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 2024р.

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)
спеціальність *125 Кібербезпека*
(код і назва спеціальності)
освітній рівень *магістр*
освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Методи безпеки в технології .NET»

Виконавець: студентка ІІ курсу, групи КБм-21

_____ **Аліна КЛЕЩЕНКО**
(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*

Здобувача КБм-21 (група) Клещенко Аліна Олексіївна (прізвище ім'я по-батькові)

Тема дипломної роботи Методи безпеки в технології .NET

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 3 від 25.11.2023

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес дослідження методів безпеки в технології .NET.
Предмет досліджень	Методи безпеки в технології .NET.
Мета	Дослідження методів безпеки в технології .NET і розробка комплексної системи захисту для вебдодатка.
Вихідні дані для проведення роботи	Методи безпеки в технології .NET.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна Удосконалення методів безпеки в технології .NET

Практична цінність Розробка додатка з використанням удосконалених методів захисту в технології .NET.

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 – 29.01.2024
Аналіз літературних джерел	30.01.2024 – 14.02.2024
Дослідження історії виникнення і розвитку технології .NET	15.02.2023 – 27.02.2024
Розгляд основних компонентів та функцій технології .NET	28.02.2024 – 10.03.2024
Ознайомлення з методами використання технології .NET	11.03.2024 – 16.03.2024
Аналіз переваг і недоліків технології .NET	17.03.2024 – 19.03.2024
Дослідження методів захисту в технології .NET	20.03.2024 – 26.03.2024
Порівняння методів захисту технології .NET	27.03.2024 – 29.03.2024
Розробка додатка з використанням методів безпеки в технології .NET	30.03.2024 – 28.04.2024
Розробка рекомендацій щодо захисту вебдодатків за допомогою методів захисту в технології .NET	29.04.2024 – 2.05.2024
Оформлення пояснювальної записки згідно з методичними рекомендаціями	3.05.2024 – 12.05.2024
Подача пакету документів на розгляд ЕК	13.05.2024 – 18.05.2024

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект Збільшення ефективності забезпечення безпеки додатків створених за допомогою технології .NET.

Соціальний ефект	Удосконалені методи безпеки в технології .NET зменшують ризик несанкціонованого доступу, а також різноманітних атак

7. ДОДАТКОВІ ВИМОГИ

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

Завдання прийняв до виконання _____ Аліна КЛЕЩЕНКО
(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 дипломної роботи «Методи безпеки в технології .NET»: 87 сторінки, 25 рисунків, 2 додатки та 2 таблиці та 80 літературних джерел.

Об’єкт дослідження – процес дослідження методів безпеки в технології .NET.

Мета роботи – дослідження методів безпеки в технології .NET і розробка комплексної системи захисту для вебдодатка за допомогою технології .NET.

Методи дослідження – аналіз, порівняння, синтез, експеримент.

У роботі досліджено методи захисту в технології .NET. Розроблено практичну реалізацію комплексної системи захисту, використовуючи удосконалені методи захисту.

Наукова новизна: запропоновано розробка додатка з використанням удосконалених методів захисту в технології .NET.

Актуальність теми: технологія .NET використовується для розробки різноманітних програмних продуктів, які часто працюють з чутливою інформацією, включаючи особисті дані користувачів, фінансові та комерційні деталі. Отже, забезпечення високого рівня безпеки в цих додатках має вирішальне значення для запобігання несанкціонованому доступу та зловживанню. Традиційні методи безпеки можуть виявитися недостатніми перед сучасними загрозами та атаками. Використання сучасних методів безпеки в технології .NET дозволяє розробникам створювати надійні та ефективні захисні механізми для своїх додатків. Це може охоплювати використання різноманітних криптографічних протоколів, механізмів контролю доступу, а також вдосконалені методи автентифікації та авторизації, що використовуються для перевірки ідентичності користувачів та надання їм відповідних прав доступу.

Ключові слова: технологія .NET, автентифікація, авторизація, шифрування, хешування, XSS, CSRF.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ABAC	–	Attribute-Based Access Control
ACL	–	Access Control List
AD	–	Active Directory
BCL	–	Base Class Library
CLI	–	Common Language Infrastructure
CLR	–	Common Language Runtime
CSP	–	Content-Security-Policy
CSRF	–	Cross-Site Request Forgery
FCL	–	Framework Class Library
GC	–	Garbage Collector
IoT	–	Internet of Things
JIT	–	Just-In-Time
JVM	–	Java Virtual Machine
MAUI	–	Multi-platform App UI
ML	–	Machine Learning
MSIL	–	Microsoft Intermediate Language
OIDC	–	OpenID Connect
SAML	–	Security Assertion Markup Language
SDK	–	Software Development Kit
SPA	–	Single Page Applications
STS	–	Security Token Service
RBAC	–	Role-Based Access Control
UWP	–	Universal Windows Platform
WCF	–	Windows Communication Foundation
WPF	–	Windows Presentation Foundation
WWF	–	Windows Workflow Foundation
XSS	–	Cross-Site Scripting

ОС	–	Операційна система
ПК	–	Персональний комп’ютер
ШІ	–	Штучний інтелект

ЗМІСТ

ВСТУП

Актуальність цієї роботи полягає у її дослідженні методів забезпечення безпеки в контексті технології .NET. На сьогоднішній день вивчення .NET є надзвичайно важливим, оскільки ця платформа є однією з найбільш продуктивних для розробників. Технологія .NET постійно еволюціонує і розширює свої можливості. Наразі .NET дозволяє створювати програми для різних платформ, включаючи мобільні пристрої, персональні комп'ютери та хмарові сервіси. .NET є конкурентоспроможним рішенням для сучасної веброзробки, зокрема серверної частини. Велика кількість сучасних вебсайтів та застосунків розроблена з використанням платформи .NET. З погляду технічної реалізації, платформа .NET пропонує альтернативний метод упакування та встановлення програм. Замість вбудовування в операційну систему, .NET складається з пакетів NuGet і може бути компільована безпосередньо в додаток або включена у папку всередині програми.

Захист даних, запобігання несанкціонованому доступу та кібератак стають все більш важливими завданнями як для приватних осіб, так і для великих організацій. У світлі постійного росту кількості кіберзагроз і зловмисних атак, ретельне вивчення та використання методів безпеки в .NET стає надзвичайно важливим завданням для розробників та адміністраторів систем.

