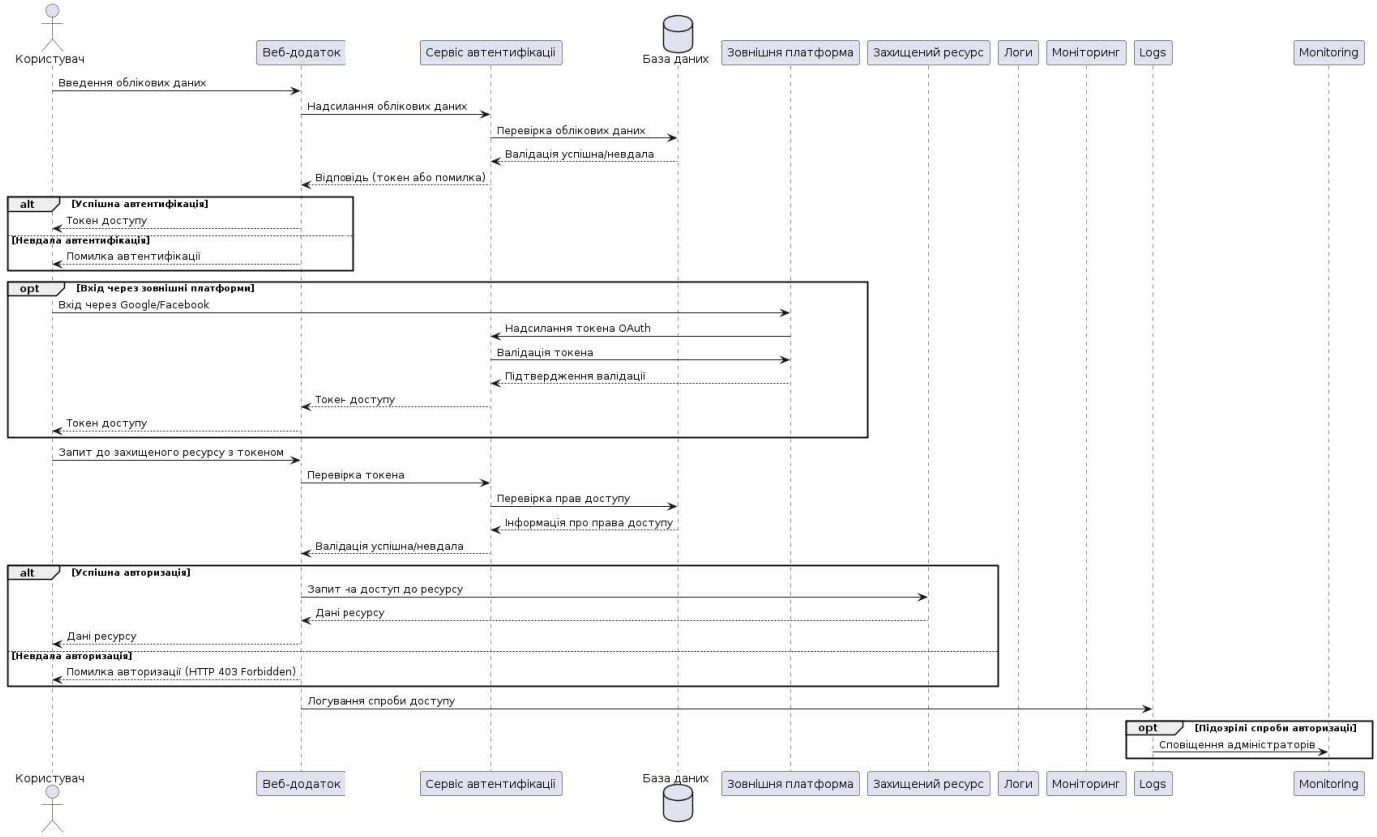


Рисунок 2.5 - Процес авторизації

Також можливий вхід через зовнішні платформи, такі як Google або Facebook, використовуючи протокол OAuth. Після успішної автентифікації система генерує токен доступу (наприклад, JWT), який включає інформацію про користувача та його права доступу.

Цей токен додається до запитів користувача і перевіряється сервером на валідність і термін дії. Кожен користувач може мати певні ролі, які визначають його права доступу до ресурсів і дій. Додатково до ролей можуть застосовуватися політики авторизації, що визначають складніші правила доступу, враховуючи такі параметри, як час доби або IP-адреса.

Якщо користувач авторизований, його запит обробляється, і він отримує доступ до запитуваного ресурсу, інакше система повертає помилку (наприклад, HTTP 403

Forbidden). Всі спроби доступу, особливо несанкціоновані, документуються в логах, що допомагає виявляти потенційні загрози і аналізувати поведінку користувачів. Інструменти моніторингу можуть сповіщати адміністраторів про підозрілі або невдалі спроби авторизації.

Ефективний процес авторизації забезпечує надійний захист конфіденційних даних і ресурсів додатка, запобігає несанкціонованому доступу та підвищує загальний рівень безпеки, покращуючи користувацький досвід за рахунок швидкого і зручного доступу до необхідних функцій і ресурсів. Використання сучасних методів автентифікації, таких як дворівнева автентифікація, і чітких політик контролю доступу дозволяє створити безпечну і гнучку систему, яка відповідає вимогам сучасних веб-додатків.

2.5 Висновки до другого розділу

На основі розглянутих аспектів можна зробити докладний висновок про процес авторизації в веб-додатках. Перш за все, важливо визначити, що сучасні підходи до авторизації, такі як OAuth 2.0 та OpenID Connect, відкривають нові можливості для забезпечення безпеки, гнучкості та зручності у використанні веб-додатків. Використання стандартів і протоколів авторизації, таких як JWT, дозволяє ефективно передавати та перевіряти інформацію про авторизацію та автентифікацію між сторонами.

У виборі системи авторизації для веб-додатків слід зосередитися на різноманітних критеріях, таких як функціональні вимоги, безпека, продуктивність, масштабованість, зручність використання та інтеграція з іншими системами. Наприклад, використання системи ідентифікації ASP.NET Core Identity може бути вигідним враховуючи повний набір функцій для управління користувачами та їхніми правами доступу.

Процес авторизації в ASP.NET Core включає кілька ключових етапів, таких як автентифікація користувача, визначення доступу за допомогою ролей та політик, і моніторинг доступу для виявлення потенційних загроз. Застосування таких методів, як

дворівнева автентифікація та чіткі політики контролю доступу, допомагає створити безпечну та гнучку систему, яка відповідає сучасним вимогам безпеки веб-додатків.

В цілому, ефективний процес авторизації в ASP.NET Core дозволяє забезпечити надійний захист даних та ресурсів веб-додатків, забезпечуючи при цьому зручний користувацький досвід і високу продуктивність. Збалансований підхід до вибору системи авторизації та впровадження сучасних методів автентифікації дозволяє створити безпечну та гнучку систему, яка задовольняє потреби як користувачів, так і розробників.

РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ АВТОРИЗАЦІЇ

3.1 Архітектура системи авторизації

Архітектура додатка передбачає багаторівневу структуру (див. рисунок 3.2): Presentation Layer (Front-end), яка відповідає за взаємодію з користувачами, відображення даних та відправлення запитів до сервера; Business Logic Layer (Back-end), що обробляє запити, реалізує бізнес-логіку, управління автентифікацією та авторизацією; та Data Access Layer, яка взаємодіє з базою даних та виконує CRUD операції.

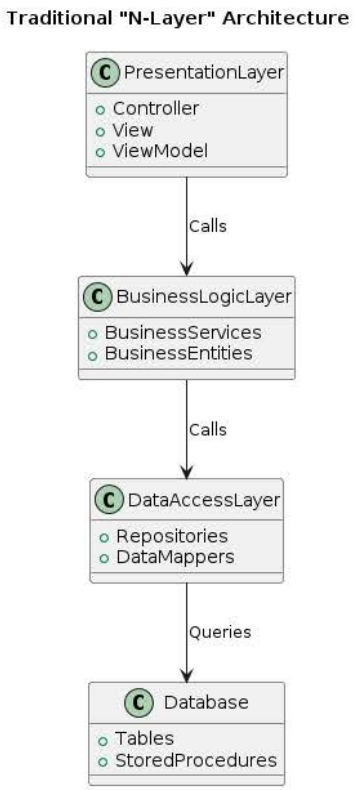


Рисунок 3.1 – Рівні логіки в багаторівневій структурі

Використовуючи цю архітектуру, користувачі роблять запити через Рівень інтерфейсу користувача (Presentation Layer) який взаємодіє тільки з рівнем бізнес логіки (Business Logic Layer). Бізнес логіка, у свою чергу, може викликати Рівень доступу до

даних (Data Access Layer) для запитів на доступ до даних. Рівень інтерфейсу користувача не повинен робити запити до рівня доступу до даних безпосередньо Таким чином, кожен рівень має свою власну, добре відому відповідальність.

Одним з недоліків цього традиційного підходу до багаторівневості є те, що залежності під час компіляції йдуть зверху вниз. Тобто, рівень UI залежить від бізнес логіки, який залежить від рівня доступу до даних. Це означає, що BLL, який зазвичай містить найважливішу логіку в додатку, залежить від деталей реалізації доступу до даних (і часто від наявності бази даних). Тестування бізнес-логіки в такій архітектурі часто буває складним і вимагає наявності тестової бази даних. Принцип інверсії залежностей може бути використаний для вирішення цієї проблеми, як ви побачите в наступному розділі.

Хоча додаток може використовувати кілька проектів для організаційних цілей, він все одно розгортається як єдине ціле, і його клієнти будуть взаємодіяти з ним як з одним веб-додатком. Це дозволяє дуже спростити процес розгортання. [21]

Клієнт та сервер взаємодіють за допомогою протоколу HTTP (Hypertext Transfer Protocol). Клієнти відправляють HTTP-запити на сервер, а сервери надсилають відповіді на ці запити.

У веб-застосунках ASP.NET Core клієнтська частина зазвичай складається з HTML, CSS і JavaScript, які генеруються сервером або відправляються на клієнтський браузер для відображення, в той час як серверна частина виконується на серверах, на яких розгорнута програма ASP.NET Core. Ці сервери обробляють HTTP-запити від клієнтів та генерують відповіді, які відправляються назад до клієнтів.

ASP.NET Core використовує технологію Razor Pages - це сторінковий підхід, де кожна сторінка представляє собою клас моделі сторінки, що відповідає конкретній веб-сторінці.

Модель сторінки включає в себе як код для обробки запитів (бекенд), так і код для відображення даних (фронтенд), що робить структуру додатка більш інтуїтивною і зручною для розробників. Кожна сторінка має файл .cshtml для представлення і файл .cs

для обробки логіки. Це дозволяє розробникам зосередитися на окремих сторінках, полегшуючи управління кодом і його повторне використання.

Архітектура Razor Pages інтегрується з системою маршрутизації ASP.NET Core, що дозволяє визначати маршрути для кожної сторінки. Наприклад, сторінка з файлом Contact.cshtml буде доступна за URL-адресою /Contact. Маршрутизація в Razor Pages є досить простою і дозволяє легко створювати зрозумілі та SEO-оптимізовані URL-адреси. Крім того, Razor Pages підтримує прив'язку моделей, що дозволяє автоматично передавати дані з HTTP-запитів до властивостей моделі сторінки, роблячи обробку форм і запитів більш зручною (див. рисунок 3.2).

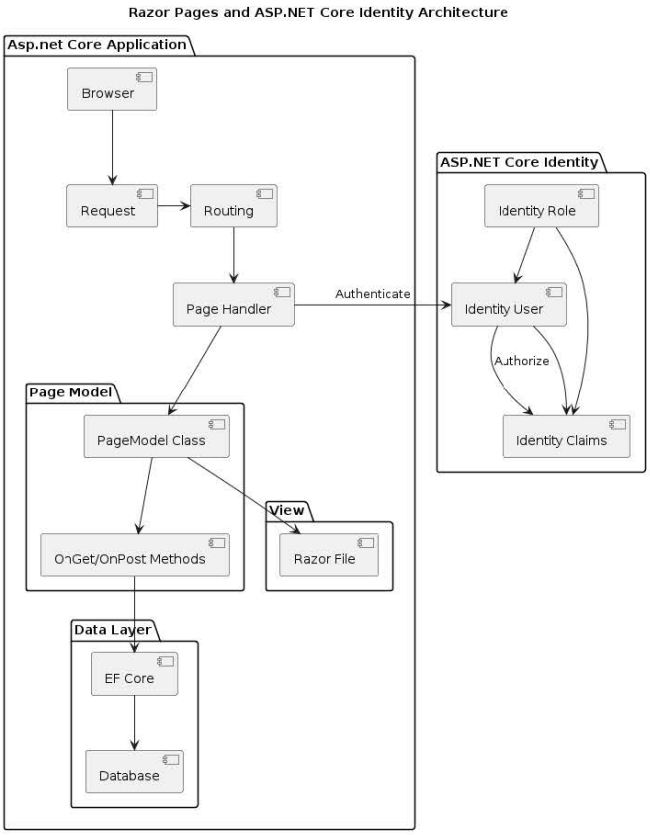


Рисунок 3.2 – Механізм роботи Razor Pages

Сервер авторизації, побудований на основі технології ASP.NET Core, обробляє ці запити за допомогою компонентів Authentication Middleware і Authorization Middleware. Ці компоненти відповідають за перевірку облікових даних користувачів, видачу токенів

доступу, а також за перевірку прав доступу до ресурсів на основі ролей та політик доступу.

Компоненти (Middleware) можуть бути додані до додатку в формі ланцюжка, де кожен компонент обробляє запити в певному порядку. Це дає можливість створювати комплексні обробники для запитів, які проходять через кілька етапів обробки. На рисунку 3.3 зображена схема роботи компонентів в asp.net core.

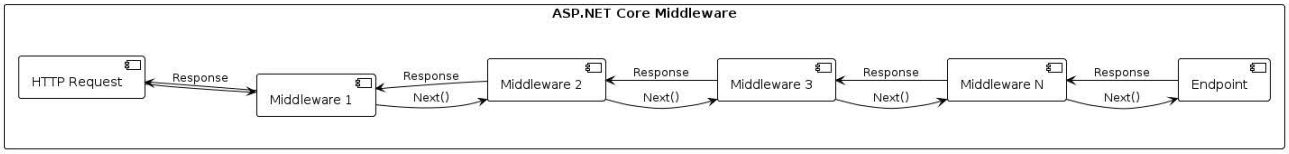


Рисунок 3.3 – Схема роботи компонентів

ASP.NET Core має вбудовану підтримку і внутрішньо використовує техніку, відому як ін'єкція залежностей. Ін'єкція залежностей - це техніка, яка забезпечує слабкий зв'язок між різними частинами програми. Більш слабкий зв'язок є бажаним, оскільки він полегшує ізоляцію частин програми, що дозволяє тестувати або замінювати їх. Це також зменшує ймовірність того, що зміна в одній частині програми матиме несподіваний вплив на іншу частину програми. Ін'єкція залежностей базується на принципі інверсії залежностей і часто є ключовим для досягнення принципу відкритості/закритості. Оцінюючи, як ваш додаток працює з залежностями, остерігайтеся запаху статичного коду і пам'ятайте афоризм «нове - це клей».

Для зручності, в ASP.NET Core використовуються контролери, які приймають HTTP-запити, обробляють їх та повертають відповіді, а також можна використовувати проміжні шари (middleware) для обробки запитів і відповідей перед їх передачею контролерам або після отримання відповідей від контролерів. На рисунку 3.4 зображений алгоритм за яким працюють контролери ASP.NET Core.

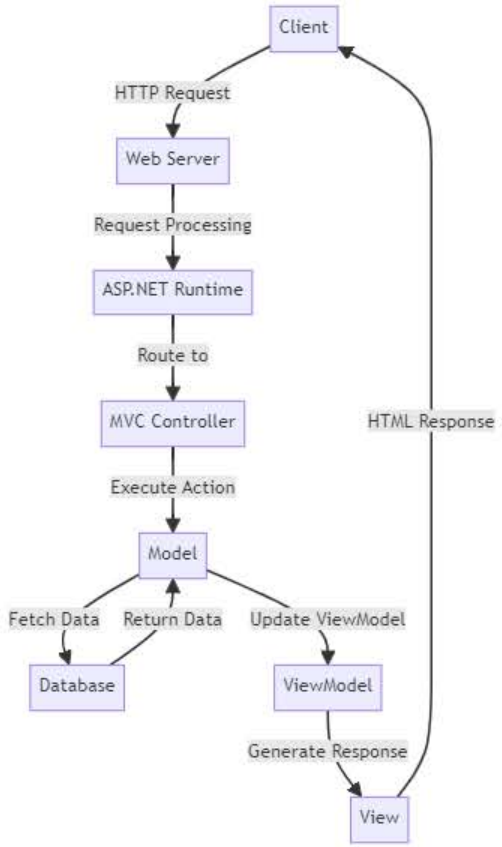


Рисунок 3.4 – Схема роботи контролерів ASP.NET Core.

Інформація про користувачів, їхні облікові дані, ролі та права доступу зберігається у базі даних, яка може бути реалізована на основі SQL або NoSQL технологій. Менеджери ролей та політик забезпечують можливість управління ролями користувачів та визначення політик доступу. Інтеграційні сервіси, такі як OAuth2, OpenID Connect, а також провайдери соціальної аутентифікації та зовнішні корпоративні служби (забезпечують підтримку сучасних стандартів аутентифікації та можливість єдиного входу.

Така архітектура забезпечує ефективну і безпечну роботу системи авторизації, відповідаючи потребам сучасних веб-додатків (див. рисунок 3.5).

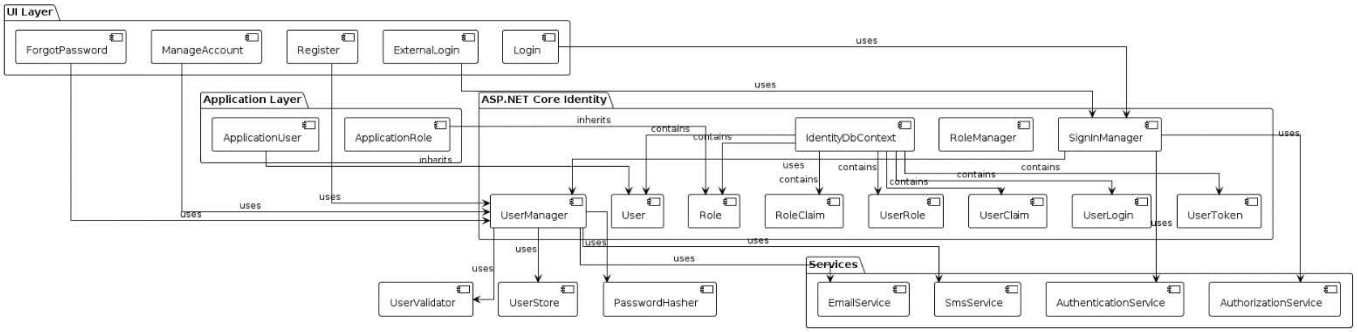


Рисунок 3.5 – Загальна архітектура програми

3.2 Вибір технологій та інструментів

Вибір технологій та інструментів для реалізації авторизації у веб-додатках включає кілька основних категорій. Серед фреймворків і бібліотек, ASP.NET Core Identity пропонує глибоку інтеграцію з ASP.NET Core, підтримуючи рольову, політично орієнтовану та claims-based авторизацію, що ідеально підходить для додатків на ASP.NET Core.

JSON Web Token (JWT) - це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON. Ця інформація може бути перевірена і їй можна довіряти, оскільки вона підписана цифровим підписом. JWT можуть бути підписані за допомогою секретного (за допомогою алгоритму HMAC) або відкритого/закритого ключа з використанням RSA або ECDSA [6].

OAuth 2.0 та OpenID Connect є стандартизованими протоколами, які дозволяють використовувати сторонні провайдери автентифікації, такі як Google або Facebook, що ідеально підходять для додатків, що потребують інтеграції з зовнішніми сервісами.

Серед серверних платформ, ASP.NET Core відзначається крос-платформенністю, високою продуктивністю та підтримкою сучасних стандартів і протоколів, що робить його ідеальним для розробки продуктивних і безпечних веб-додатків. Фреймворк ASP.NET Core підтримує Razor Pages, який дозволяє використовувати мову C# для

програмування веб сторінок, що дає розробнику кодувати серверну та клієнтську частину веб-сайту на одній мові програмування, що прискорює і полегшує розробку веб-додатку.

Одним із ключових принципів роботи ASP.NET Core є його модульна архітектура. Різні складові системи, такі як HTTP-процесори, системи маршрутизації, обробники запитів та інші, реалізовані у вигляді окремих модулів. Це дозволяє платформі бути більш гнучкою та розширюваною, а також спрощує розробку та тестування.

Технологія ASP.NET Core пропонує вбудовану підтримку асинхронної обробки запитів, що дозволяє ефективно використовувати обмежені ресурси сервера та забезпечує відмінну відзивчивість додатків під час великих навантажень.

Однією з головоних особливостей ASP.NET Core є його кросплатформеність. Платформа може працювати на операційних системах Windows, macOS і Linux, що дозволяє розгортати додатки на різноманітних середовищах і платформах хмарних обчислень.

Також ASP.NET Core відомий своєю високою продуктивністю та ефективним використанням ресурсів сервера. Платформа використовує ряд оптимізаційних технік, щоб забезпечити швидку обробку запитів та оптимальне використання пам'яті.

Фреймворк ASP.NET Core надає вбудовані шаблони для реалізації різноманітних аспектів безпеки, таких як аутентифікація, авторизація, захист від зловмисних атак і т.д. Це спрощує розробку безпечних та надійних веб-додатків, має вбудовану підтримку управління залежностями через інструмент NuGet, а також підтримку контейнеризації за допомогою Docker. Це дозволяє легко вирішувати проблеми розгортання та масштабування додатків.

ASP.NET Core легко інтегрується з іншими продуктами та технологіями Microsoft, такими як Azure, Visual Studio, SQL Server тощо. Це забезпечує зручну розробку, тестування та розгортання додатків.

ASP.NET Core надає різні інструменти для забезпечення безпеки веб-додатків, включаючи аутентифікацію, авторизацію, захист від міжсайтових скриптів (XSS) та міжсайтової підробки запитів (CSRF)

Ці принципи роботи ASP.NET Core забезпечують потужні та гнучкі інструменти для створення сучасних веб-додатків і служб, що відповідають вимогам сучасного програмного забезпечення.

Разом ASP.NET Core буде використано Scaffolding, що є функцією, яка дозволяє швидко генерувати базовий код для різних частин програми, таких як моделі, контролери та представлення (views), а також ця функція дозволяє швидко додати до проекту стандартні сторінки реєстрації, входу, відновлення паролю та інші компоненти, пов'язані з керуванням користувачами.

Дані користувачів будуть зберігатися в реляційній базі даних Microsoft SQL Server, яка має глибоку інтеграцію з ASP.NET Core та надає широкі можливості для роботи з даними. Для зручного і ефективного управління даними в базі даних буде використано Entity Framework який виступає як об'єктно-реляційний мапер (ORM), що дозволяє розробникам працювати з базою даних за допомогою об'єктів, а не писати SQL-запити вручну [7].

Використання Scaffolding Identity дозволяє розробникам легко налаштовувати та розширювати функціонал аутентифікації без необхідності створювати все з нуля. Процес включає встановлення відповідних пакетів, налаштування сервісів у Program.cs та виконання команди для генерації необхідних файлів, що значно спрощує інтеграцію і налаштування системи ідентифікації у додатку.

Для управління автентифікацією та авторизацією, ASP.NET Core Identity пропонує розширювану платформу, яка підтримує OpenID Connect та OAuth 2.0, що підходить для корпоративних додатків. Auth0, хмарне рішення для автентифікації та авторизації, підтримує різні протоколи та провайдери автентифікації, що є ідеальним для стартапів та компаній, які бажають швидко впровадити надійне рішення без необхідності управління інфраструктурою.

Сам сайт буде розгорнутий завдяки хмарному сервісу MonsterASP.NET який надає безкоштовні послуги для розгортання, масштабування та керування простих ASP.NET Core додатків.

3.3 Опис інтерфейсу програми

Інтерфейс веб-додатку з авторизацією відіграє важливу роль у забезпеченні зручності використання, ефективної навігації та безпеки користувачів. Основні елементи інтерфейсу включають головну сторінку, сторінки реєстрації та входу, профіль користувача і панель адміністрування.

Головна сторінка (рисунок 3.6) містить логотип та назву додатку, навігаційне меню з посиланнями на основні розділи (Головна, Про нас, Можливості, Вхід, Реєстрація), а також основний контент, який може варіюватися залежно від специфіки додатка.

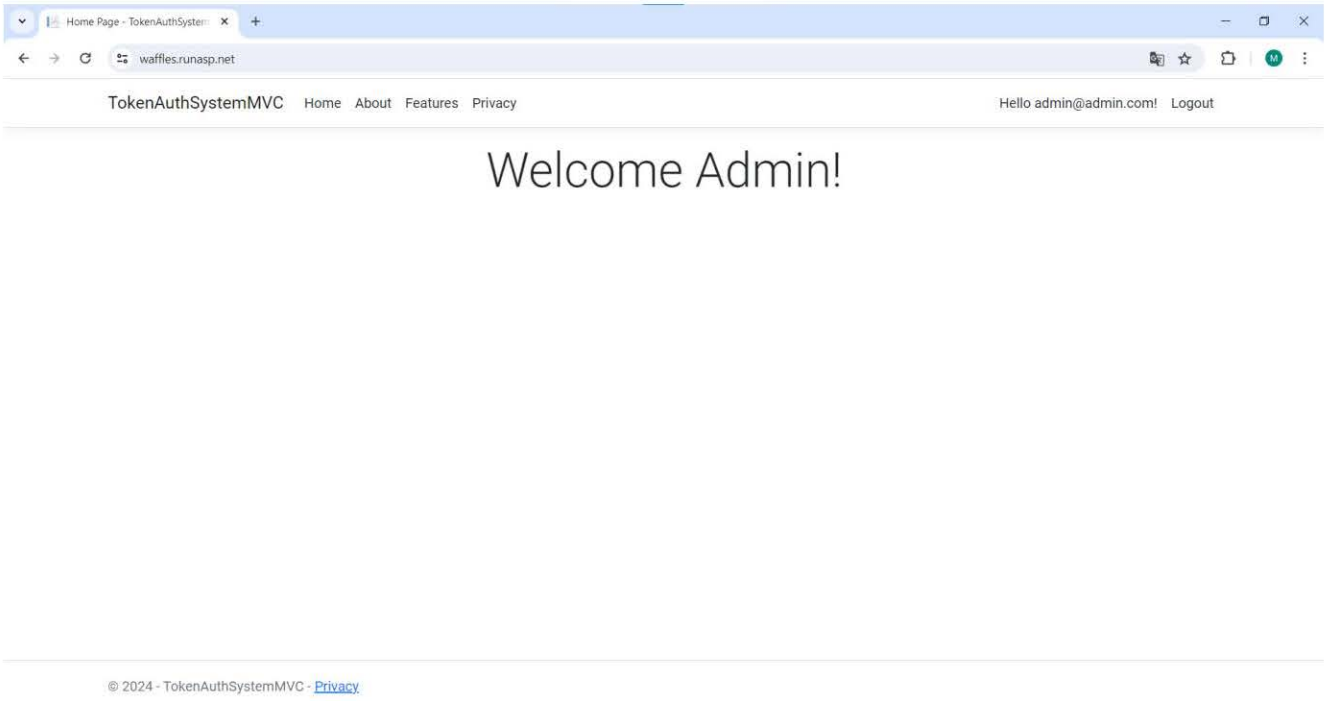


Рисунок 3.6 - Головна сторінка

Сторінка реєстрації (рисунок 3.7) дозволяє новим користувачам створити обліковий запис через форму реєстрації, яка включає поля для введення імені, прізвища, електронної пошти, пароля та підтвердження пароля. Після заповнення всіх полів користувач натискає кнопку "Зареєструватися".

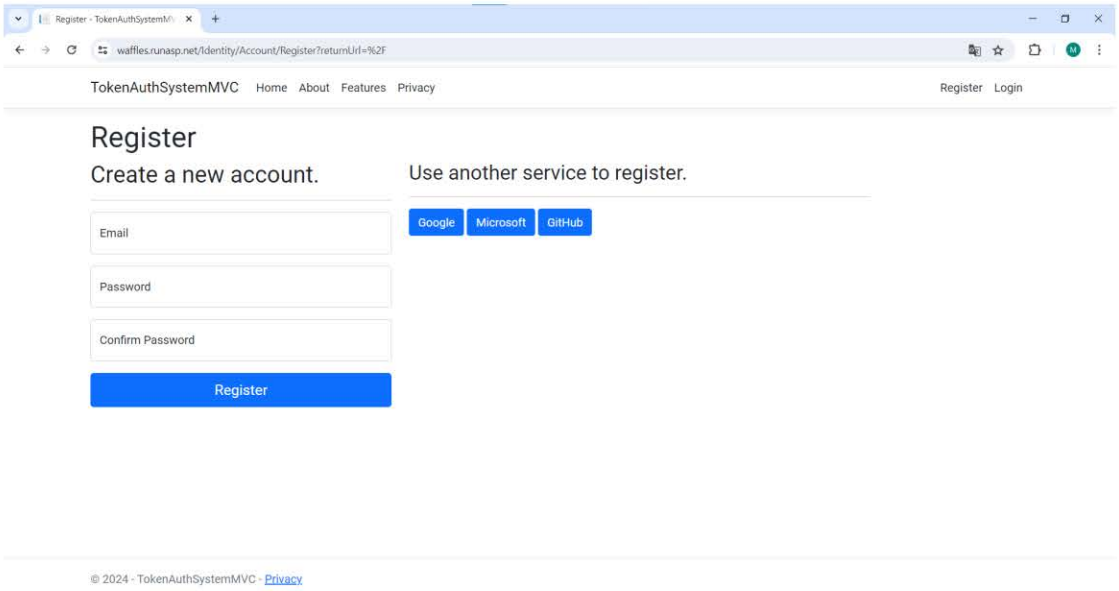


Рисунок 3.7 - Сторінка реєстрації

Сторінка входу (рисунок 3.8) дозволяє користувачам увійти до свого облікового запису, ввівши електронну пошту та пароль у відповідні поля форми входу, і натиснувши кнопку "Увійти". Для користувачів, які забули свій пароль, є посилання "Забули пароль?", що дозволяє відновити доступ. Сторінка профілю користувача містить особисту інформацію (ім'я, прізвище, електронну пошту, дату реєстрації) та надає можливість редагувати ці дані, а також змінювати пароль та налаштування облікового запису. Зміни зберігаються після натискання відповідної кнопки.

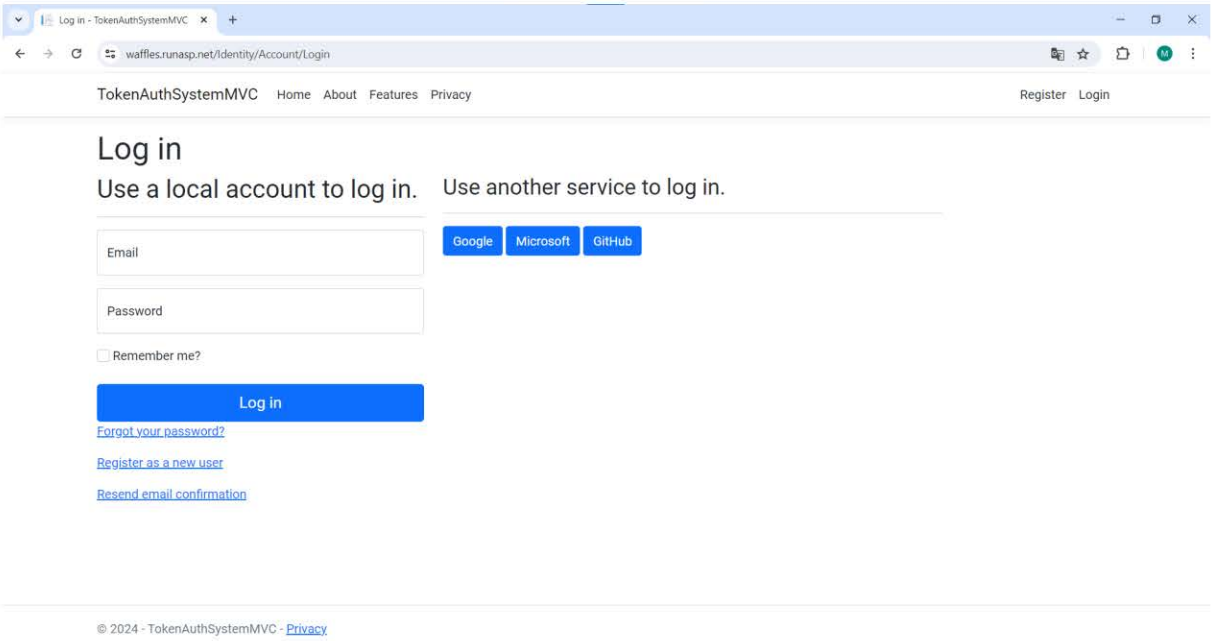


Рисунок 3.8 - Сторінка входу

Панель адміністрування (рисунок 3.9) доступна лише користувачам з адміністративними правами і дозволяє керувати користувачами та контентом додатка. Вона включає інструменти для редагування, блокування або видалення облікових записів, призначення ролей та налаштування прав доступу, а також створення, редагування та видалення контенту (статті, новини, оголошення). Панель адміністрування також містить аналітичні дані щодо активності користувачів, відвідуваності сайту та використання ресурсів.

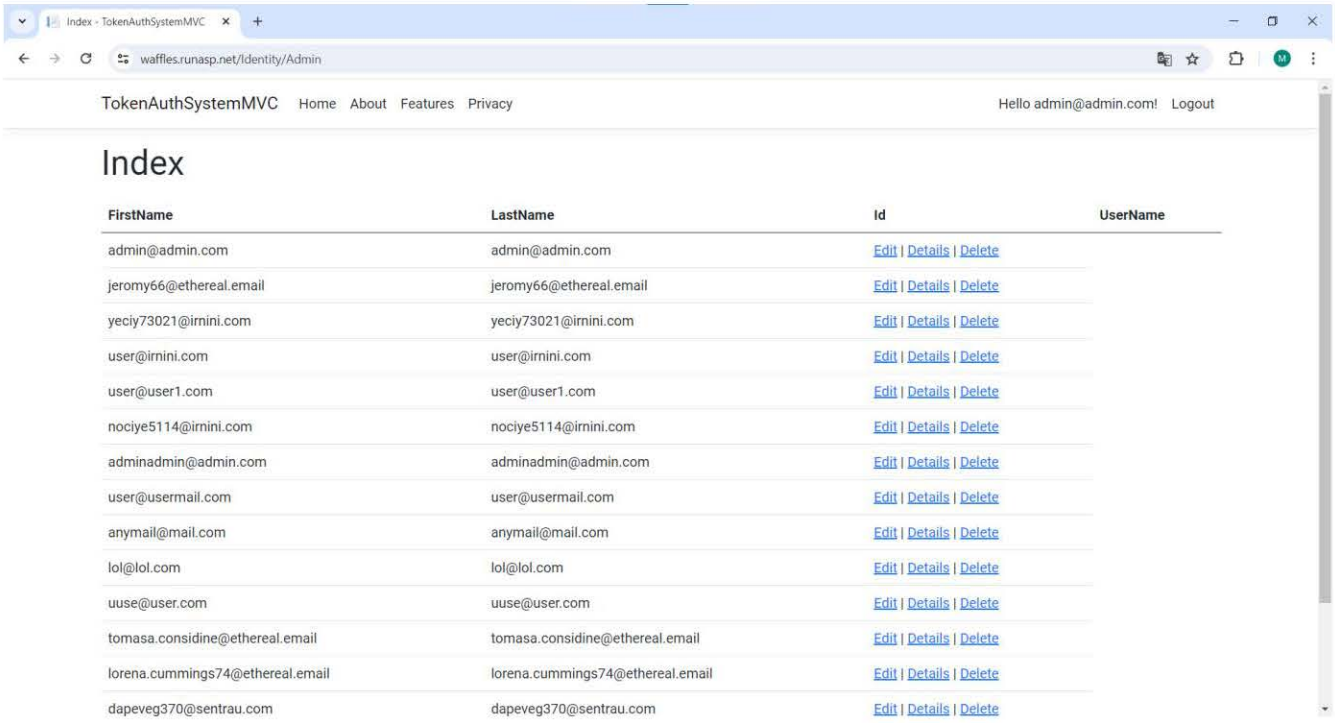


Рисунок 3.9 - Панель адміністрування

Для забезпечення безпеки додаток використовує дворівневу автентифікацію (2FA), налаштування для увімкнення/вимкнення 2FA, та управління пристроями для додаткової автентифікації. Форми для відновлення пароля через електронну пошту або мобільний телефон дозволяють користувачам відновити доступ. Інтерфейс для перегляду історії входів та інших важливих дій користувачів допомагає виявити підозрілу активність. Використання інтуїтивно зрозумілих форм і чітких інструкцій забезпечує зручність взаємодії з додатком, а впровадження необхідних засобів захисту гарантує безпеку даних користувачів.