Метою кваліфікаційної роботи є дослідження методів безпеки в технології .NET і розробка комплексної системи захисту для вебдодатка за допомогою технології .NET.

Для досягнення поставленої мети необхідно виконати такі *завдання*:

- дослідити теоретичні відомості про технологію .NET;
- проаналізувати методи безпеки в технології .NET;
- розробити новий метод безпеки в технології .NET.

Науковою новизною цієї кваліфікаційної роботи є розробка додатка з використанням удосконалених методів захисту в технології .NET. Проведено аналіз їх ефективності, складності впровадження та кращих практик використання з

акцентом на актуальні та новітні методи. Ці методи включають автентифікацію та авторизацію, шифрування даних, контроль доступу, захист від SQL-ін’єкцій, XSS та CSRF атак.

Об’єктом дослідження є процес дослідження методів безпеки в технології .NET.

Предметом дослідження є методи безпеки в технології .NET.

Сформовані в результаті, теоретичні та практичні рекомендації можуть бути використані для забезпечення захисту вебдодатків за допомогою технології .NET.

РОЗДІЛ 1
ОГЛЯД ТЕХНОЛОГІЇ .NET

1.1 Визначення технології .NET

.NET – це платформа розробки програмного забезпечення, яка створена корпорацією Microsoft, і має відкритий вихідний код. Вона дозволяє створювати різноманітні додатки для різних операційних систем (ОС) та пристроїв, а також мобільних пристроїв, персональних комп’ютерів, а також хмарних сервісів. Технологія .NET містить в собі мови програмування, середовище виконання та класи бібліотек, які допомагають розробникам забезпечити функціональність своїх програм. Платформа має можливість запускати програми, написані ключовими мовами, найпопулярнішою з яких є C# [1, 2].

Середовище виконання мов Common Language Runtime (CLR) є центральною складовою технології .NET і покладається на системні служби ОС для керування виконанням коду, що написаний будь-якою із сучасних мов програмування. Завдяки базовому набору класів, розробники отримують доступ до сервісів платформи, які можна використовувати з будь-якою мовою програмування. CLR можна уявити як код, який завантажує програму, виконує її та надає забезпечує її всіма необхідними службами [1, 3].

Однією з головних складових в технології .NET є мова програмування C#, яка є однією з найпопулярніших мов програмування у світі. Окрім C#, платформа також підтримує інші мови програмування, такі як F#, Visual Basic, C++/CLI та PowerShell, а також мови інших організацій крім Microsoft (наприклад, Cobol, Java, PHP, Python, Scheme). Також є підтримка інших мов програмування (близько 25 мов). Існує також ряд мов, які більше не використовуються, наприклад IronRuby. На це можуть бути різні причини, наприклад, недостатній попит на конкретну мову, обмежені ресурси для підтримки, зміни стратегії компанії або спільноти розробників [1-3].

Системи, що розгортуються на платформі .NET, використовують специфічний підхід: замість безпосередньої компіляції в машинний код процесора, вихідні коди перетворюються в код проміжного рівня програмного забезпечення, відомий як Microsoft Intermediate Language (MSIL). Файли, що містять MSIL, можуть бути виконані на різних процесорах, якщо ОС має вбудований .NET CLR [3].

1.2 Історія виникнення та розвиток технології .NET

У 1995 році компанія Sun Microsystems створила Java – високорівневу об’єктно-орієнтовану мову програмування. Цікавістю цієї мови було те, що програми на Java перетворювалися в байт-код, який виконувався віртуальною машиною JVM (Java Virtual Machine), незалежно від архітектури комп’ютера. Оскільки її реалізація вважається досить простою, JVM швидко стала доступною для багатьох середовищ, забезпечуючи кросплатформність розроблених додатків. Однак однією з нерозв’язаних проблем розробників Java була проблема багатомовного програмування, коли програми, написані на різних мовах, не могли ефективно взаємодіяти. Ця взаємодія необхідна для розробки великих систем з розподіленим програмним забезпеченням, а також для створення компонентів програмного забезпечення, що можуть використовуватися в широкому спектрі ОС і мов програмування [4].

У 1999 році Microsoft розпочала розробку своєї альтернативи Java, а у 2002 році було оголошено про створення нової платформи програмування під назвою .NET Framework. Ця платформа дозволяє розробляти проекти, частини яких можуть бути написані на різних мовах програмування. Основним компонентом .NET є загальномовне середовище виконання CLR, яке може працювати з різними мовами програмування, такими як C#, F#, J#, Ada, C++ і т. ін. Основна відмінність від Java полягає в тому, що .NET Framework офіційно призначений для використання з ОС родини Microsoft Windows. Подібно до технології Java, середовище розробки .NET також генерує байт-код, призначений для виконання віртуальною машиною, відомою як .NET Runtime. Так само, як і байт-код для віртуальної машини Java, програми для

середовища .NET Runtime можуть виконуватися на будь-якій ОС, на якій встановлено .NET Runtime [4, 5].

З того часу, як платформа була випущена у версії 1.0, вона пройшла великий шлях розвитку. Навіть з появою нового версії платформи, а саме .NET Core, вона залишається дуже популярною. На сьогодні існує значна кількість бібліотек, програмних продуктів та фреймворків, які розроблені та активно розвиваються для .NET Framework [5].

.NET Framework, що існує з 2002 року, є реалізацією платформи .NET, що містить додаткові API, специфічні для ОС Windows, такі як API для розробки настільних додатків Windows через Windows Forms і WPF. Ця реалізація оптимізована для створення настільних додатків Windows [4].

Перші бета-версії .NET Framework вийшли наприкінці 2000-х років, а перша стабільна версія, .NET 1.0, була випущена 13 лютого 2002 року. Однією з його ключових особливостей було впровадження середовища CLR, що сприяло об’єктно-орієнтованій розробці вебдодатків [4, 6].

У 2002 році була випущена платформа .NET, яка включала три моделі додатків: ASP.NET Forms, Windows Forms і WPF, а також бібліотеку базових класів (BCL, Base Class Library), яка давала можливості для майбутніх проектів [6]:

Windows Presentation Foundation (WPF) – це графічна підсистема для відображення користувальницьких інтерфейсів, документів, зображень і медіафайлів у програмах Windows [3].

Windows Forms (WinForms) надає платформу для створення настільних програм за допомогою .NET Framework. Він містить різноманітні елементи керування GUI, такі як текстові поля, прапорці, перемикачі тощо, які можна використовувати для створення форм, діалогів та інших графічних інтерфейсів користувача [4].

ASP.NET – це фреймворк вебдодатків, який використовується для створення вебсайтів і вебдодатків за допомогою .NET Framework. Він надає набір елементів керування на стороні сервера, які можна використовувати для створення інтерактивних динамічних вебсторінок із підтримкою прив’язки даних і сценаріїв на стороні клієнта [1, 5].

З 2002 по 2007 рік були представлені наступні версії [6]:

– .NET 1.1 (квітень 2003): містив в собі численні вдосконалення, такі як покращена безпека для ASP.NET, підтримка Internet Protocol Version 6, ODBC;

– .NET 2.0 (листопад 2005): впровадив загальні колекції, нові функції для ASP.NET, ітератори, типи з можливістю nullable тощо;

– .NET 3.0 (листопад 2006): були введені WCF (Windows Communication Foundation), WPF (Windows Presentation Foundation), WWF (Windows Workflow Foundation).

Mono був заснований у 2001 році компанією Novell з метою забезпечити сумісність .NET з Linux та іншими платформами. Mono є однією з реалізацій платформи .NET, яка зазвичай використовується у випадках, коли необхідна оптимізація часу виконання. Це середовище підтримує такі додатки як Xamarin для Android, macOS, iOS, watchOS та tvOS, зосереджене переважно на зменшенні розміру. Крім того, Mono підтримує ігри, розроблені за допомогою платформи Unity, і відповідає всім поточним версіям стандарту .NET [4, 7].

Універсальна платформа Windows (UWP) є ще однією реалізацією платформи .NET, яка застосовується для створення додатків Windows з сенсорним управлінням та програмним забезпеченням для Інтернету речей (IoT). UWP призначена для об’єднання різних типів пристроїв, включаючи ПК, планшети, телефони та навіть Xbox. Ця платформа надає різноманітні сервіси, такі як централізований магазин додатків, середовище виконання (AppContainer) та набір API для Windows, що використовується замість Win32 (WinRT). Програми можна розробляти мовами програмування C++, C#, Visual Basic та JavaScript [4].

Перш ніж випустити версію .NET 3.5 у листопаді 2007, компанія Microsoft у жовтні оголосила про доступність вихідного коду для бібліотек майбутніх версій за допомогою ліцензії Microsoft Reference Software License. Репозиторій був вже доступним на початку 2018 року. Випуск .NET 3.5 включав такі важливі можливості, такі як підтримування AJAX, LINQ, а також динамічні дані й ASP.Net MVC [6].

Також 2007 року Silverlight додав .NET у браузер, надаючи розробникам можливість для створення багатофункціональних інтернет-додатків, подібних до Adobe Flash [8].

Корпорація Microsoft представила декілька версій до 2014 року [6]:

– .NET 4.0 (квітень 2010): були впроваджені такі нововведення, як Managed Extensibility Framework, бібліотека паралельних завдань, DLR та система перегляду Razor;

– .NET 4.5 (серпень 2012): ця версія представила підтримку Async, покращену підтримку ASP.NET, можливість стиснення Zip, а також вдосконалений CLR 4.0;

– .NET 4.5.1 (жовтень 2013): версія запропонувала кращу продуктивність і можливості налагодження, а також розширену підтримку розробки додатків та Windows Store.

Технологія .NET мала численні переваги, але одним із найважливіших недоліків була його обмежена сумісність лише з платформою Windows. У 2014 році, для виправлення цієї проблеми, корпорація Microsoft анонсувала радикальні зміни, представивши .NET Core – нову кросплатформну, дружню до хмари версію фреймворку з відкритим кодом. А у 2016 році, окрім традиційного .NET Framework, була випущена модульна платформа .NET Core, яка була сумісна з різними ОС, тобто, вона може працювати на різних платформах, що відкрило нові можливості для розробників. Ця кросплатформність .NET Core розширила область застосування і сприяло популяризації .NET серед представників бізнесу та, головне, розробників [1, 5, 6].

У той самий період часу, .NET систематично отримував нові версії та додаткові можливості, що сприяли його постійному розвитку та збереженню високих стандартів для розробників [6]:

– .NET 4.5.2 (серпень 2014);

– .NET 4.6 (липень 2015): введення нового компілятора JIT, підтримування TLS 1.1 і TLS 1.2, розширення з відкритим кодом, а також покращене відстежування подій;

– .NET 4.6.1 (листопад 2015): підвищена продуктивність, створення розподілених транзакцій в базі даних Azure SQL та покращене підтримування алгоритму цифрового підпису.

2016 року корпорація Microsoft придбала Xamarin. Xamarin.Forms – це те, за допомогою чого можна було створити мобільну програму на основі .NET. Xamarin.Forms давало розробникам можливість писати агностичний код платформи, який можна скомпілювати для роботи на iOS, Android та UWP. Підтримка Xamarin закінчилася 1 травня 2024 року [9].

У травні 2019 року компанія оголосила про великий випуск, який об’єднає екосистему: усі елементи .NET мали бути включені в платформу розробки .NET 5. Попри зміни в розкладі через COVID-19, уніфікована платформа розробки .NET 5 була нарешті представлена в листопаді 2020 року. Наступник .NET Framework 4.8 і .NET Core 3.1, .NET 5 наводить порядок у фрагментації .NET світ і надає багато функцій для створення програм на Windows, Android, Linux, macOS, iOS, watchOS, tvOS або за допомогою WebAssembly. Платформа має нові API, мовні функції та можливості виконання. Крім того, .NET 5 включає ASP.NET Core, Entity Framework Core, Xamarin, WinForms, WPF і ML.NET [1].

Реалізації .NET включають .NET Framework, .NET 5+ (i.NET Core) і Mono. Кожна реалізація .NET включає такі компоненти [4]:

– одне або кілька середовищ виконання, таких як CLR в .NET Framework та CLR в .NET 5+;

– у .NET Framework та .NET 5+ також можуть бути включені одне або кілька інфраструктур додатків, таких як ASP.NET, Windows Forms і WPF;

– бібліотека класів, такі як базова бібліотека класів .NET Framework та базова бібліотека класів .NET 5+;

– засоби розробки, які можуть використовуватися спільно між декількома реалізаціями.