3.4 Моделювання користувацьких сценаріїв

Моделювання користувацьких сценаріїв є важливим етапом у розробці веб-додатків, оскільки дозволяє краще зрозуміти потреби користувачів і забезпечити ефективний та інтуїтивно зрозумілий інтерфейс. Це також допомагає ідентифікувати та

виправити потенційні проблеми на ранніх етапах розробки. Розглянемо декілька типових сценаріїв користування веб-додатком з акцентом на процесах авторизації та управління доступом. На рисунку 3.10 зображені сценарії аутентифікації та використані інструменти.

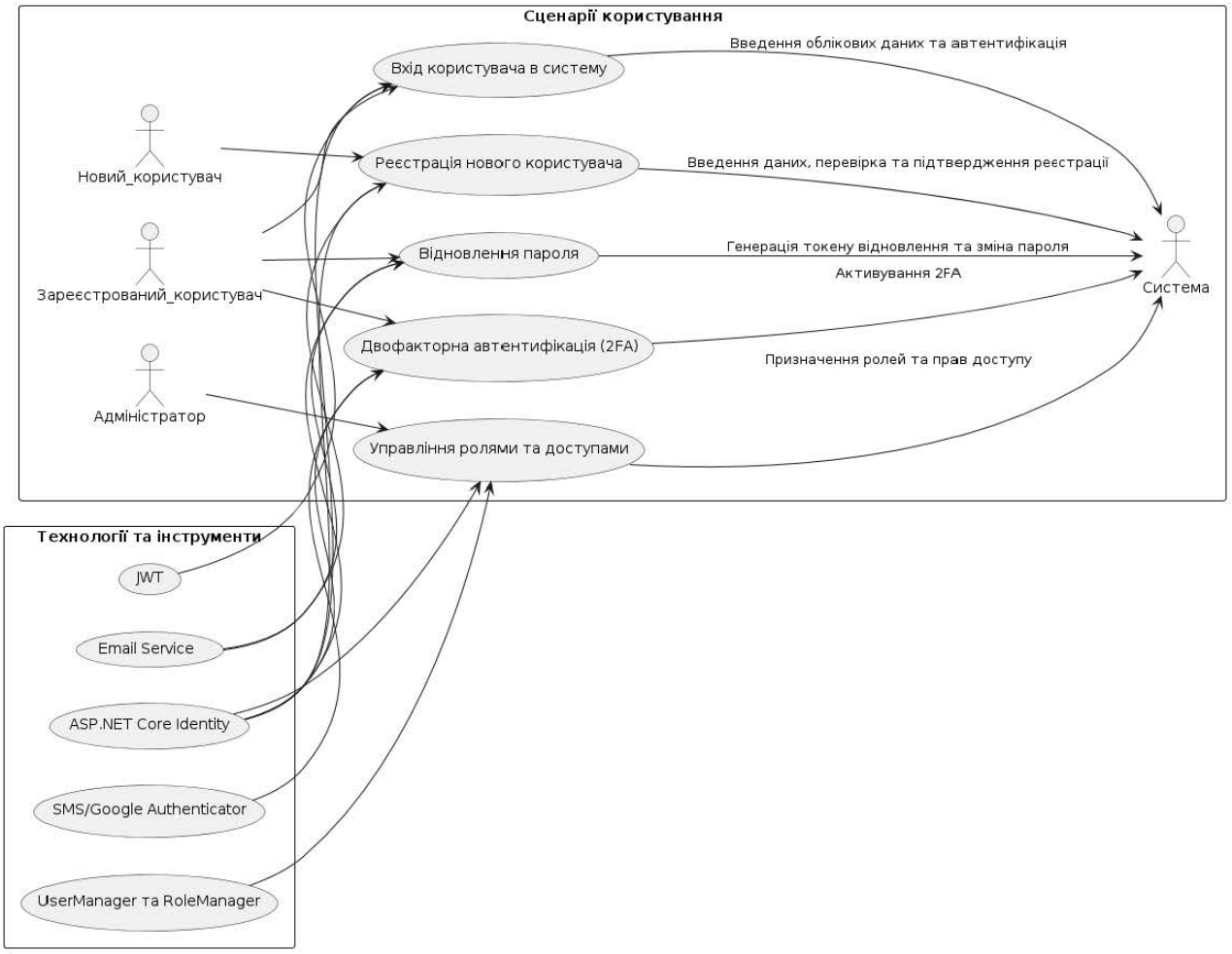


Рисунок 3.10 – Сценарій користування

Перший сценарій стосується реєстрації нового користувача. Новий користувач заходить на сайт і бажає створити обліковий запис. Він натискає кнопку "Реєстрація", заповнює форму реєстрації, вводячи необхідні дані (ім'я, електронна пошта, пароль), а система перевіряє коректність введених даних та унікальність електронної пошти. Після отримання підтвердження реєстрації на електронну пошту обліковий запис активується,

і користувач може увійти в систему. Для цього використовуються ASP.NET Core Identity для управління користувачами та Email Service для відправлення підтверджень.

Другий сценарій стосується входу користувача в систему. Зареєстрований користувач заходить на сайт і вводить свої облікові дані для входу. Він натискає кнопку "Вхід", вводить свою електронну пошту та пароль, і система перевіряє правильність введених даних. У випадку правильної автентифікації користувач перенаправляється на головну сторінку з персоналізованим контентом, а у разі помилкових даних отримує повідомлення про помилку та пропозицію відновити пароль. Для цього використовуються ASP.NET Core Identity для автентифікації та JWT для генерації токенів (якщо використовується безстейтна автентифікація).

Третій сценарій передбачає двофакторну автентифікацію (2FA). Користувач бажає підвищити безпеку свого облікового запису, увімкнувши двофакторну автентифікацію. Він заходить у налаштування облікового запису, вибирає опцію двофакторної автентифікації, а система пропонує вибір способу (SMS, мобільний додаток). Користувач підтверджує свій вибір, вводячи код, отриманий через вибраний спосіб, і після підтвердження 2FA активується для облікового запису. Для цього використовуються ASP.NET Core Identity та сторонні сервіси для SMS або мобільного додатку (наприклад, Google Authenticator).

Четвертий сценарій стосується відновлення пароля. Користувач забув свій пароль і бажає його відновити. Він натискає посилання "Забули пароль?" на сторінці входу, вводить свою електронну пошту, і система генерує токен відновлення та відправляє посилання на відновлення пароля на вказану електронну пошту. Користувач переходить за посиланням, вводить новий пароль, і після підтвердження пароль змінюється, і користувач може увійти з новим паролем. Для цього використовуються ASP.NET Core Identity та Email Service для відправлення посилань на відновлення пароля.

П'ятий сценарій стосується управління ролями та доступами. Адміністратор керує правами доступу користувачів, призначаючи їм відповідні ролі. Він заходить в панель управління, вибирає користувача зі списку, призначає або змінює роль користувача

(наприклад, користувач, модератор, адміністратор), і після збереження змін права доступу користувача оновлюються відповідно до нової ролі. Для цього використовуються ASP.NET Core Identity, UserManager та RoleManager для управління користувачами та ролями.

Моделювання користувацьких сценаріїв дозволяє виявити та виправити потенційні проблеми на ранніх етапах розробки веб-додатків. Вибір відповідних технологій та інструментів, таких як ASP.NET Core Identity, JWT, OAuth 2.0, Email Service та двофакторна автентифікація, забезпечує надійну і безпечну роботу додатку, а також покращує користувацький досвід.

3.5 Висновки до третього розділу

Ураховуючи описану архітектуру системи авторизації в контексті веб-додатка, можна зробити наступні висновки. Перш за все, важливо визнати, що багаторівнева структура додатка сприяє ефективному розподілу функціоналу та забезпеченню його безпеки та стабільності. Поділ на рівні логіки дозволяє кожному рівню виконувати власні завдання, що сприяє полегшенню розробки, тестування та підтримки коду.

Далі, важливо відзначити переваги технологій та інструментів, що використовуються в архітектурі. ASP.NET Core, як базова технологічна платформа, забезпечує надійність та безпеку, а також широкі можливості розширення за допомогою Middleware та інших компонентів. Використання принципу ін'єкції залежностей спрощує управління залежностями та робить код більш гнучким та легко тестованим.

Вибір технологій та інструментів визначає ефективність та масштабованість веб-додатка. Використання потужних інструментів, таких як ASP.NET Core Identity для керування користувачами, JWT для безпеки та масштабованості авторизації, сприяє створенню стабільної та безпечної системи.

Опис інтерфейсу програми веб-додатка з авторизацією надає загальне уявлення про його функціональність та спрямованість на зручність, безпеку та ефективність використання.

Головна сторінка дозволяє легко переміщуватися між основними розділами додатку, а сторінки реєстрації та входу забезпечують зручний та безпечний спосіб створення облікового запису та входу до системи.

Особистий профіль користувача дозволяє керувати особистою інформацією та налаштуваннями облікового запису, а панель адміністрування забезпечує адміністраторам широкі можливості керування користувачами та контентом додатку. Додаткові засоби безпеки, такі як дворівнева аутентифікація та історія входів, підвищують рівень захисту даних користувачів.

Моделювання користувацьких сценаріїв є важливим етапом у розробці веб-додатків, оскільки дозволяє краще зрозуміти потреби користувачів та забезпечити ефективний та інтуїтивно зрозумілий інтерфейс. Сценарії включають процеси реєстрації, входу, двофакторної автентифікації та управління ролями, що свідчить про проникливе вивчення користувацьких потреб та надійне планування функціональності веб-додатку. Це сприяє створенню продукту, який відповідає потребам користувачів та забезпечує їхню задоволеність від використання.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ АВТОРИЗАЦІЇ

4.1 Детальний опис розробки

Розробка системи авторизації для веб-додатку включає декілька ключових етапів, таких як налаштування середовища розробки, інтеграція з базою даних, налаштування автентифікації та авторизації, створення моделей даних користувачів та ролей, а також впровадження політик доступу. Спершу створюється новий проект ASP.NET Core з використанням шаблону "Web Application (Model-View-Controller)" та встановлюється пакет NuGet для ASP.NET Core Identity, якщо він ще не доданий.

Потім налаштовується база даних, зокрема додається пакет для роботи з базою даних SQL Server, та встановлюється Entity Framework Core Tools. Наступним кроком є налаштування сервісів Identity у Program.cs, де додаються налаштування для сервісів Identity та middleware аутентифікації та авторизації. Після цього створюються класи для користувачів і ролей, наприклад, клас користувача наслідується від IdentityUser, а також створюється клас ApplicationDbContext, який наслідується від IdentityDbContext<ApplicationUser>.

Додаємо пакети Microsoft.AspNetCore.Authentication.Google,Microsoft.AspNetCore.Authentication.JwtBearer, Microsoft.AspNetCore.Authentication.MicrosoftAccount та AspNet.Security.OAuth.Github. В файлі Program.cs додаємо підтримку різних способів авторизації OAuth, методом builder.Services.AddAuthentication().

Використовуючи сервіси Google зареєструємо програму для використання Google OAuth, створимо проект у Google API Console. Оберимо тип програми (наприклад, Web application), введіть назву клієнта та додайте авторизовані URI для перенаправлення (наприклад, http://localhost:8000/oauth2callback для локальної розробки). Після створення облікових даних буде надано Client ID та Client Secret, які потрібно зберегти для налаштування вашої програми. Це забезпечує інтеграцію Google OAuth у додаток, дозволяючи користувачам легко аутентифікуватися за допомогою своїх облікових

записів Google, забезпечуючи при цьому безпеку та конфіденційність їх даних. Результат реєстрації програми можна побачити на рисунку 4.1.

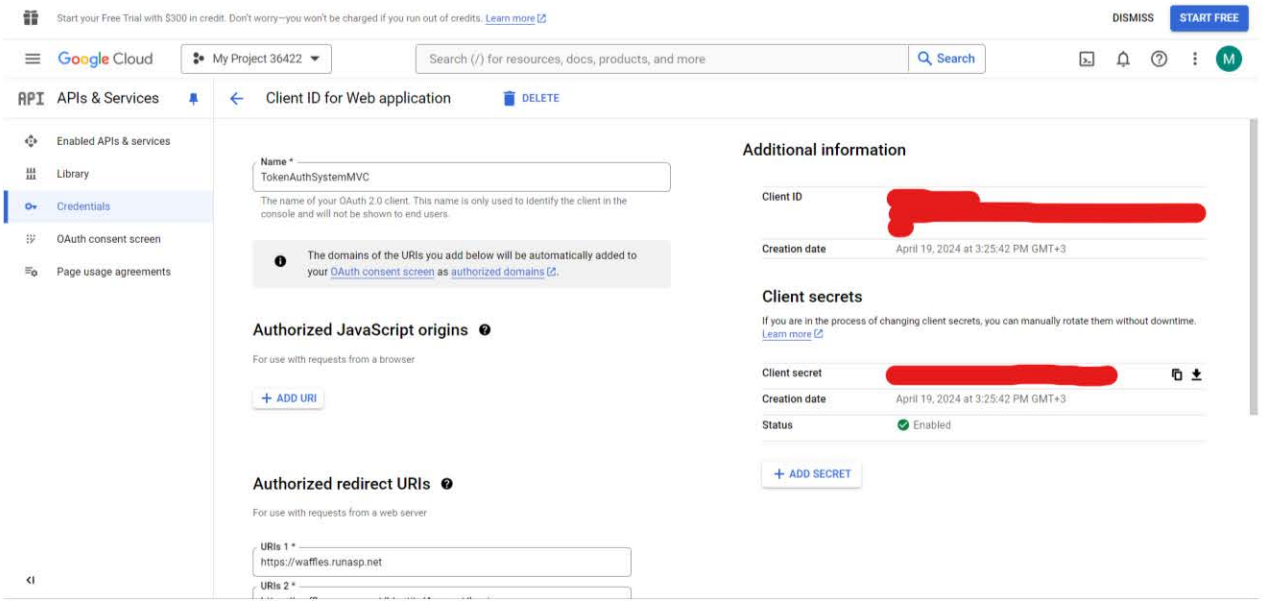


Рисунок 4.1 – Проект для авторизації через Google

Після цього, таким же чином зареєструємо варіанти авторизації через Microsoft та Github. (рисунок 4.2 – 4.3).

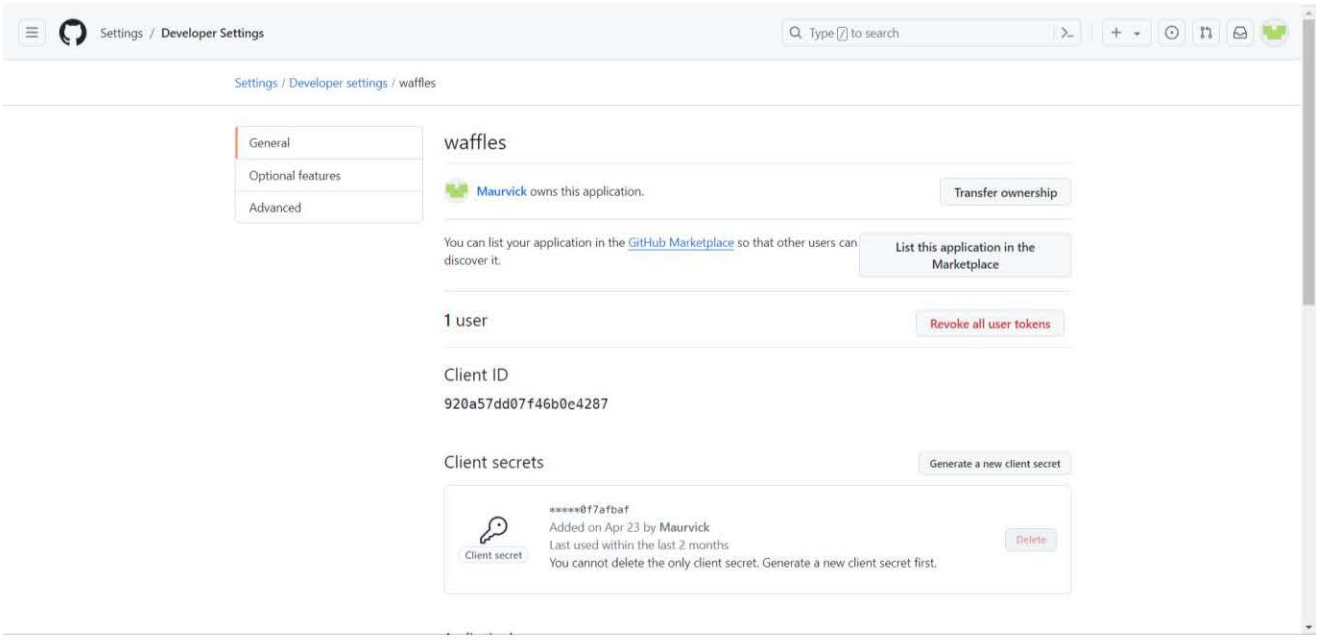


Рисунок 4.2- Проект для авторизації через Github

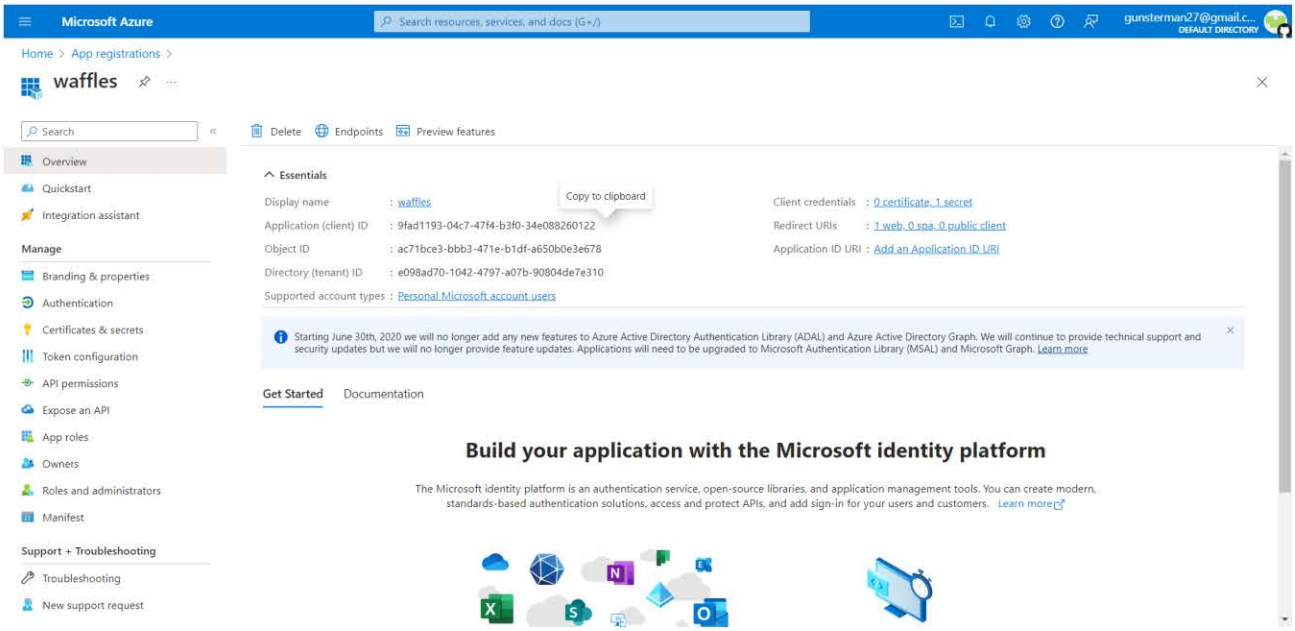


Рисунок 4.3- Проект для авторизації через Microsoft

Створюємо клас JwtTokenProvider.cs, він додає дані користувача в токен, який потім можна використовувати для авторизації користувача, перевірки його ролі. Додаємо ще один клас, EmailSender.cs, який реалізує інтерфейс IEmailSender, для підтвердження аккаунта завдяки пошті.