Microsoft підтримує чотири реалізації .NET: .NET 5 (.NET Core) та пізніші версії, .NET Framework, Mono, UWP [4]. .NET 5+ (раніше відомий як .NET Core) є кросплатформною реалізацією платформи .NET, призначеною для роботи з

серверними і хмарними навантаженнями в масштабі. Вона також підтримує різноманітні інші типи робочих навантажень, включаючи настільні додатки, і працює на ОС Windows, macOS і Linux. .NET 5+ реалізує стандарт .NET Standard, що означає, що код, розроблений для .NET Standard, може бути використаний на платформі .NET 5+. ASP.NET Core, Windows Forms і WPF також підтримуються на .NET 5+ [4].

На стан 1 травня 2024 року .NET 8 – це остання випущена версія реалізації .NET. Вона перебуває на стадії попереднього перегляду та пропонує найновіші функції, але наразі її не рекомендується використовувати у великих проектах. Але вже анонсована наступна версія, а саме .NET 9 [10].

Розробка мобільних додатків – це ще одна сфера, де .NET за останні роки набула більшої популярності. 1 травня 2024 року проект Xamarin перейшов на .NET MAUI. Отже, .NET MAUI – це один з останніх релізів Microsoft на базі Xamarin.Forms. Це не просто перепакована версія, а кросплатформний інтерфейс користувача додатків. Він використовує новітні підходи для спрощення процесу написання кросплатформних мобільних додатків з використанням C# та XAML [11].

1.3 Основні компоненти та функцій технології .NET

Раніше існувало чотири основні варіації платформи .NET: .NET Framework, .NET Core, Xamarin і UWP. Ці реалізації спільно формували екосистему для розробки на .NET, кожна з яких містила свій набір бібліотек і фреймворків, щоб створювати різноманітні додатки. Але на момент травня 2024 року Xamarin більше не підтримується (Xamarin.Android, Xamarin.iOS, Xamarin.Mac були інтегровані безпосередньо в .NET, а саме в NET MAUI), .NET Core після .NET 5 є основною реалізацією .NET, розробка .NET Framework припинена, а UWP використовує власний набір API [1, 4, 11 – 13].

Хоча розробка .NET Framework припинилася, але середовище все ще підтримується і може використовуватися для нових проектів. Однак рекомендується використовувати .NET (об’єднання .NET Core та .NET Framework) [10].

.NET Framework – це середовище розробки програмного забезпечення, яке надає простір для виконання та набору бібліотек, а також інструментів для створення і запуску додатків в ОС Windows. Фреймворк включає різноманітні мови програмування, такі як C#, F# та Visual Basic, і підтримує ряд типів додатків, включаючи настільні, веб, мобільні та ігрові додатки [14].

.NET Framework включає два основні компоненти (рис. 1.1): середовище виконання CLR і бібліотеку класів .NET Framework. CLR відповідає за керування виконанням коду, написаного будь-якою з підтримуваних мов, тоді як бібліотека класів надає великий набір попередньо створених функцій і класів для створення різних програм [14, 15].

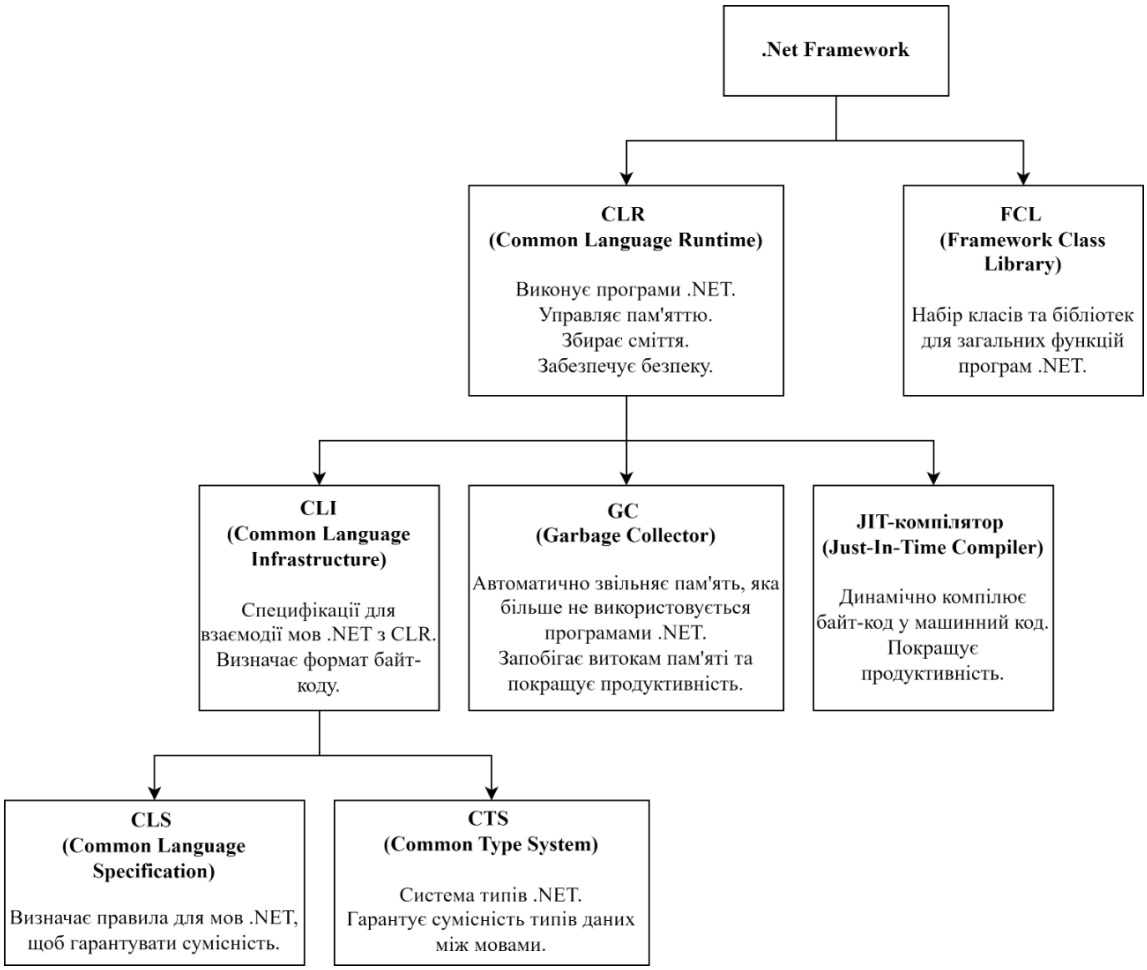

Рисунок 1.1 – Складові .NET Framework

.NET Framework побудований на спільному середовищі виконання для різних мов програмування, яка відповідає за управління виконанням коду та сприяє

полегшенню процесу розробки. Це виконавче середовище контролює код під час його роботи, забезпечуючи основні функції, такі як управління пам’яттю та потоками. Керування кодом є ключовим принципом цього середовища. Код, який виконується у цьому середовищі, відомий як керований код, у той час, як код, що не призначений для виконання під .NET, називається некерованим кодом [4, 15].

Платформа .NET Framework є повністю об’єктно-орієнтованою та дозволяє не лише використовувати наявні типи даних, але й створювати нові. У так званому світі технології .NET термін "тип" охоплює структури, класи, перерахування й інші форми даних. Крім того, платформа .NET також дає можливість розробляти компоненти, відомі як збірки, що забезпечують доступ до визначених у них типів іншим компонентам, навіть якщо вони розроблені за допомогою інших мов програмування [16].

Для доступності функціонала .NET Framework на інших ОС можна використати Mono. Спонсорована компанією Microsoft, Mono є реалізацією, яка відкритий вихідний код Microsoft .NET Framework, що є частиною .NET Foundation і базується на стандартах ECMA для C# та CLR [17].

Mono працює і активно підтримується на різних платформах, таких як Android, iOS і WebAssembly, забезпечуючи сумісність з .NET Framework та роблячи його функціонал доступним на цих системах. Mono все ще активно використовується і підтримується, але його фокус змістився в бік забезпечення сумісності з .NET Framework, а не з найновішими версіями .NET [18, 19].

Mono використовує власну реалізацію CLR, що забезпечує сумісність з .NET Framework, а також дозволяє виконувати код, який написаний на C# або інших мовах .NET, подібно до того, як це робить .NET Framework на Windows. Mono також містить в собі власні бібліотеки, які сумісні з бібліотеками .NET Framework [19].

Раніше .NET Core було версією платформи .NET, яка пропонує кросплатформний та модульний підхід з відкритим вихідним кодом. .NET, який раніше називався .NET Core, наразі є основною реалізацією [20].

Модульний підхід .NET означає можливість використовувати лише ті компоненти, які потрібні для створення програми, що робить її легшою та швидшою.

Крім того, .NET містить багато покращень продуктивності порівняно з .NET Framework, зокрема швидший час запуску та менше використання пам’яті [21].

Середовище виконання (runtime), бібліотеки та мови – це стовпи стека .NET. Компоненти вищого рівня, такі як інструменти .NET та стеки додатків, такі як ASP.NET Core, будуються на цих стовпах. Ці стовпи мають симбіотичний зв’язок, оскільки були розроблені та створені разом однією групою. Компоненти .NET показано в табл. 1.1 [13, 21, 22].

Таблиця 1.1 – Компоненти .NET

Компонент	Опис
Runtime (середовище виконання)	Це ядро .NET, яке відповідає за запуск та виконання коду вашої програми. Воно керує пам’яттю, безпекою та іншими важливими аспектами роботи програми.
Бібліотеки	Бібліотеки .NET надають готові блоки коду для виконання різних завдань. Вони можуть включати функціональність для роботи з файлами, мережею, базами даних, JSON, XML та багатьма іншими речами.
Компілятор	Компілятор перетворює ваш вихідний код (наприклад, написаний на C#) в машинний код, який може виконуватись виконавчим середовищем. Він аналізує код, перевіряє його на помилки та генерує інструкції, які розуміє виконавче середовище.
SDK (Software Development Kit) та інші інструменти	.NET SDK пропонує набір інструментів командного рядка та бібліотек для побудови, тестування, пакування та розгортання ваших програм. Він також може включати інструменти для моніторингу продуктивності програми після її запуску.
Стеки програм (App Stacks)	Стеки програм .NET, такі як ASP.NET Core для веброзробки або Windows Forms для десктопних додатків, надають каркас для створення певних типів програм. Вони пропонують готові шаблони, компоненти та функціональність, які спрощують процес розробки.

По суті, платформа .NET має інший спосіб упакування та встановлення. Замість того, щоб бути вбудованою в ОС, .NET складається з пакетів NuGet та може бути компільованою безпосередньо в додаток або розміщеною у папці всередині програми. Це свідчить про те, що додатки можуть включати .NET і, таким чином, повністю співіснувати на комп'ютері. NuGet є інструментом керування пакетами для .NET, який має в собі значну кількість пакетів (сотні тисяч пакетів), які реалізують різноманітні функціональні можливості для багатьох сценаріїв. Більшість програм розраховують на пакети NuGet для певних функцій [13, 18].

Середовище виконання .NET забезпечує автоматичне керування пам'яттю за допомогою збирача сміття (Garbage Collector, GC). Модель управління пам'яттю являє собою одну з ключових характеристик для будь-якої мови програмування, і в контексті мов .NET це особливо важливо. В системі .NET використовується автономний контроль GC. Він спроєктований для забезпечення оптимальної роботи без активного втручання, а також має можливості налаштування для різних вимог інтенсивності завантаження. Поточна реалізація GC є результатом багаторічних досліджень і оптимізацій, проведених в рамках різноманітних робочих навантажень [13].

Основні бібліотеки містять тисячі типів, багато з яких інтегровані з мовою C# та відповідають їй. Наприклад, конструкція foreach мови C# дозволяє нумерувати довільні колекції. Використання шаблонів дозволяє List<T> ефективно та просто опрацьовувати колекції. Розробники можуть дозволити керувати ресурсами для GC, але також можна здійснювати оперативне очищення за допомогою прямої мовної підтримки IDisposable, яка є у конструкції using [14].

SDK .NET є основою для розробки програмного забезпечення на платформі .NET. Керування версіями для .NET SDK працює дещо інакше, ніж для Runtime.NET. Для узгодження з новими випусками Visual Studio оновлення .NET SDK іноді включають нові функції або нові версії компонентів, як-от MSBuild і NuGet. Ці нові функції чи компоненти можуть бути несумісними з версіями, які постачалися в попередніх оновленнях SDK для тієї самої основної чи проміжної версії [23].

В технології .NET є стеки програм. Наприклад, Windows Forms – це фреймворк для створення настільних програм Windows. Він пропонує інтерфейс перетягування для створення інтерфейсів користувача та підтримує прив'язку даних, що дозволяє розробникам легко підключати елементи інтерфейсу до джерел даних. Windows Forms – це зріла технологія, яка існує з перших днів .NET і широко використовується для створення бізнес-додатків. А ще однією програмою є ASP.NET, яку можна застосувати для розробки динамічних вебсайтів і вебдодатків. В його основі лежить CLR, який дозволяє розробникам писати код ASP.NET за допомогою різних видів мов програмування .NET [1, 11].