Налаштуємо контролер HomeController.cs для захисту контролерів та їх дій від несанкціонованого доступу використовуючи атрибути авторизації ([Authorize], [AllowAnonymous]).[6]

Завдяки функціоналу ASP.NET Core Scaffolding генеруємо сторінки, використовуючи шаблон Identity, та обравши файли Login.cshtml і Register.cshtml. Після цього обираємо шаблон Razor Pages using EntityFramework (CRUD) та створюємо панель адміністрування, роблячи деякі правки для того, щоб сторінки використовували класи ASP.NET Core Identity. В файлах Login.cshtml і Register.cshtml додаємо функцію генерації токенів після успішної авторизації користувача. Редагуємо Register.cshtml для того, щоб при реєстрації нового користувача йому надавались ролі.

В стартовому файлі Program.cs реєструємо користувача с правами адміністрування.

Створюємо змінні середовища (Environment variables) де будуть зберігатися поля для підключення до бази даних та секрети для використання сервісів авторизації.

Далі використовуються інструменти UserManager і RoleManager для управління користувачами та ролями, які вводяться через DI в контролери або служби. Наприклад, для створення користувача використовується метод CreateAsync(), а для створення ролі - метод CreateAsync() з використанням RoleManager. Для впровадження політик авторизації в Program.cs додаються політики авторизації, які потім використовуються в контролерах для контролю доступу до ресурсів.

Клас UserManager відповідає за керування користувачами. Це включає створення нових користувачів, оновлення інформації про користувачів, перевірку паролів, роботу з токенами підтвердження, тощо.

Клас RoleManager керує ролями в системі. Ролі використовуються для визначення прав доступу користувачів. Цей клас дозволяє створювати, оновлювати, видаляти ролі та призначати їх користувачам.

Клас SignInManager керує процесом автентифікації користувачів. Це включає вхід, вихід користувачів, перевірку багатофакторної автентифікації тощо.

Завершальним етапом є міграція бази даних для створення необхідних таблиць, що включає додавання та застосування міграцій за допомогою команд dotnet ef migrations add InitialCreate та dotnet ef database update.

Також для роботи класів необхідні моделі для опису даних такі як UserModel . Яка має поля Id, Email, Name, Role.

Після цього , сайт необхідно розгорнути на сервісі MonsterASP.NET. Використовуючи панель адміністрування (див. рисунок 4.4), за допомогою технології WebDeploy, встановлюємо сайт на хмару.

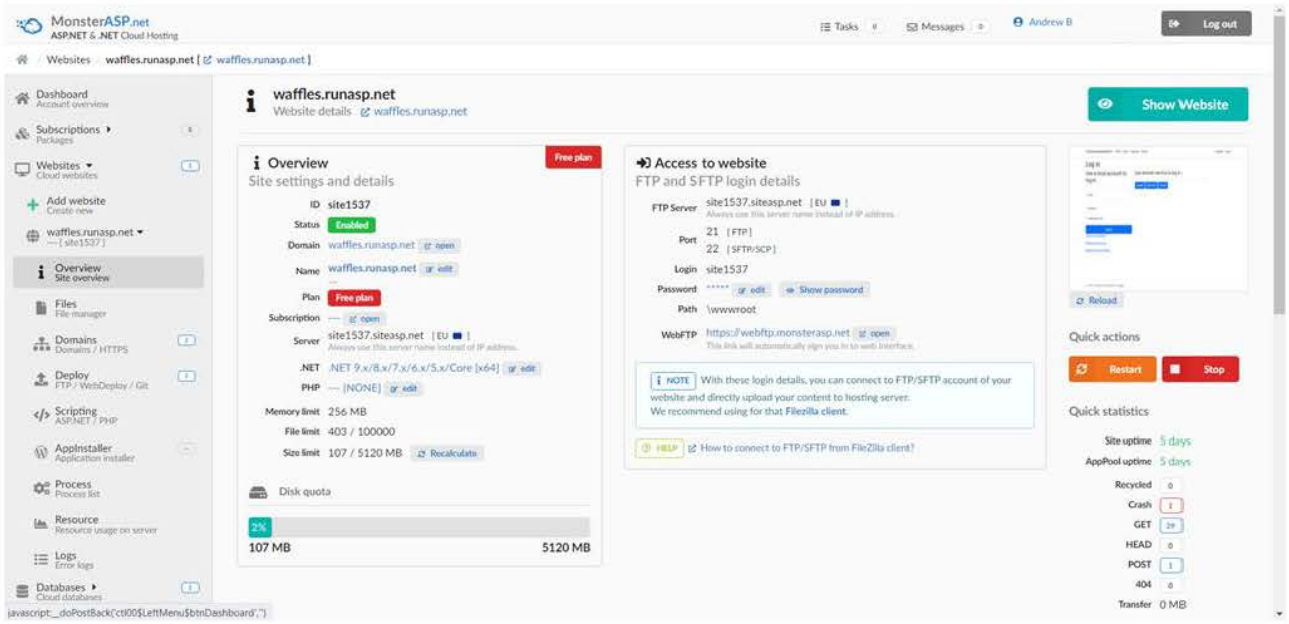


Рисунок 4.4 – Панель адміністрування MonsterASP.NET

Таким чином, розробка системи авторизації в ASP.NET Core включає налаштування середовища розробки, інтеграцію з базою даних, налаштування Identity, створення моделей даних користувачів та ролей, а також впровадження політик доступу. Використання класів UserManager і RoleManager забезпечує зручне управління користувачами та ролями, а впровадження політик авторизації дозволяє гнучко налаштовувати доступ до ресурсів системи.

4.2 Інтеграція з базою даних

Інтеграція з базою даних є одним із ключових аспектів розробки веб-додатка з авторизацією, оскільки вона забезпечує збереження та управління даними користувачів, ролей та іншої інформації, необхідної для функціонування додатка. Вибір бази даних залежить від середовища розгортання додатка: для додатків у середовищі Microsoft рекомендується використовувати SQL який забезпечує високу продуктивність і розширені функції. Налаштування бази даних включає створення нової бази даних через відповідні інструменти, такі як SQL Server Management Studio (SSMS) або VS Code.

Для інтеграції з базою даних використовується Entity Framework Core. Спочатку потрібно додати відповідні пакети NuGet до проекту. Далі створюється клас ApplicationDbContext, що успадковується від IdentityDbContext, і налаштовується контекст бази даних у Program.cs, де додається рядок підключення до файлу appsettings.json. Для конфігурації підключення до SQL Server використовуються відповідні методи UseSqlServer.

База даних для системи авторизації буде розгорнута на сервісі MonsterASP.NET. Цей сервіс надає безкоштовні бази розміром до 1 гігабайту (рисунок 4.5). Для того щоб підключитися до бази даних, необхідно спочатку розмістити додаток на сайті MonsterASP.NET. Для того , щоб використати базу даних під час розробки необхідно включити опцію віддаленого підключення, але її не рекомендується використовувати в других умовах (наприклад коли сайт розгорнутий на сайті) через втрату швидкості обробки даних (рисунок 4.6).

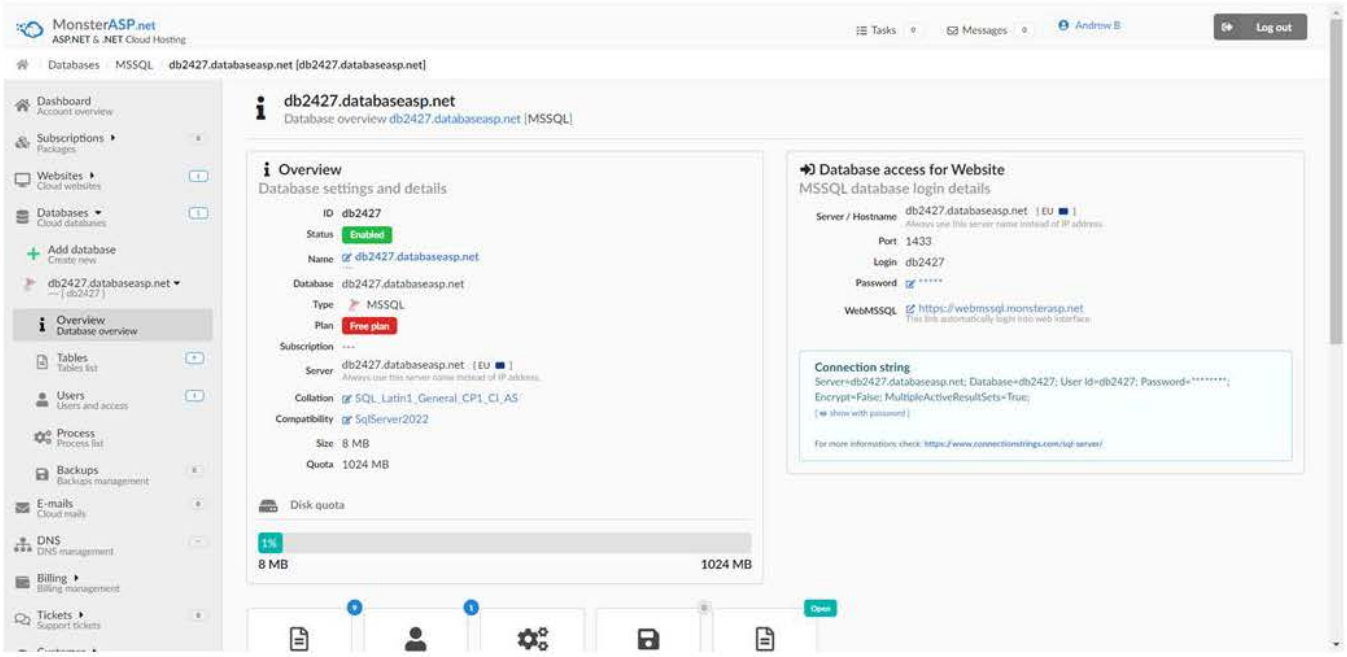


Рисунок 4.5 – База даних на сервісі MonsterASP.NET

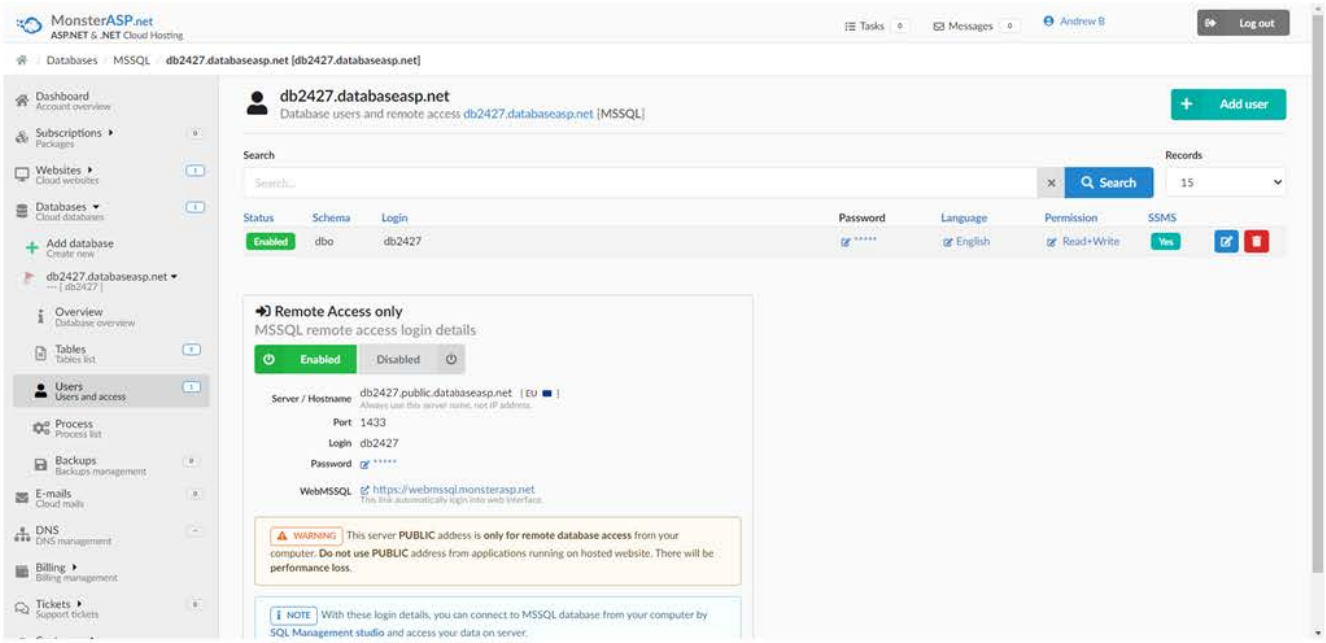


Рисунок 4.6 – База даних на сервісі MonsterASP.NET для розробки

Міграції бази даних здійснюються за допомогою інструментів командного рядка, що дозволяє створювати та застосовувати міграції для початкового налаштування бази даних та внесення подальших змін. Основні операції з базою даних включають CRUD (створення, читання, оновлення та видалення) операції з даними користувачів та ролей. Для реалізації цих операцій використовується LINQ для запитів до бази даних. У нашому випадку необхідно виконати міграцію задля того, щоб створити необхідні таблиці в базі даних, щоб зберігати відповідну інформацію та зчитувати її (дані користувачів, ролі, токени для входу). На рисунку 4.7 зображена схема відношення таблиць Identity

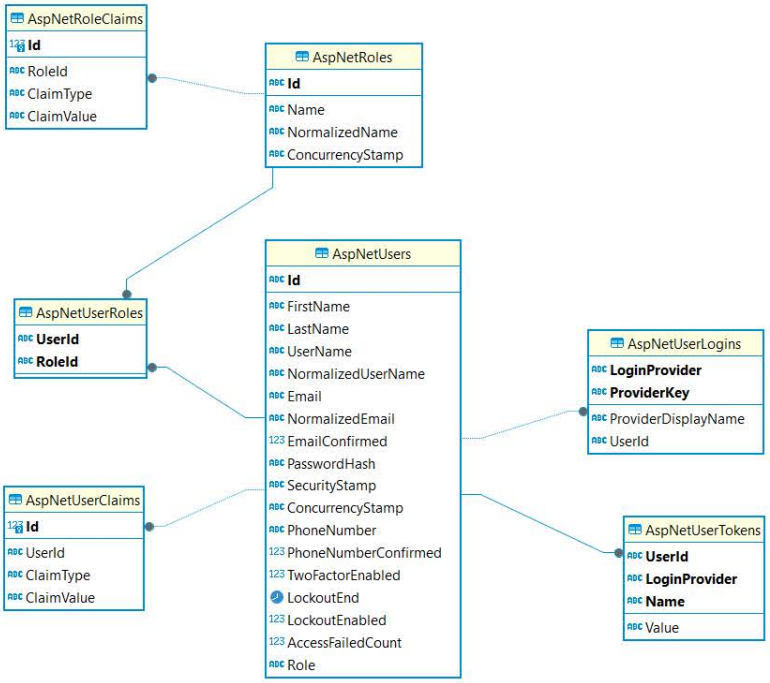


Рисунок 4.7 – таблиці Identity

Кожен User може мати багато UserClaims.

Кожен User може мати декілька UserLogins.

Кожен User може мати багато UserTokens.

Кожна Role може мати багато пов'язаних з нею Вимог до Ролі.

Кожен User може мати багато асоційованих Ролей, і кожна Роль може бути асоційована з багатьма Користувачами. Це зв'язок «багато-до-багатьох», який вимагає наявності в базі даних таблиці з'єднання. Об'єднана таблиця представлена сутністю UserRole.

Таблиця User Зберігає та витягує інформацію про користувача (наприклад, хеш імені користувача та пароля) (рисунок 4.8).

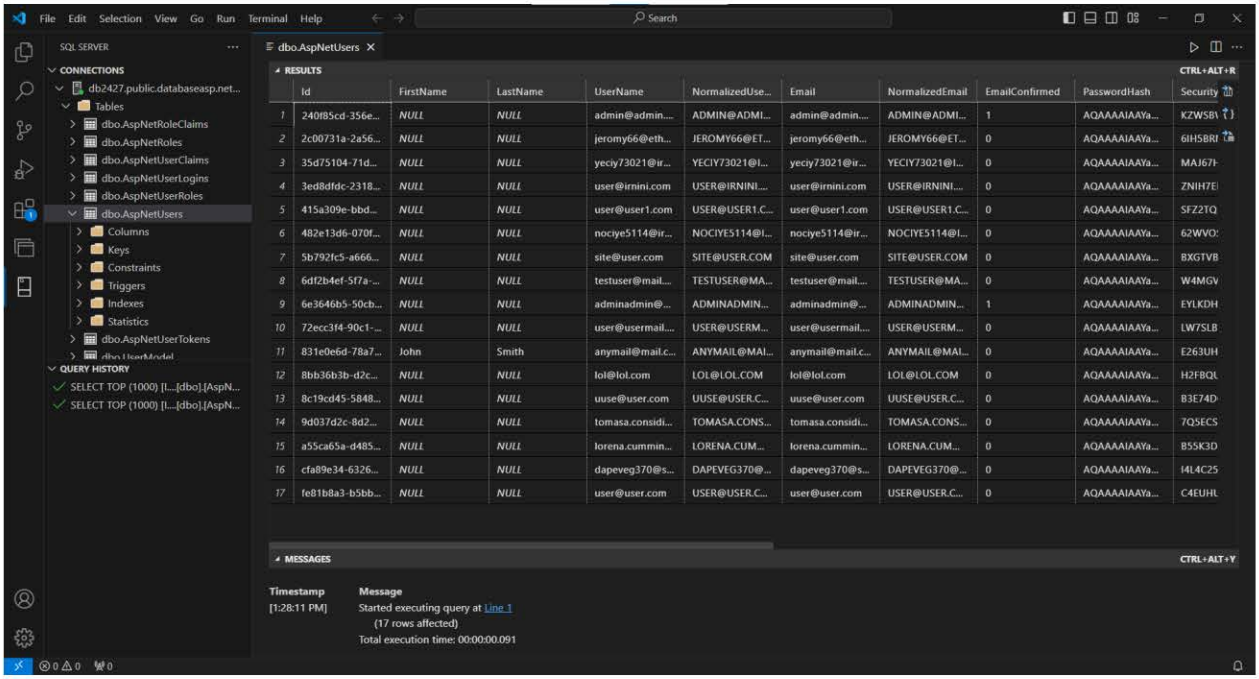


Рисунок 4.8 – Таблиця користувачів

Таблиця Role Зберігає і витягує інформацію про роль (рисунок 4.9) .

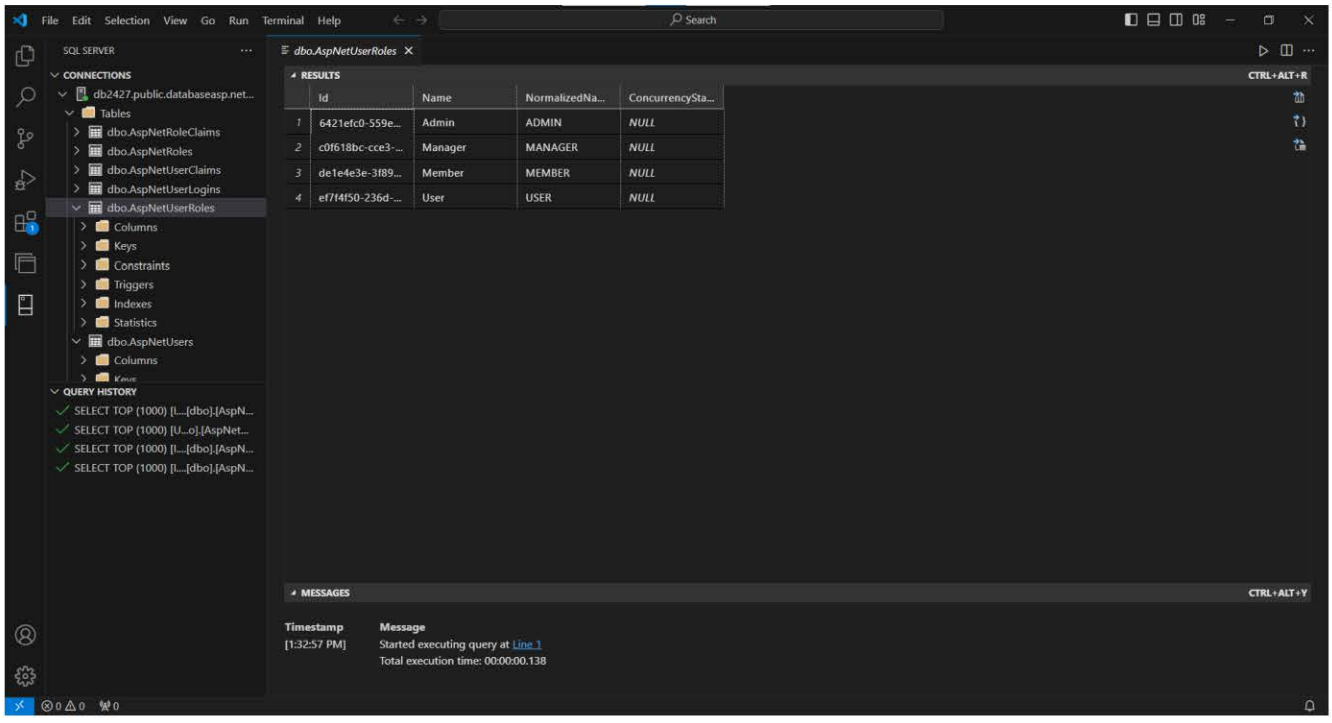


Рисунок 4.9 – Таблиця ролей користувачів

UserClaims Зберігає та отримує інформацію про вимоги користувача (наприклад, тип та значення вимоги).

UserLogins Зберігає та отримує інформацію про логін користувача (наприклад, зовнішній провайдер автентифікації). Приклад

UserRole Зберігає та отримує інформацію про те, які ролі призначено користувачам.[22]

Технологія ASP.NET Core Identity забезпечує безпеку даних, завдяки засобам захисту, таких як SSL/TLS для шифрування підключень, збереження паролів у хешованому вигляді використовуючи алгоритм PBKDF2, що робить їх захищеними від простого перегляду у випадку витоку бази даних. Хешування також включає сіль (salt), що ускладнює атаки словником (Chosen-plaintext attack) і атак на базі попередньо обчислених хешів (rainbow tables)., а також налаштування політик безпеки та аудиту для бази даних. Оптимізація продуктивності досягається за рахунок використання індексів для швидшого доступу до даних, оптимізації запитів та мінімізації їх кількості.

Інтеграція з базою даних за допомогою Entity Framework Core забезпечує зручний спосіб управління даними у веб-додатку з авторизацією. Основний компонент EF — контекст (DbContext), який виступає посередником між додатком і базою даних, забезпечуючи функціональність для виконання операцій CRUD (створення, читання, оновлення, видалення) та управління об'єктами доменної моделі. Контекст дозволяє відслідковувати зміни в об'єктах, управління транзакціями, конфігурацію зв'язків між сутностями та генерацію SQL-запитів для взаємодії з базою даних. Завдяки абстракції від конкретної СУБД, EF надає зручний та ефективний спосіб роботи з даними, що значно спрощує процес розробки додатків..Entity Framework (EF) створює моделі даних, використовуючи підхід, який називається "Code First". Це означає, що розробник спочатку створює класи .NET, які представляють сутності (таблиці) в базі даних, а потім EF використовує ці класи для створення або оновлення схеми бази даних(див. рисунок 4.10).

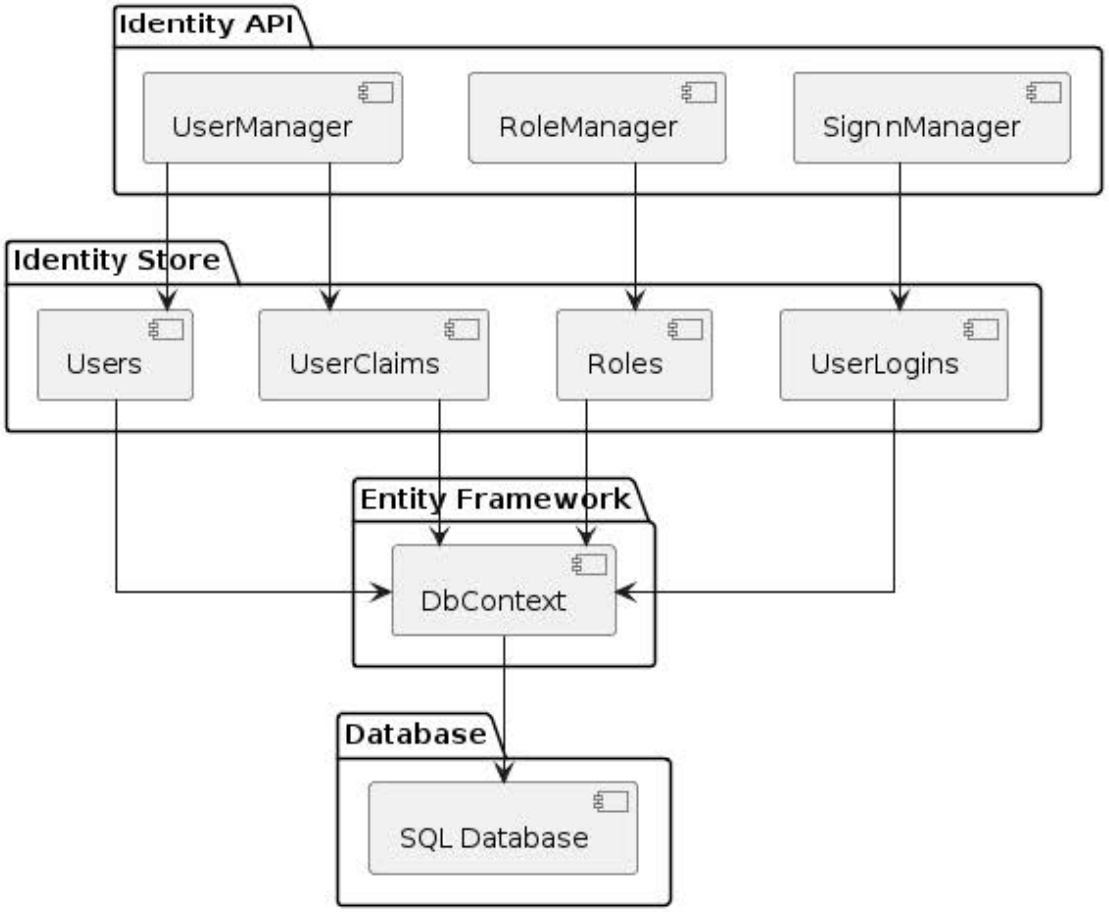


Рисунок 4.10 – Схема роботи Entity Framework

4.3 Забезпечення безпеки даних

Забезпечення безпеки даних є критичним аспектом розробки веб-додатка з авторизацією. Це включає захист даних користувачів, захист від несанкціонованого доступу, шифрування передавання даних, а також належне управління автентифікацією та авторизацією. Використання SSL/TLS для шифрування даних під час передавання між клієнтом і сервером гарантує, що всі дані, які передаються через мережу, захищені від перехоплення та маніпуляцій. Налаштування SSL-сертифікатів на сервері забезпечує HTTPS-з'єднання. Шифрування даних у базі даних, наприклад, за допомогою прозорого шифрування даних (TDE) у SQL Server, дозволяє зберігати чутливі дані у зашифрованому вигляді.

Для захисту автентифікації використовується хешування паролів з додаванням "солі" (salt) перед збереженням у базі даних, що забезпечує стійкість до атак методом перебору (brute-force). Впровадження дворівневої автентифікації (2FA) підвищує рівень безпеки користувачів, додаючи додаткові фактори автентифікації, такі як SMS-коди або мобільні додатки для генерації одноразових кодів. Захист від атак включає використання токенів CSRF (Cross-Site Request Forgery) для захисту від підробки запитів, екранування (escaping) вводу користувача для захисту від атак типу XSS (Cross-Site Scripting), а також використання параметризованих запитів та ORM для запобігання SQL-ін'єкціям.

Управління доступом здійснюється за допомогою ролей і прав доступу для користувачів до різних ресурсів додатка, що реалізується за допомогою ASP.NET Core Identity. Визначення політик безпеки та використання атрибутів авторизації контролюють доступ до контролерів і дій. Моніторинг та аудит включають впровадження систем логування для відстеження дій користувачів та подій у системі, а також використання інструментів моніторингу, таких як Serilog та Application Insights, для виявлення аномалій та потенційних загроз. Регулярне проведення аудитів безпеки для виявлення вразливостей і використання інструментів для автоматизованого тестування безпеки.

Забезпечення безпеки даних у веб-додатку з авторизацією вимагає комплексного підходу, що включає шифрування даних, захист автентифікації, захист від атак, управління доступом, а також моніторинг та аудит. Використання Identity та методів безпеки дозволяє створити надійний та захищений додаток, що відповідає вимогам сучасних користувачів і захищає їхні дані від потенційних загроз.

4.4 Висновки до четвертого розділу

Розробка системи авторизації включає багато етапів, починаючи від налаштування середовища розробки і закінчуючи розгортанням сайту на хмарному сервісі. Важливими

кроками є інтеграція з базою даних, створення моделей користувачів та ролей, а також впровадження політик доступу.

Використання сервісів Identity дозволяє забезпечити ефективне управління користувачами та їх ролями, а реєстрація інших методів авторизації, таких як OAuth через Google, Microsoft або Github, розширює можливості вхідного процесу для користувачів.

Класи UserManager і RoleManager виконують важливу роль у керуванні користувачами та ролями, дозволяючи легко створювати, оновлювати та видаляти облікові записи і ролі. Політики доступу надають можливість гнучко контролювати доступ до ресурсів системи, забезпечуючи високий рівень безпеки. Крім того, використання інструментів для роботи з токенами та надсиланням електронних листів для підтвердження аккаунтів додає до системи додаткову функціональність та безпеку.

Обираючи SQL Server для середовища Microsoft, розробники забезпечують високу продуктивність та розширені функції. Використання Entity Framework Core спрощує цей процес, дозволяючи ефективно взаємодіяти з базою даних та управляти даними користувачів та їх ролями. Робота з міграціями бази даних допомагає створювати та змінювати структуру бази даних з легкістю, забезпечуючи необхідну функціональність для додатка.

Забезпечення безпеки даних у веб-додатках включає в себе широкий спектр заходів. Від шифрування передавання даних за допомогою SSL/TLS до шифрування даних у базі даних, як TDE у SQL Server. Використання хешування паролів з сіллю та дворівневої автентифікації підвищує рівень безпеки користувачів. Захист від атак, таких як CSRF і XSS, разом з управлінням доступом та моніторингом та аудитом, створює комплексний захист даних у додатку.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА АНАЛІЗ БЕЗПЕКИ

5.1 Тестування системи авторизації

Тестування системи авторизації в ASP.NET Core Identity включає декілька кроків, що охоплюють різні аспекти функціональності, безпеки та інтеграції. Нижче наведено покроковий процес тестування:

1. Введення правильних облікових даних (ім'я користувача та пароль) для кожного тестового користувача та перевірка успішного входу (рисунок 5.1).

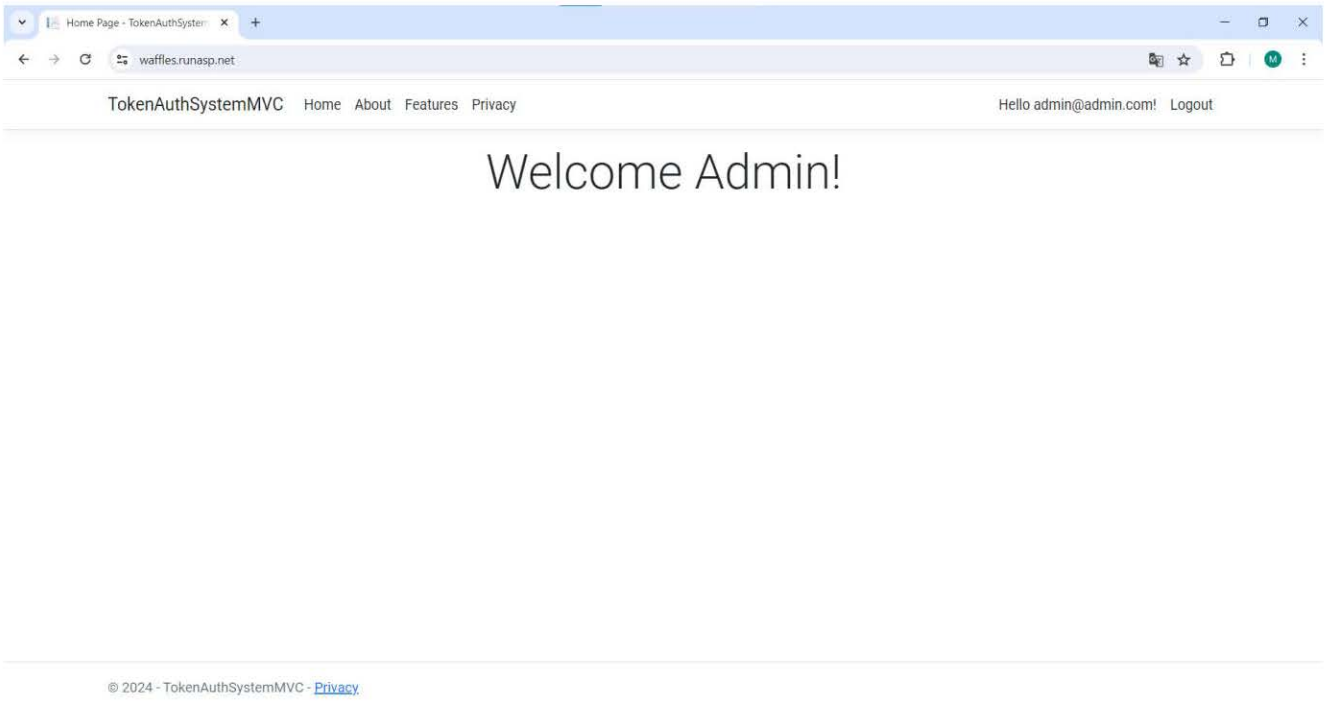


Рисунок 5.1 - Вхід з правильними обліковими даними як адміністратор

2. Введення неправильних облікових даних та перевірка повідомлення про помилку (рисунок 5.2).