У технології .NET є загальномовне середовище виконання (CLR), яке відповідає за виконання коду та надає служби, які полегшують процес розробки. Інструменти та компілятори використовують функції CLR для створення коду, який може ефективно працювати в керованому середовищі виконання. Цей код, відомий як керований код, володіє рядом переваг, включаючи міжмовну інтеграцію, обробку винятків між різними мовами, підвищену безпеку, підтримку керування версіями та розгортання, простішу модель взаємодії між компонентами, а також зручні служби для налагодження та профілювання [24].

Для того, щоб середовище виконання могло надавати послуги керованому коду, яке забезпечує обслуговування керованого коду, компілятори мов повинні створювати метадані, які визначають типи, члени та посилання у коді розробника. Ці метадані зберігаються разом з кодом, при цьому кожен виконуваний файл середовища має портативний формат, який включає метадані. Середовище виконання використовує ці дані для здійснення пошуку та завантаження класів, виділення пам'яті для екземплярів об'єктів, викликів методів, генерації машинного коду, забезпечення безпеки та встановлення обмежень часу виконання [4, 24].

Середовище виконання автоматично обробляє макет об'єктів і керує посиланнями на них, звільняючи пам'ять, коли вони вже не потрібні. Такі об'єкти, що керуються цим механізмом, називаються керованими даними. Процес GC допомагає уникати витоків пам'яті (memory leak) та деяких інших помилок програмування. Якщо код є керованим, то в програмі можна використовувати як керовані, так і

некеровані дані, або навіть обидва типи одночасно. Оскільки мовні компілятори надають власні типи, наприклад примітивні, часом важко визначити, чи дані тобто контрольовані середовищем виконання [24].

CLR полегшує розроблення компонентів та програм, що дозволяє об’єктам взаємодіяти між собою незалежно від мови програмування. Об’єкти, написані на різних мовах, можуть злагоджено взаємодіяти та інтегруватися між собою. Наприклад, можна описати клас одною мовою, а після цього використати його в іншій або викликати метод у вихідному класі, написаному іншою мовою. Також можна передавати екземпляри класів написаному іншою мовою методу класів. Ця можливість міжмовної інтеграції стає можливою завдяки тому, що мовні компілятори та інструменти, орієнтовані на середовище виконання, користуються загальною системою типів, визначеною самим середовищем. Вони дотримуються стандартів середовища, щоб визначити нові типи та для їх створення, збереження, використання, а також прив’язки до типів [4, 24].

У метаданих, що супроводжують керовані компоненти, міститься інформація про компоненти та ресурси, які були використані для їх створення. Середовище виконання використовує цю інформацію для перевірки компонента або програми, а саме те, що вони мають необхідні версії всіх потрібних складових, що зменшує ймовірність помилки коду через невідповідності в залежностях. Інформація про реєстрацію та стан більше не зберігається в реєстрі, де їх було важко знайти та обслуговувати. Натомість інформація про типи, які визначаються, та їхні залежності зберігаються разом із кодом як метадані. Це спрощує завдання розгортання та видалення компонентів [24].

Компілятори та інструменти мов розкривають можливості середовища виконання таким чином, щоб сприяти їх зручності та зрозумілості для розробників. Особливості самого середовища виконання можуть бути більш помітними в одному контексті, ніж в іншому [4, 24].

Тепер детальніше розглянемо процес компіляції програмного коду. Спочатку розробник створює вихідний код додатка, використовуючи мову програмування, що підтримується в технології .NET, наприклад, C#, C++/CLI, Visual Basic .NET та інші.

Цей процес зазвичай відбувається у певному середовищі розробки, наприклад, у Microsoft Visual Studio. Після написання вихідного коду, компілятор створює збірку – файл, який містить в собі CIL-інструкції, метадані та маніфест. Маніфест є описом збірки, складеним за допомогою метаданих [16].

Після введення в дію додатку на комп'ютері, механізм виконання .NET починає свою роботу. Якщо вихідний код використовує BCL, такі як ті, що містяться у збірці mscorlib.dll, вони автоматично завантажуються за допомогою спеціального завантажувача класів. JIT-компілятор адаптує збірку з урахуванням апаратного та програмного забезпечення ПК, на якому запускається додаток. Після цього відбувається виконання програмного коду (рис. 1.2) [16].

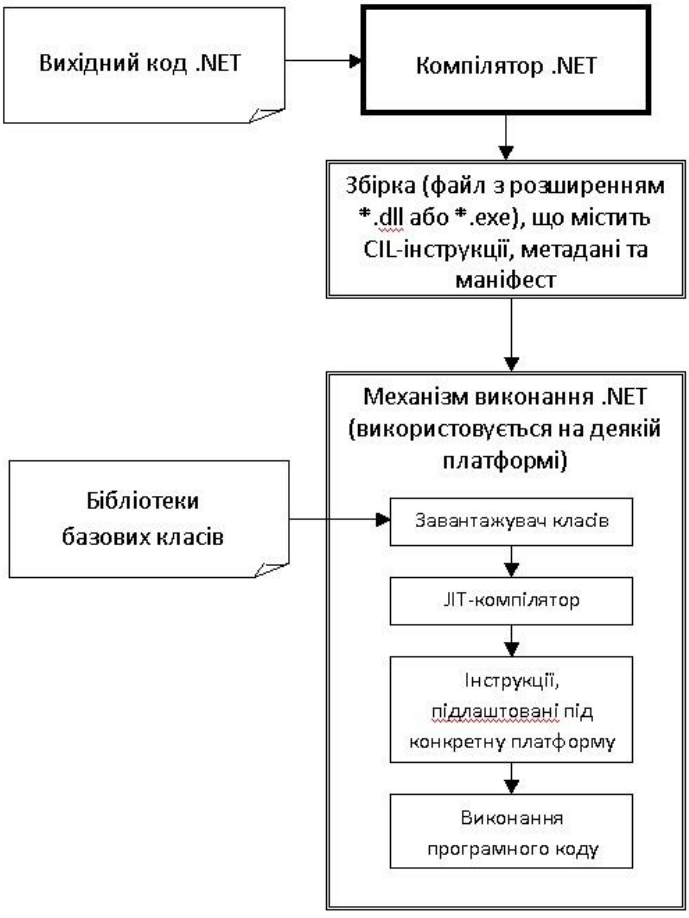

Рисунок 1.2 – Взаємодія між вихідним кодом .NET, компілятором .NET і механізмом виконання .NET

1.4 Використання технології .NET

.NET славиться своєю універсальністю, здатністю опановувати практично будь-яке завдання розробки, з яким можна стикнутися. Технологія .NET застосовується для реалізації API мікросервісу через вебплатформу ASP.NET. Завдяки вбудованій підтримці та можливостям відкритого вихідного коду, є змога збільшити темп процесу розробки і створювати високоякісні проекти, використовуючи потужні інструменти, доступні під рукою [25].