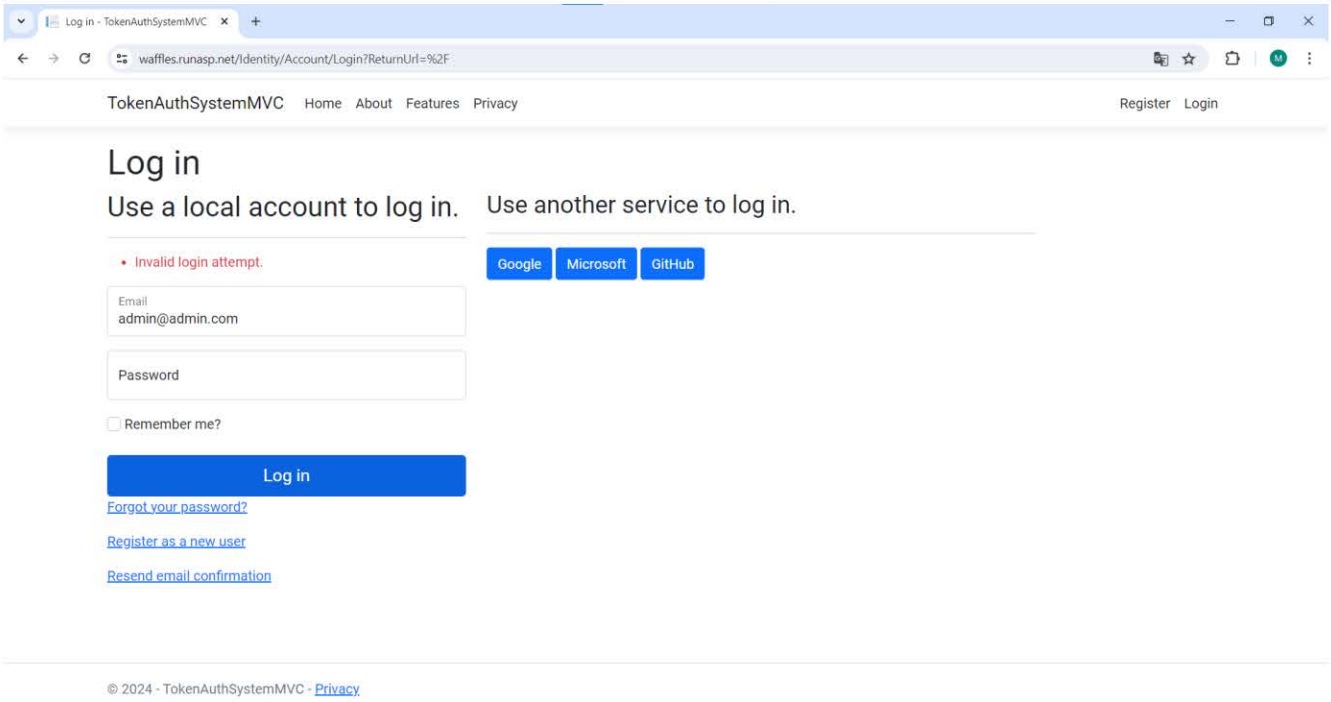


Рисунок 5.2 - Вхід з неправильними обліковими даними

3. Перевірка реакції системи на порожні поля форми входу (рисунок 5.3).

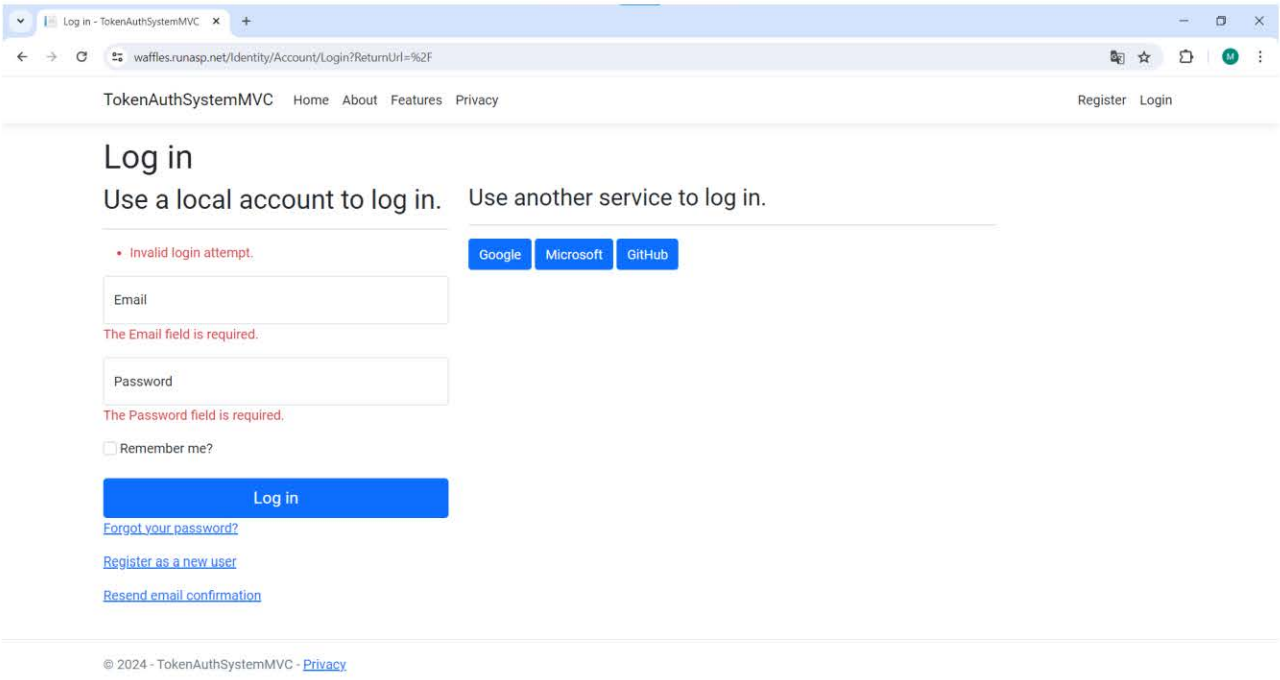


Рисунок 5.3 – Спроба авторизації в систему без введення даних

4. Тестування авторизації ролей. Вхід під користувачем з роллю адміністратора та перевірка доступу до адміністративних розділів додатка (рисунок 5.4).

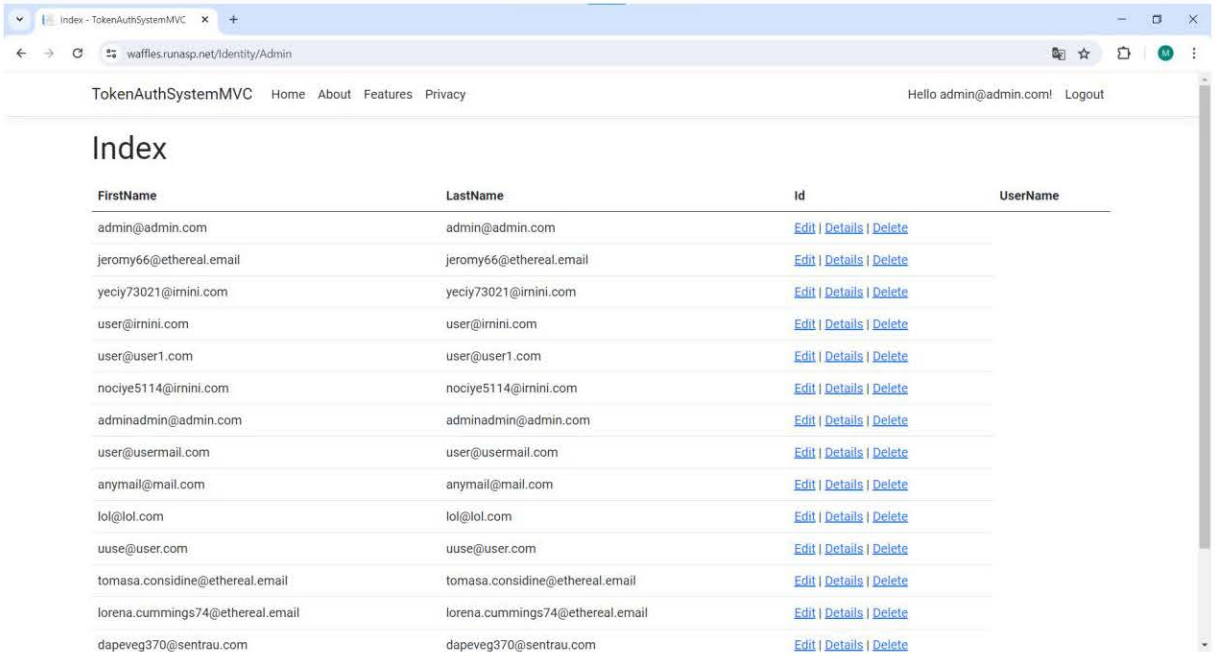


Рисунок 5.4 – Вхід до панелі адміністрування

5. Вхід під звичайним користувачем та перевірка доступу до ресурсів, доступних лише для цієї ролі (рисунок 5.5 – 5.8).

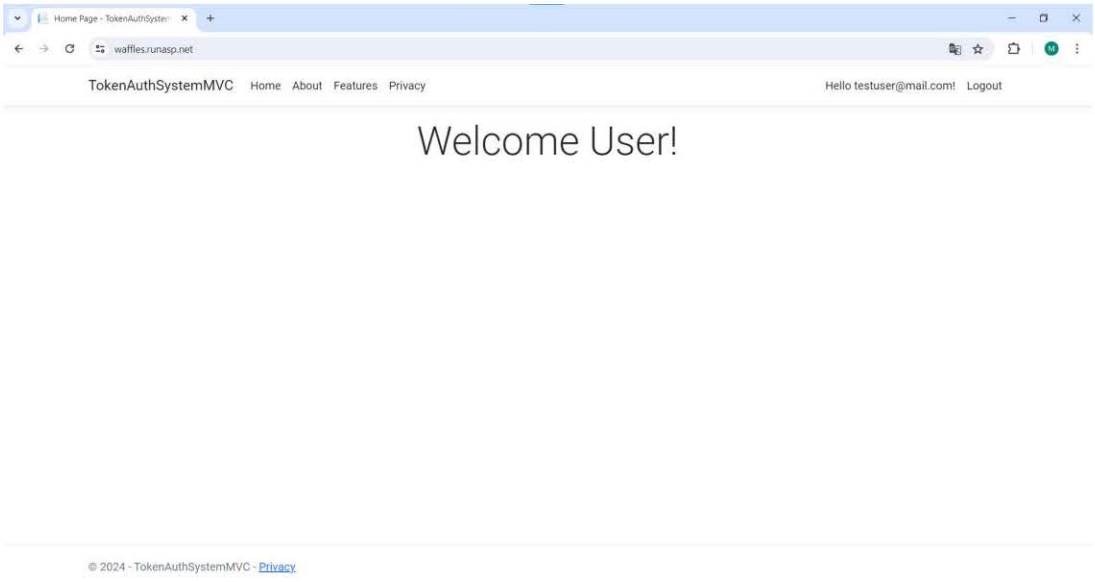


Рисунок 5.5 – Головна сторінка для користувача без прав адміністратора

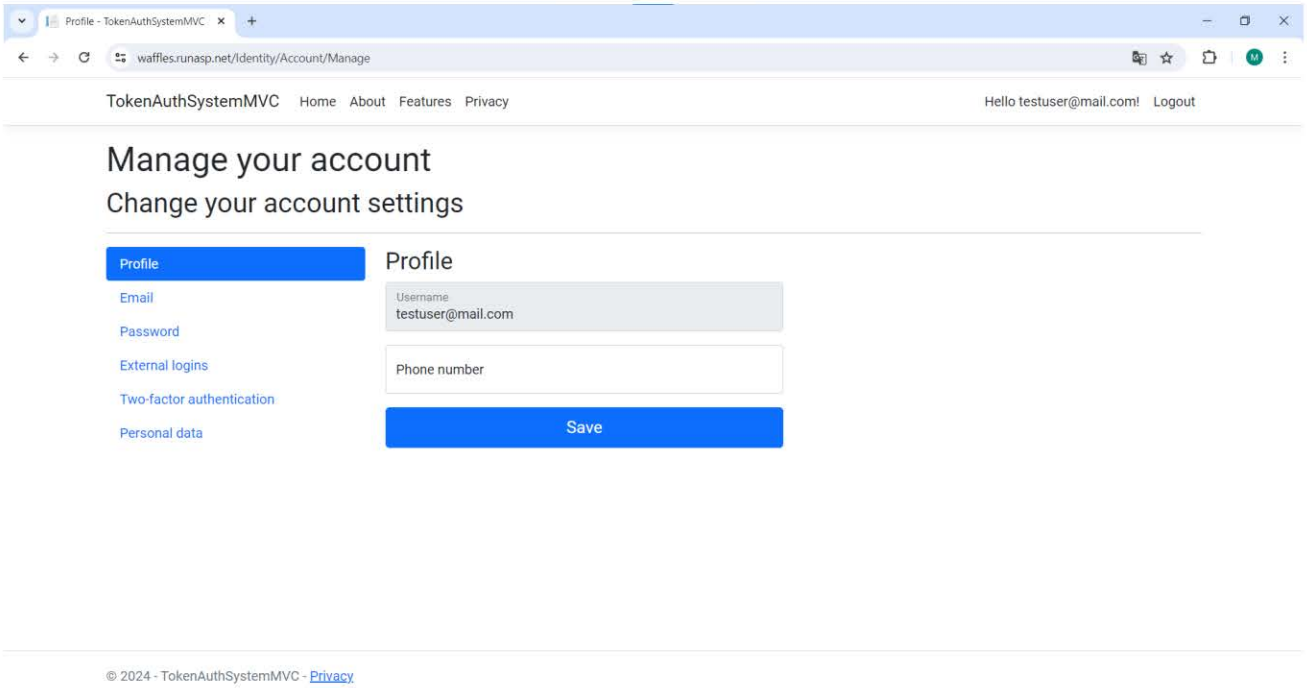


Рисунок 5.6 – Сторінка профілю користувача без прав адміністратора

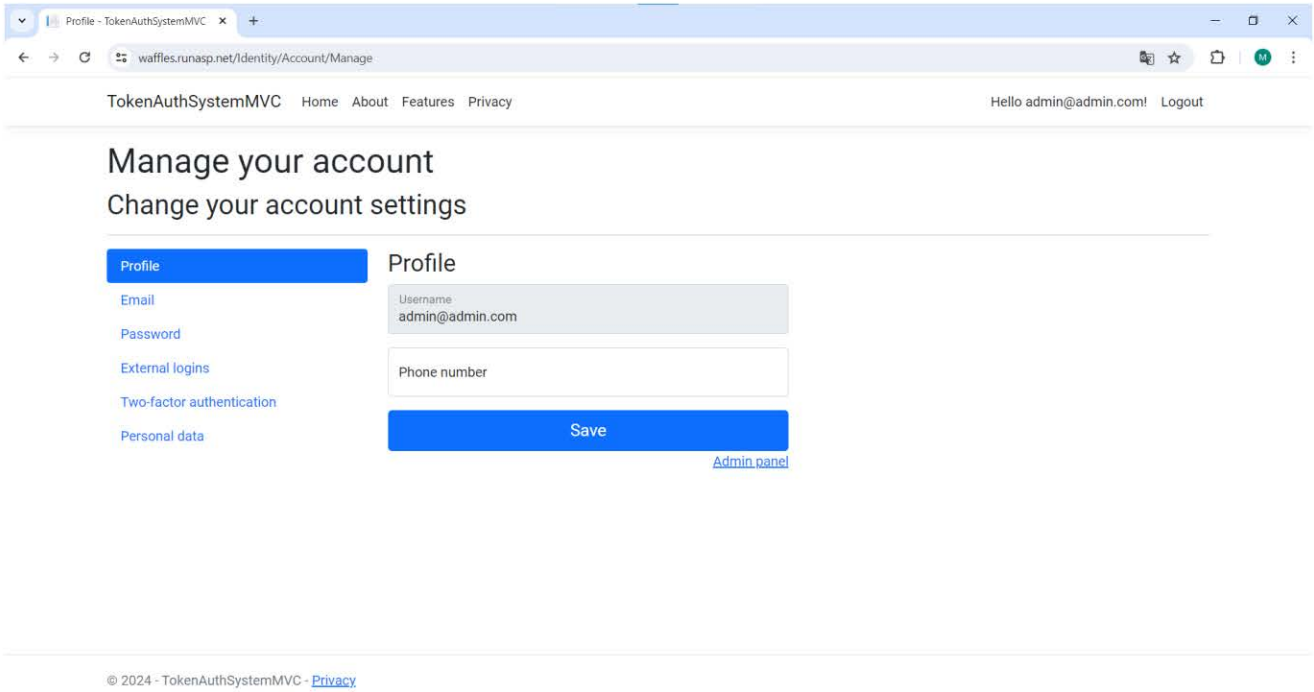


Рисунок 5.7 – Сторінка профілю користувача з правами адміністратора

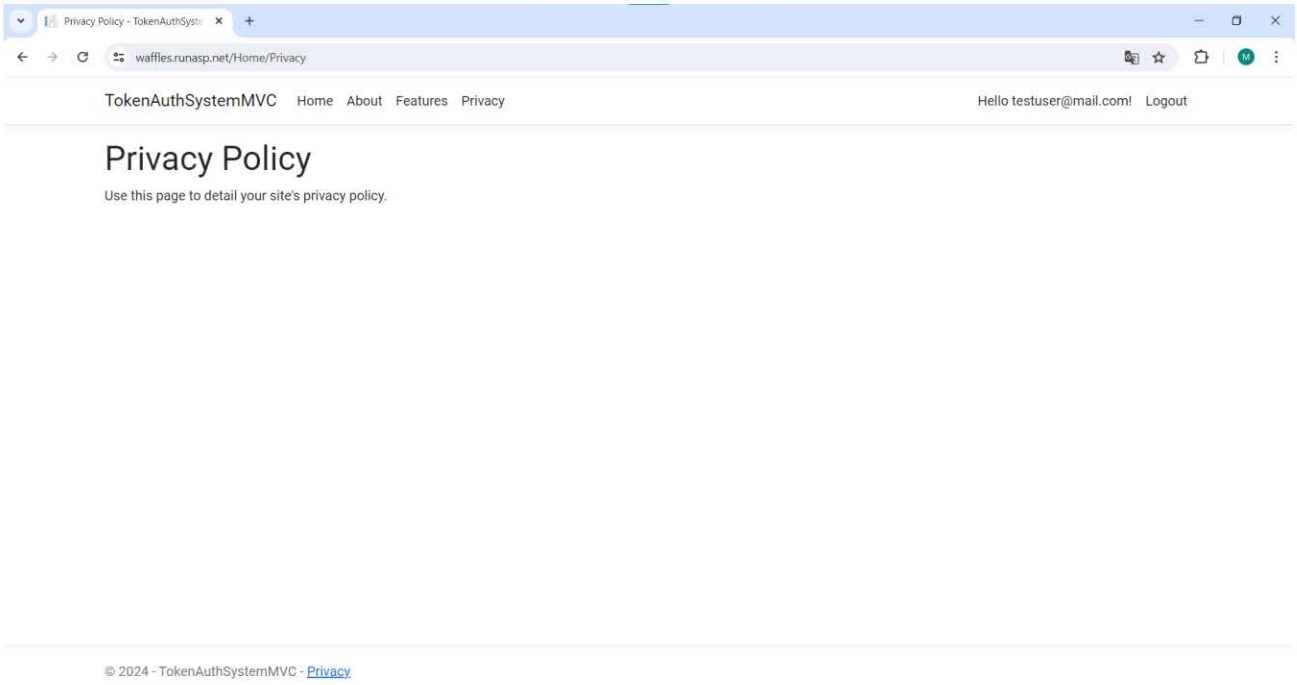


Рисунок 5.8 – Сторінка політики конфіденційності

6. Вхід під користувачем без прав доступу та перевірка відсутності доступу до захищених розділів (рисунок 5.9). Спроба доступу до захищених ресурсів за допомогою прямого введення URL.