Технологія .NET ідеально підходить для розробки динамічних вебсайтів та вебдодатків різної складності, які працюють на ОС Windows, Linux і macOS. Також розробники можуть створювати вебсайти з великим об'ємом даних, щоб максимально використовувати їх потенційну швидкість. Для створення вебдодатка розробнику необхідно володіти мовою програмування C# і мати знання про фреймворк ASP.NET MVC. Крім того, програмісту потрібно розуміти поняття клієнт/сервер, протокол HTTP, REST, мову JavaScript, а також знати різницю між Frontend і Backend. А також розробнику важливо мати розуміння про домени, хостинги, плани, а також про хмарні технології (наприклад, MS Azure, Amazon, Google Cloud) [5, 25].

Багато корпоративних застосунків і порталів створюються за допомогою ASP.NET. Цей фреймворк забезпечує розширеним набором інструментів, який підходить для того, щоб створювати складні системи з обробкою даних, управлінням доступом та інтеграцією з іншими корпоративними системами. ASP.NET є одним із найпопулярніших фреймворків для веброзробки з .NET. Він підтримує створення вебдодатків, включаючи підтримку вебформ, архітектури MVC (Model-View-Controller) і Web API. Використовуючи ASP.NET, розробники можуть створювати вебпрограми, які є безпечними, масштабованими та простими в обслуговуванні. ASP.NET також надає багатий набір бібліотек для типових завдань веброзробки, таких як доступ до даних, автентифікація та кешування [11, 26].

Ще одним популярним фреймворком для веброзробки з .NET є Blazor, який є новим фреймворком інтерфейсу користувача для створення вебдодатків за допомогою .NET і C#. Blazor дозволяє розробникам розпочати розробку власних

вебдодатків з архітектурою на основі компонентів, подібною до популярних фреймворків інтерфейсу, таких як React та Vue.js. За допомогою Blazor розробники можуть писати як зовнішню, так і внутрішню логіку додатку, використовуючи C# для створення та підтримки вебдодатків [11].

.NET також надає широкий спектр інструментів та бібліотек, щоб працювати з вебтехнологіями, такими як HTML, JavaScript та CSS. Наприклад, розробники можуть використовувати служби JavaScript .NET для запуску програм Node.js у програмі .NET або використовувати популярну бібліотеку jQuery для сценаріїв на стороні клієнта [11].

За допомогою технології .NET, використовуючи .NET MAUI (Multi-platform App UI), можна розробляти не тільки веб, а й клієнтські додатки – продукти, які запускаються на ПК і мобільних пристроях кінцевих користувачів. Ці додатки працюють як на iOS й Android, так і на Windows, використовуючи єдину кодову базу і забезпечуючи їхню сумісність і привабливий вигляд на всіх пристроях [5, 25].

У середовищі .NET можна створювати нативні додатки для ОС Windows і macOS, використовуючи вебтехнології, які працюють у будь-якому місці, такі як WPF і Windows Forms [25, 27].

WPF – це ще один фреймворк, розроблений для того, щоб допомогти розробникам створювати сучасні, візуально привабливі настільні додатки. Він надає потужний набір елементів керування інтерфейсом і підтримує розширені можливості, такі як 3D-графіка та анімація. WPF також використовує XAML, мову розмітки, яка дозволяє розробникам відокремити інтерфейс від логіки програми, що полегшує її модифікацію та підтримку [11].

.NET пропонує багато різних технології та інструментів для розробки ігор, які працюють у двовимірному (2D) чи тривимірному (3D) просторі. Ця платформа дозволяє розробникам швидко та ефективно інтегрувати в ігри необхідний функціонал, використовуючи різноманітні можливості .NET. Необхідно відзначити, що ігровий рушій Unity, що має широку популярністю серед розробників, ґрунтується на реалізації .NET, а саме Mono [5, 25].

Багато відомих ігор, таких як Inside, Kerbal Space Program, Endless Legend та Pokemon Go, були успішно створені з використанням платформи Unity. Необхідно зазначити, що для розробки ігор під платформу Unity необов’язково володіти глибокими знаннями мови програмування C++, оскільки для цього використовується мова C#, яка є доступнішою для більшості розробників. Ще однією з переваг використання .NET з Unity є можливість використовувати бібліотеки та інструменти .NET для розробки ігор. Наприклад, розробники можуть використовувати фреймворк MonoGame. Бувши кросплатформною платформою з відкритим кодом, MonoGame надає інструменти та бібліотеки для створення 2D і 3D ігор за допомогою .NET [5, 11].

За допомогою технології .NET, використовуючи ML.NET, є можливість розробки високопродуктивних моделей машинного навчання з інтуїтивно зрозумілими структурами кодування. Це ідеальний вибір для розробників, які прагнуть інтегрувати можливості штучного інтелекту (ШІ) у свої продукти [25, 28].

ML.NET ("Machine Learning") – популярна платформа з відкритим кодом , яка призначена для розробки програм ШІ. До неї входить низка бібліотек і інструментів для побудови та навчання моделей машинного навчання, а також інструменти для оцінки та розгортання цих моделей. Завдяки ML.NET розробники можуть створювати програми ШІ за допомогою мов програмування C# або F#, не вимагаючи глибоких знань про алгоритми або бібліотеки машинного навчання [11, 28].

Окрім ML.NET, .NET надає низку інших інструментів і бібліотек для машинного навчання та ШІ. Наприклад, Microsoft Cognitive Services пропонує низку готових моделей машинного навчання та API для обробки природної мови, комп’ютерного зору та інших завдань ШІ. Ці служби можна інтегрувати в програми .NET за допомогою простих REST API, що полегшує додавання можливостей AI до будь-якої програми. Іншим популярним інструментом для машинного навчання та ШІ з .NET є бібліотека TensorFlow.NET. TensorFlow.NET надає інтерфейс .NET для популярної системи машинного навчання TensorFlow, що дозволяє розробникам створювати та навчати моделі ШІ за допомогою бібліотеки TensorFlow у програмі .NET [11].