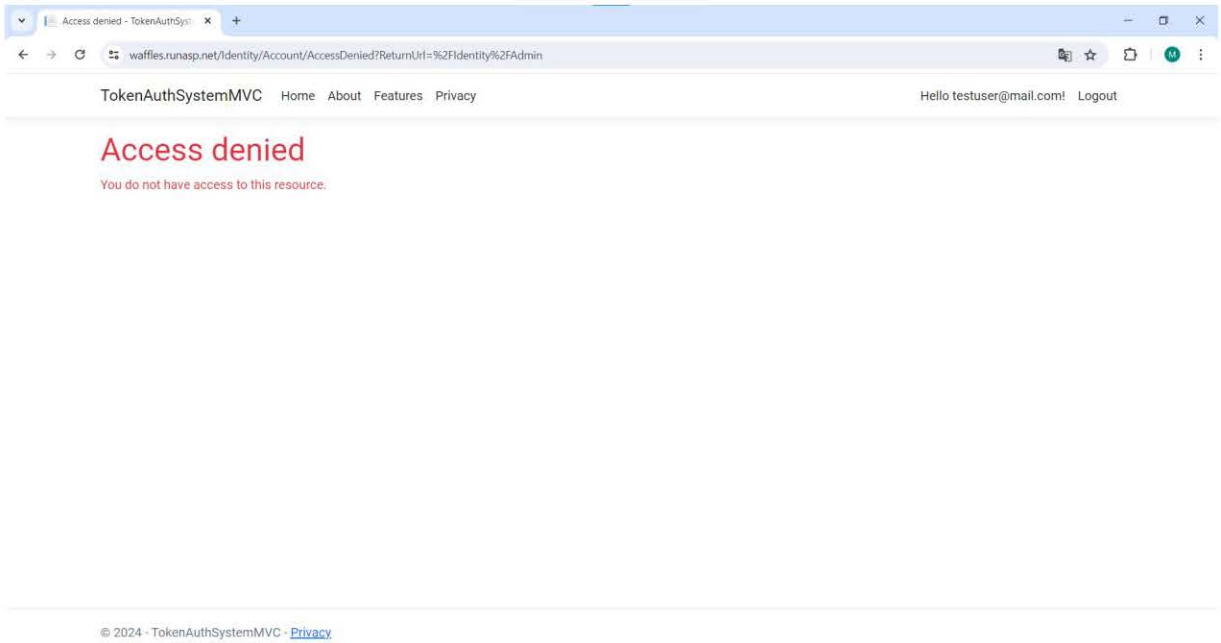


Рисунок 5.9 – Спроба відкрити сторінку без прав доступу

3. Тестування панелі адміністрування (рисунок 5.10 – 5.13).

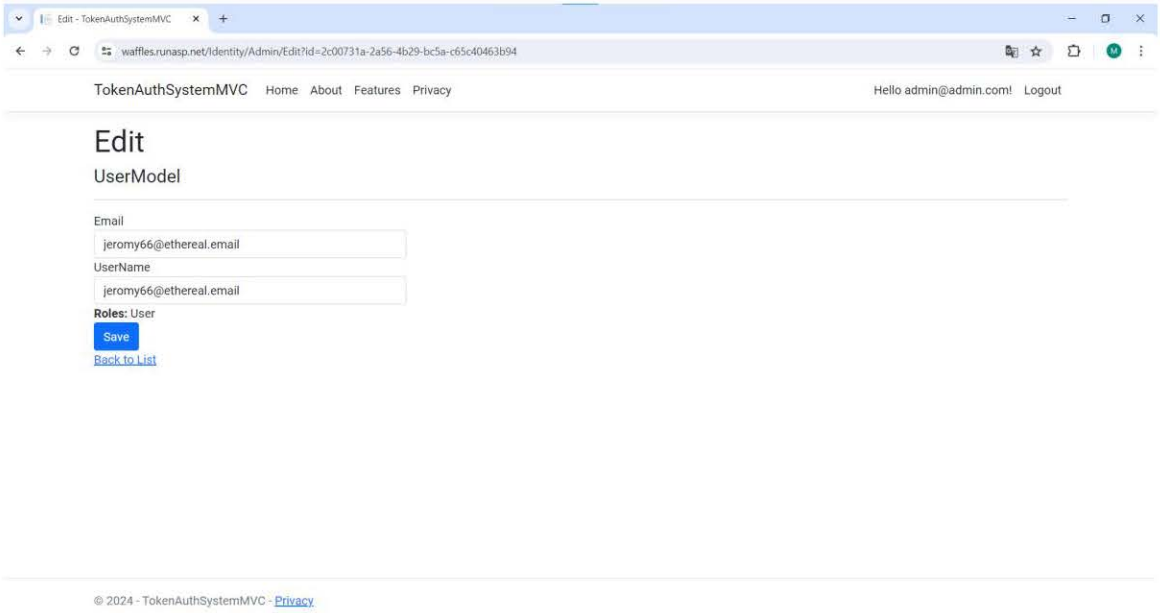


Рисунок 5.10 – Сторінка для зміни даних користувача

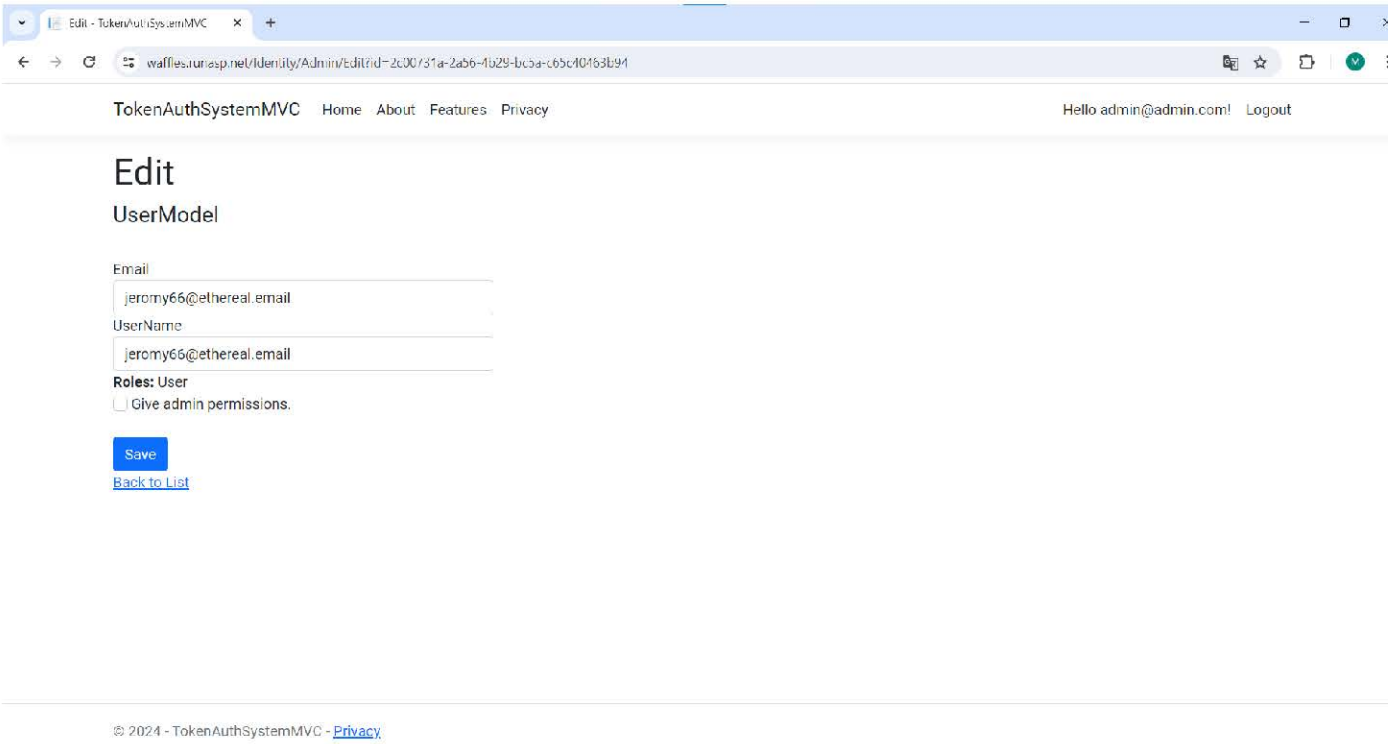


Рисунок 5.11 – Сторінка для зміни даних користувача

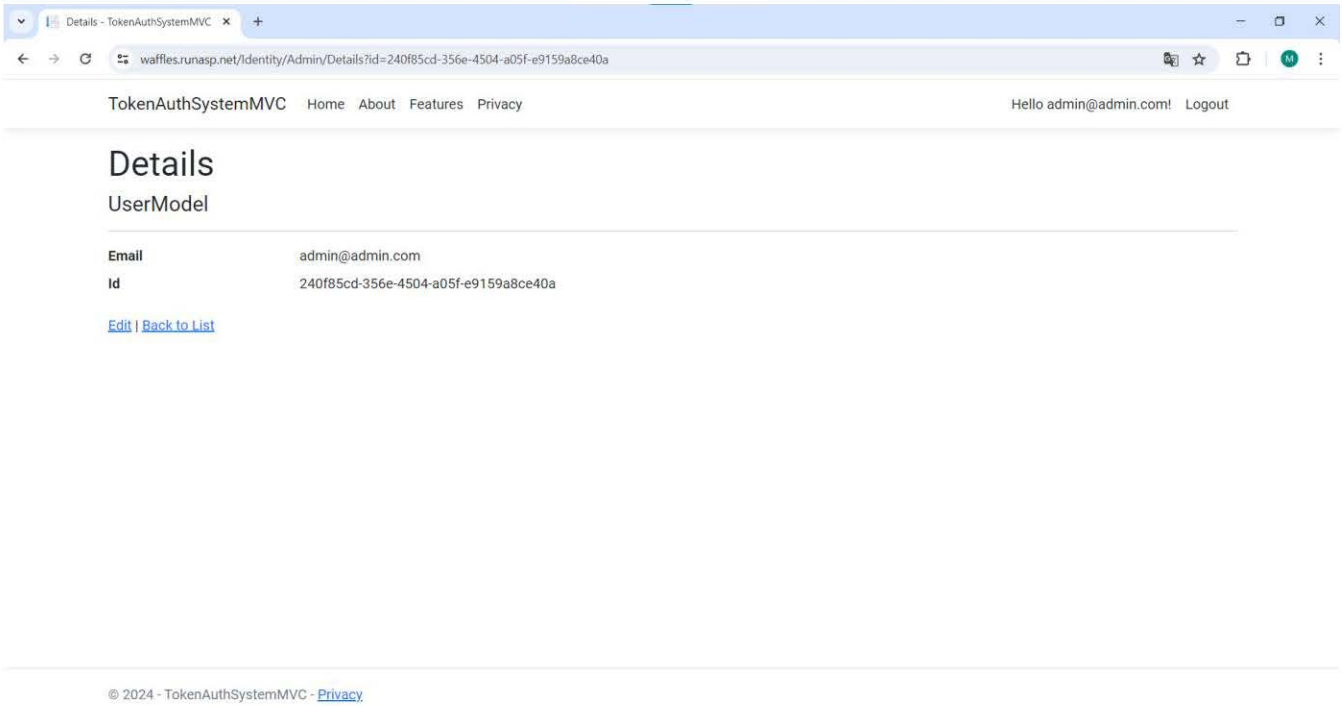
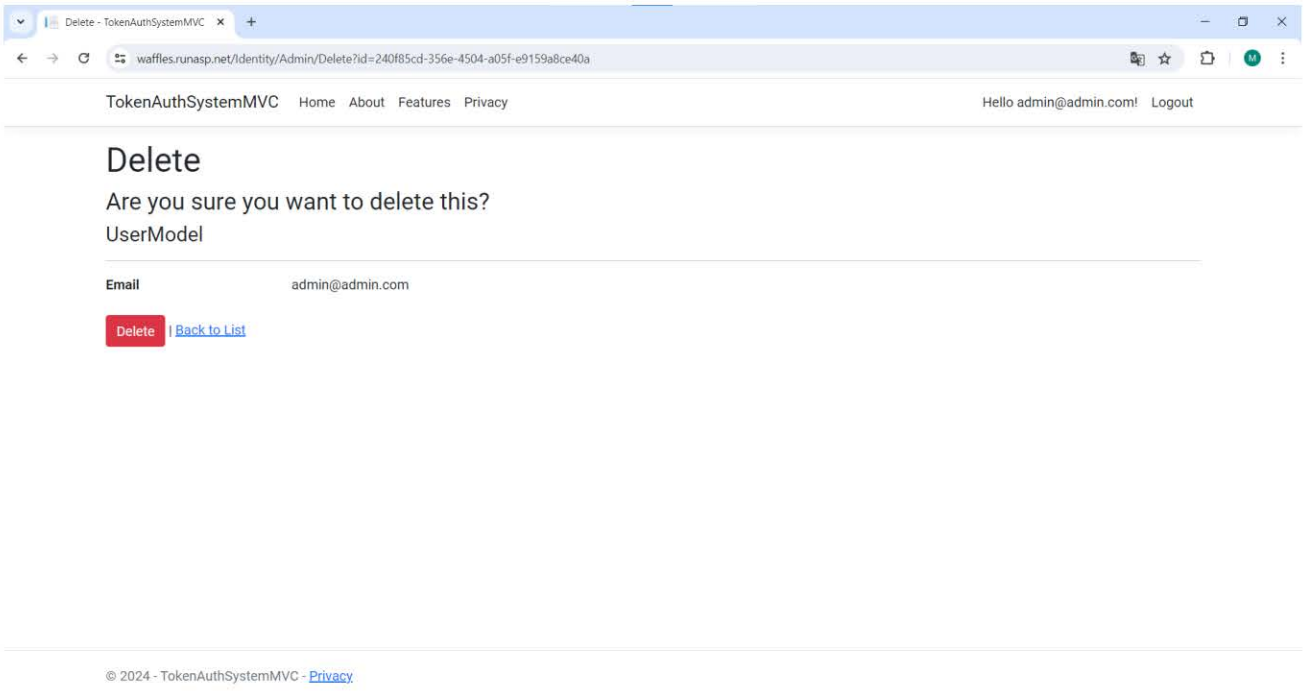


Рисунок 5.12 – Сторінка для перегляду даних користувача



Сторінка 5.13 - – Сторінка для видалення даних користувача

4. Тестування реєстрації в додатоку, використовуючи пошту або сторонні сервіси(рисунок 5.14 – 5.16).

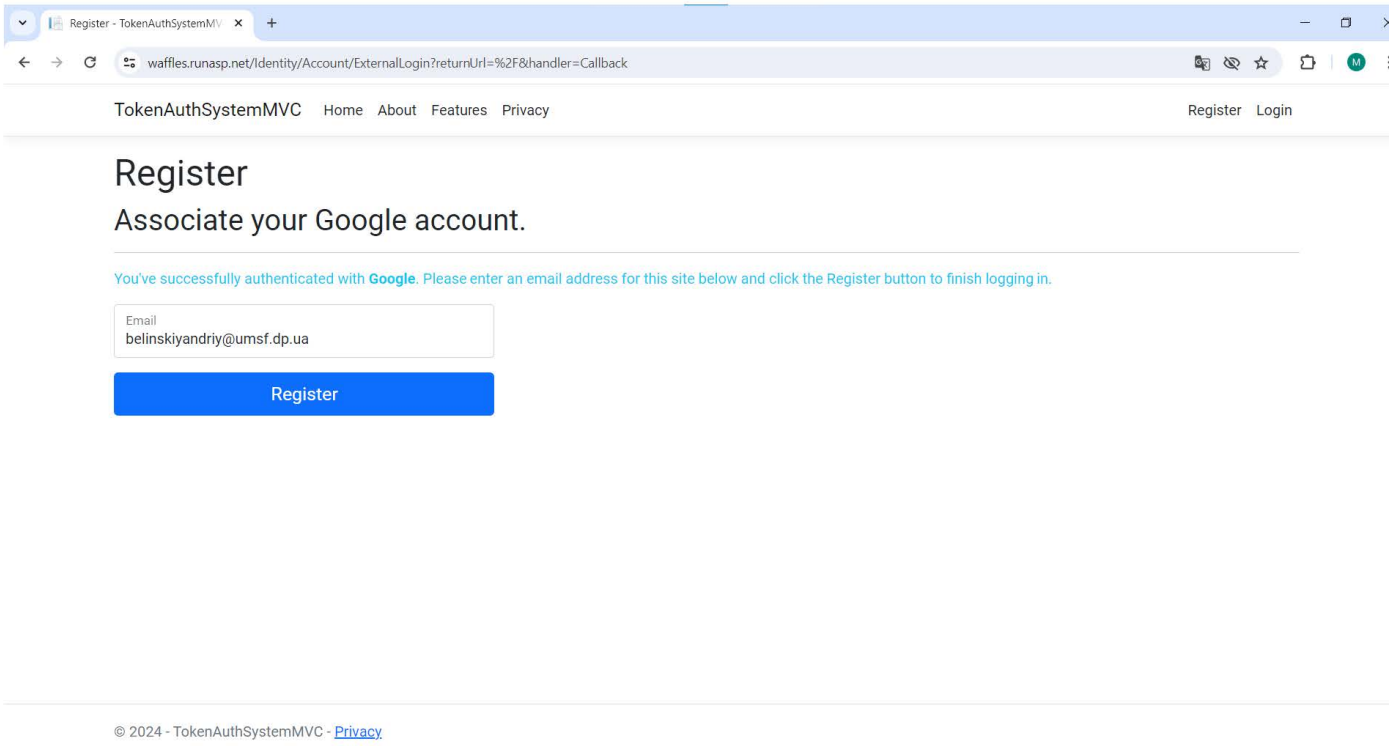


Рисунок 5.14 – Реєстрація використовуючи аккаунт Google

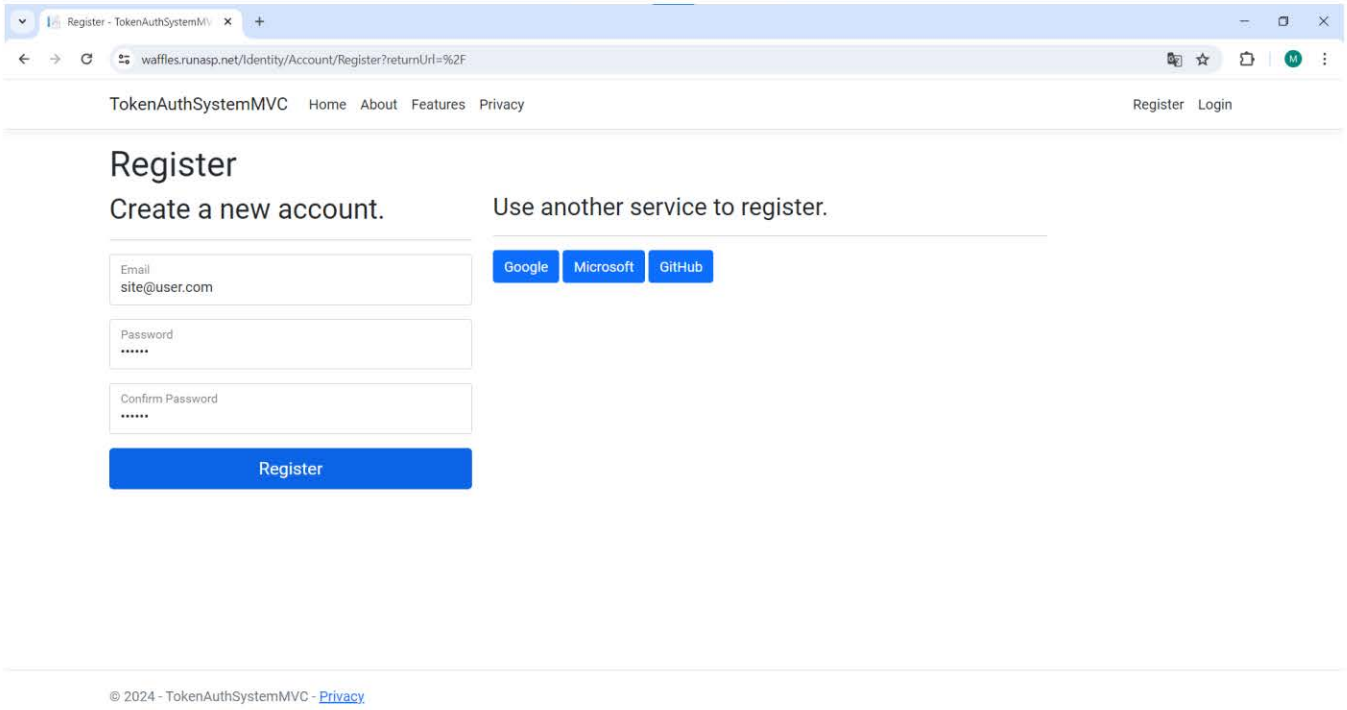


Рисунок 5.15 – Реєстрація використовуючи пошту.

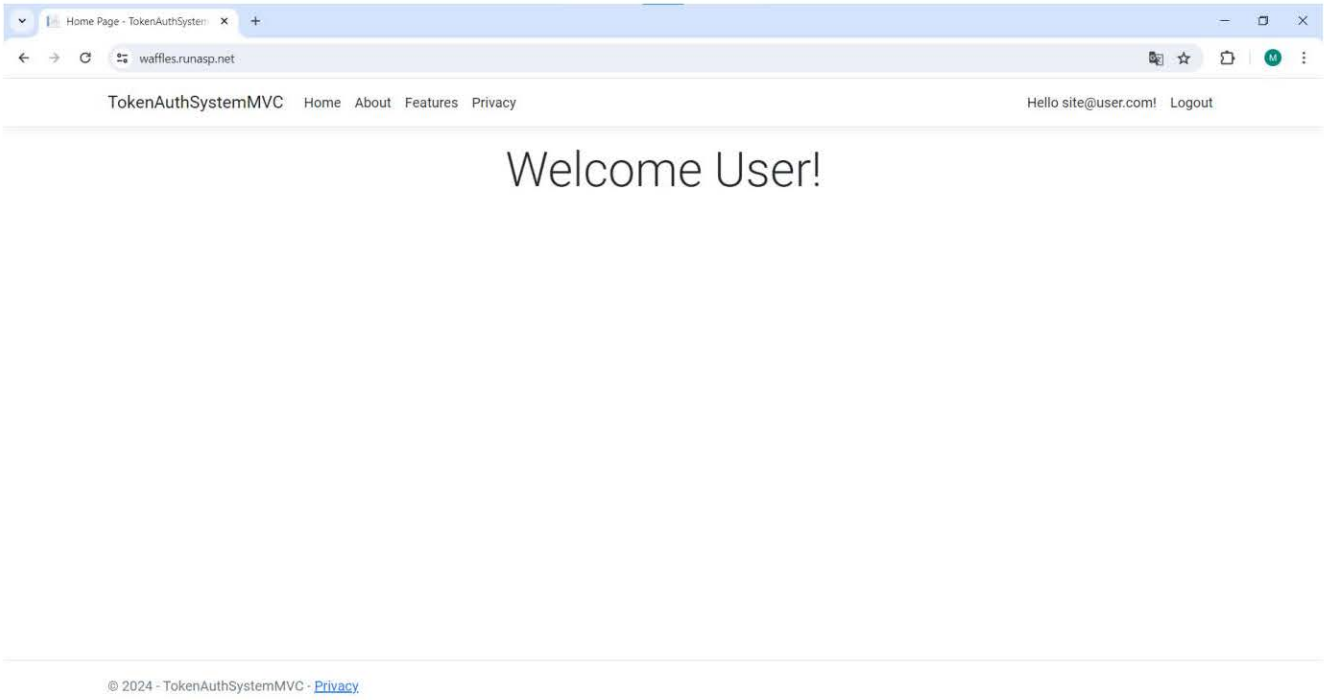


Рисунок 5.16 – Успішна реєстрація використовуючи пошту.

Для тестування були використані методи модульного тестування з використанням бібліотеки xUnit.NET, яка є популярним фреймворком для тестування .NET додатків та ручні тести (manual tests) або тестами на ручне тестування (manual acceptance tests). У цих тестах людина вручну перевіряє різні аспекти функціональності, використання та зовнішнього вигляду веб-сайту. Такі тести особливо корисні для перевірки користувацького досвіду, специфікаційних вимог та взаємодії з реальними користувачами.. Для виконання HTTP-запитів до веб-додатків ASP.NET Core була використана клієнтська бібліотека HttpClient. Окрім того, для деяких тестів були використані бібліотеки серіалізації та десеріалізації JSON для обробки відповідей від веб-сервера.

Тести були структуровані з використанням атрибутів Fact та Task, що вказують на те, що методи є фактичними тестами, а також що вони є асинхронними і повертають об'єкти типу Task.

Для перевірки результатів тестів використовувалися методи Assert, які надаються бібліотекою xUnit.NET. Наприклад, Assert.True для перевірки на істинність певного виразу, Assert.Equal для перевірки рівності двох значень тощо.

Також, в тестах були використані різні методи, що надаються об'єктами клієнта HttpClient, такі як PostAsJsonAsync для відправлення POST-запитів з JSON-даними, GetAsync для відправлення GET-запитів тощо.

Загалом, ці методи тестування дозволяють комплексно оцінити систему авторизації, забезпечити її надійну та безпечну роботу, а також покращити користувацький досвід.

5.2 Аналіз вразливостей

Однією з основних вразливостей є SQL Injection, яка виникає при використанні введених користувачем даних у SQL-запитах без належної фільтрації або екранування.

Використання параметризованих запитів та ORM, таких як Entity Framework (EF) Core, допомагає запобігти цій загрозі. Інша поширена вразливість – Cross-Site Scripting (XSS), що виникає при вставці шкідливого коду у веб-сторінки, який може викрадати сесії або поширювати шкідливе ПЗ. Для захисту від XSS необхідно валідувати та екранувати всі введені дані перед їх відображенням, використовуючи вбудовані механізми ASP.NET Core, такі як HtmlEncoder.

Cross-Site Request Forgery (CSRF) є ще однією серйозною загрозою, коли зловмисник змушує користувача виконувати небажані дії на сайті, на якому він аутентифікований. Захист від CSRF включає додавання маркерів CSRF до всіх форм і AJAX-запитів та перевірку їх на сервері. Витік інформації про помилки може надати зловмисникам цінну інформацію про структуру системи або вразливості.

Недостатня валідація даних є ще однією потенційною вразливістю, яка може призвести до різних атак. Використання валідаторів для всіх введених даних і регулярних виразів для забезпечення правильного формату є необхідним. Недостатній контроль доступу може дозволити користувачам отримати доступ до ресурсів, до яких вони не повинні мати доступ. Використання ролей та політик авторизації в ASP.NET Core Identity допомагає налаштовувати права доступу детально. Нарешті, використання застарілих або небезпечних бібліотек може створити додаткові ризики. Регулярне оновлення всіх бібліотек і компонентів та перевірка наявності відомих вразливостей є необхідним. Отже, захист системи авторизації вимагає постійного моніторингу та оновлення для протидії новим загрозам.

5.3 Оцінка ефективності системи

Оцінка ефективності системи авторизації на ASP.NET Core Identity передбачає аналіз її продуктивності, безпеки, масштабованості та інтеграційних можливостей. Ефективність системи визначається її здатністю забезпечувати надійний захист даних користувачів через багатофакторну аутентифікацію (MFA) та підтримку різних

протоколів, таких як OAuth2 і OpenID Connect, що гарантує гнучкість і сумісність з іншими системами. Продуктивність системи оцінюється через її здатність обробляти високі навантаження з мінімальним часом відгуку, що забезпечується оптимізованим управлінням сесіями та кешуванням.

Безпека системи є одним з найважливіших показників її ефективності, досягнута завдяки вбудованим механізмам захисту від поширених вразливостей, таких як SQL Injection, XSS, і CSRF, а також використанню сучасних методів шифрування для захисту даних під час передачі та зберігання. Масштабованість системи дозволяє їй адаптуватися до зростання кількості користувачів і запитів без значних втрат продуктивності, що реалізується через підтримку розподіленої архітектури та можливості вертикального і горизонтального масштабування.

Інтеграційні можливості ASP.NET Core Identity включають безшовну інтеграцію з популярними зовнішніми сервісами аутентифікації та корпоративними системами, що сприяє підвищенню зручності використання та адміністрування. Крім того, система підтримує детальне логування та аудит подій аутентифікації, що забезпечує прозорість і можливість оперативного реагування на інциденти безпеки.

Таким чином, ефективність системи авторизації на ASP.NET Core Identity визначається її здатністю забезпечувати високий рівень безпеки, продуктивності та масштабованості, а також інтеграційних можливостей, що відповідають сучасним вимогам розробки веб-додатків.

4.4 Висновки до п’ятого розділу

Після здійснення тестування системи авторизації в ASP.NET Core Identity, аналізу вразливостей та оцінки ефективності можна зробити наступні висновки.

Початкові результати тестування свідчать про те, що система демонструє високу робочу ефективність. Успішний вхід з правильними обліковими даними, а також вдала обробка неправильних введень і порожніх полів форми входу підтверджують гнучкість

та надійність системи. Детальне тестування авторизації ролей і реєстрації через пошту чи сторонні сервіси також підтверджує функціональну повноту системи.

Використання методів модульного тестування разом з бібліотекою xUnit.NET дозволило детально оцінити різні аспекти функціональності та безпеки системи. Додатково, ручні тести сприяли перевірці користувацького досвіду та взаємодії з реальними користувачами, що є важливим аспектом в процесі розробки веб-додатків.

Проте, аналіз вразливостей розкриває ряд потенційних загроз для безпеки. SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та інші вразливості можуть стати серйозною проблемою, якщо не будуть вжиті відповідні заходи захисту. Тому необхідно вжити заходів для ефективного контролю та фільтрації введених даних, а також застосувати сучасні методи шифрування для захисту інформації.

З оцінки ефективності випливає, що система має високий рівень продуктивності, безпеки, масштабованості та інтеграційних можливостей.

Застосування багатофакторної аутентифікації та підтримка різних протоколів, таких як OAuth2 і OpenID Connect, дозволяють системі гнучко взаємодіяти з іншими сервісами та забезпечувати високий рівень сумісності.

Продуктивність системи вище завдяки оптимізованим управлінням сесіями та кешуванням, а її безпеку підтверджує вбудовані механізми захисту від поширених вразливостей. Масштабованість системи забезпечує її здатність адаптуватися до зростання кількості користувачів та запитів. Інтеграційні можливості забезпечують легкість впровадження та взаємодії з іншими системами, що полегшує адміністрування та користування системою.

ВИСНОВКИ

З метою підвищення безпеки та контролю доступу до веб-сайту, було розроблено систему авторизації. Ця система забезпечує надійний механізм контролю доступу, використовуючи сучасні технології та підходи, такі як багатофакторна аутентифікація, ролі та політики авторизації, а також шифрування даних.

Актуальність дослідження полягає у зростаючій потребі забезпечення безпеки в умовах широкого використання обчислювальної техніки та штучного інтелекту, а також у необхідності захисту конфіденційних даних та запобігання несанкціонованому доступу до систем управління. Впровадження системи авторизації дозволяє автоматизувати та вдосконалити процеси контролю доступу, що є важливим у різних галузях.

Таким чином, в результаті виконання даної роботи була досягнута її мета – забезпечення безпечної та ефективної роботи веб-сайту через впровадження надійної системи авторизації. Було виконано наступне:

- проаналізовано методи та технічні засоби, що застосовуються для реалізації системи авторизації,
- розроблено вимоги до програмного забезпечення для реалізації системи авторизації на основі аналізу переваг та недоліків існуючих рішень,
- спроектовано та розроблено нову систему авторизації на основі потреб користувачів.

У результаті дослідження були визначені методи та технічні засоби для реалізації системи авторизації. Було сформульовано вимоги до програмного забезпечення та здійснено проектування і розробку системи. У рамках практичної частини проекту було успішно реалізовано систему авторизації, що включала багатофакторну аутентифікацію та детальні політики доступу.

Під час тестування системи авторизації було підтверджено її відповідність очікуванням, зокрема щодо ефективного контролю доступу та стабільної роботи в різних

умовах. Отримані результати мають практичне значення, оскільки сприяють підвищенню безпеки та надійності роботи системи підтримки прийняття рішень.

Отже, розроблена система авторизації є важливим кроком у покращенні безпеки та ефективності управління доступом до веб-сайту. Вона використовує передові технології та забезпечує надійну і точну роботу, що сприяє захисту конфіденційних даних і забезпечує безпеку співробітників та майна підприємств.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Коноваленко І.В., Марущак П.О., Савків В. Розділ 3. Класи: опис об'єктів // Програмування мовою C# 7.0 С. 73

2. Andrew Troelsen, Phil Japikse Introducing Entity Framework Core // Pro C# 10 with .NET 6 Foundational Principles and Practices in Programming C. 910

3. Christian Nagel, Professional ASP.NET Core // C# 6 and .NET Core 1.0 - c. 1900

4. [Електронний ресурс] – How to Secure the Web: A Comprehensive Guide to Authentication Strategies for Developers - https://dev.to/ma7moud3bas/how-to-secure-the-web-a-comprehensive-guide-to-authentication-strategies-for-developers-48od

5. [Електронний ресурс] – Overview of ASP.NET Core - https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0

6. [Електронний ресурс] – JWT Authentication And Authorization In .NET 6.0 With Identity Framework - https://www.c-sharpcorner.com/article/jwt-authentication-and-authorization-in-net-6-0-with-identity-framework/

7. [Електронний ресурс] – Entity Framework 6 - https://learn.microsoft.com/en-us/ef/ef6/

8. [Електронний ресурс] – New in .NET 8: ASP.NET Core Identity and How to Implement It - https://www.telerik.com/blogs/new-net-8-aspnet-core-identity-how-implement

9. [Електронний ресурс] – Relationships, navigation properties, and foreign keys - https://learn.microsoft.com/en-us/ef/ef6/fundamentals/relationships

10. [Електронний ресурс] – What is the Microsoft identity platform? - https://learn.microsoft.com/en-us/entra/identity-platform/v2-overview

11. [Електронний ресурс] – Token-based authentication and authorization (JWT Bearer) with ASP.NET Core - https://rafaelneto.dev/en/blog/aspnet-core-jwt-bearer-authentication-authorization/

12. [Електронний ресурс] – Google external login setup in ASP.NET Core - https://learn.microsoft.com/en-us/aspnet/core/security/authentication/social/google-logins?view=aspnetcore-8.0

13. [Електронний ресурс] – .NET 8: What's New for Authentication and Authorization - https://auth0.com/blog/whats-new-dotnet8-authentication-authorization/

14. [Електронний ресурс] – ASP.NET overview - https://learn.microsoft.com/en-us/aspnet/overview

15. [Електронний ресурс] – Adding ASP.NET Core Identity to an Existing Application: A Comprehensive Guide to User Authentication - https://www.webdevtutor.net/blog/adding-aspnet-core-identity-to-existing-application

16. [Електронний ресурс] – Add Authentication to Your ASP.NET Core MVC Application - https://auth0.com/blog/add-authentication-aspnet-core-mvc/

17. [Електронний ресурс] – JWT Validation and Authorization in ASP.NET Core - https://devblogs.microsoft.com/dotnet/jwt-validation-and-authorization-in-asp-net-core/

18. [Електронний ресурс] – Asp Net Core — Rest API Authorization with JWT (Roles Vs Claims Vs Policy) - Step by Step - https://mohamadlawand.medium.com/asp-net-core-rest-api-authorization-with-jwt-roles-vs-claims-vs-policy-step-by-step-f3d0838a70b

19. [Електронний ресурс] – Adding two-factor authentication to an application using ASP.NET Identity - https://devblogs.microsoft.com/dotnet/adding-two-factor-authentication-to-an-application-using-asp-net-identity/

20. [Електронний ресурс] – Implement ASP.NET Core Identity Getting Started - https://procodeguide.com/programming/aspnet-core-identity/

21. [Електронний ресурс] – Common web application architectures - https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

22. [Електронний ресурс] – Identity model customization in ASP.NET Core - https://learn.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-8.0

23. [Електронний ресурс] – Develop ASP.NET Core MVC apps - https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/develop-asp-net-core-mvc-apps

24 [Електронний ресурс] – Custom storage providers for ASP.NET Core Identity - https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-custom-storage-providers?view=aspnetcore-8.0

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