Також, використовуючи технологію .NET, розробник може розгортати власні хмарні сервіси або використовувати наявні, що дає змогу розміщувати додатки на одній з основних хмарних платформ. Microsoft Azure — це одна з популярних платформ хмарних обчислень, яка надає ряд послуг для розміщення та масштабування програм .NET. За допомогою Azure розробники можуть розміщувати додатки .NET у хмарі, використовувати хмарне сховище та бази даних, а також використовувати ряд інших хмарних служб, таких як машинне навчання та аналітика [11, 25].

Крім Azure, .NET надає ряд інших інструментів і бібліотек для створення хмарних програм. Наприклад, Azure SDK для .NET надає низку API та інструментів для роботи зі службами Azure, такими як зберігання, обмін повідомленнями та автентифікація. Іншим популярним інструментом для хмарної розробки з .NET є Docker, платформа для створення, доставки та запуску додатків у контейнерах. За допомогою Docker розробники можуть створювати та розгортати програми .NET масштабованим і ефективним способом, не турбуючись про базову інфраструктуру [11].

Використання бібліотек та фреймворків з відкритим кодом в технології .NET дозволяє створювати додатки IoT, які можуть працювати на різних пристроях, таких як Raspberry Pi, HummingBoard, BeagleBoard, Pine A64 та інші. Для будь-яких рішень, недоступних у рамках платформи, є можливість знайти багато спеціальних бібліотек функцій у загальнодоступному репозиторії NuGet [5, 25, 28].

Крім того, .NET nanoFramework є платформою з відкритим доступом до вихідного коду, яка призначена для розробки додатків на пристроях з обмеженими ресурсами, наприклад мікроконтролерах. Це повна та сучасна реалізація .NET, яка включає підмножину бібліотек класів .NET і середовища виконання та оптимізована для невеликих пристроїв з обмеженим обсягом пам’яті. Розробники можуть використовувати мову C# і Visual Studio або іншу можливу мову для написання програм для таких пристроїв, як датчики IoT і невеликі вбудовані системи. .NET nanoFramework підтримує широкий спектр пристроїв і архітектур, включаючи ARM

Cortex-M, ESP32 і STM32. Він також містить низку драйверів і бібліотек, які спрощують процес розробки [11].

1.5 Переваги та недоліки використання технології .NET

Переваги використання технології .NET [11, 29]:

– кросплатформність: .NET сприяє розробці програмних продуктів для різних ОС, включаючи Windows, Linux і macOS. Крім того, як платформа з відкритим кодом, вона зосереджена на більшій гнучкості та охопленні;

– спрощення процесу розробки: платформа надає зручні API, потужні бібліотеки та інтегровані інструменти для розробки, прикладом є Visual Studio, а також дозволяє розробникам писати код різними мовами. Це все спрощує та прискорюють процес створення програм;

– підтримка різноманітних мов програмування: .NET підтримує ряд мов програмування, серед яких C#, F#, і VB.NET, що надає розробникам можливість вибрати мову відповідно до вимог і потреб конкретного проекту;

– якісний інтерфейс та архітектура: ця платформа пропонує якісний користувацький інтерфейс та архітектуру. Під час розробки програм розробник може скористатися перевагами динамічного та ефективного інтегрованого середовища розробки (IDE);

– високий рівень безпеки та надійності: .NET забезпечує ефективний контроль за безпекою та надійністю програм, включаючи механізми керування пам’яттю та обробки винятків. Розробники можуть використовувати численні корисні класи та сервіси, що пропонуються платформою .NET та її CLR. Ці ресурси полегшують розробку безпечного коду, впровадження рольової безпеки та використання криптографії. Це допомагає захиститися від поширених загроз безпеки, наприклад, SQL-ін’єкції та міжсайтовий скриптинг;

– мовна сумісність: .NET дозволяє розробникам писати код різними мовами, а також забезпечує можливість інтеграції з наявним кодом, написаним іншими мовами.

Це полегшує повторне використання наявного коду та створення нових програмних рішень;

– надійні інструменти та фреймворки: .NET надає розробникам широкий спектр інструментів та фреймворків для створення та розгортання додатків. Сюди входять середовища розробки, такі як Visual Studio з автоматично встановленими інструментами, які спрощують процес створення вебдодатків і взаємодії з базами даних;

– ефективна робота з даними: .NET надає потужні інструменти для взаємодії з базами даних, включаючи засоби об’єктно-реляційного відображення, такі як Entity Framework;

– гнучкість та розширюваність: платформа дозволяє легко інтегрувати сторонні бібліотеки та сервіси, що збільшує гнучкість і масштабованість розроблюваних застосунків;

– підтримка хмарних сервісів: .NET є тісно інтегрованою з хмарними платформами, зокрема з Azure, що сприяє простоті створення, розгортання та управління хмарними додатками;

– велика спільнота розробників: .NET має активну та велику спільноту розробників, яка надає безліч ресурсів та підтримку для розробників. Сюди входять бібліотеки з відкритим вихідним кодом, форуми, групи користувачів, зустрічі та заходи. Кількість .NET розробників сягає 8 мільйонів.

Недоліки використання технології .NET [30]:

– вартість ліцензування: для більшості служб у стеку .NET ліцензії не надаються безплатно. Наприклад, використання компонентів Visual Studio IDE, які поліпшують якість коду та надають додаткові послуги забезпечення якості, може підвищити вартість розробки, використовуючи .NET, для великих компаній. Іноді фінальні витрати на розробку проекту можуть перевищити початкові оцінки, оскільки може виникнути потреба в додаткових інструментах та сервісах під час процесу розробки;

– скомпільоване програмування: NET використовує скомпільовані мови програмування, що іноді може призвести до менш лаконічного коду, особливо якщо

порівнювати з більш ефективними технологіями, такими як Python. Хоча кожне оновлення .NET прагне вирішити цю ситуацію, деякі аспекти програмування за допомогою .NET все ще можуть становити більше труднощі, ніж кодування за допомогою мов програмування із скриптами;

– обмеження розробки інтерфейсу: .NET може бути не оптимальним вибором для розробки інтерфейсу. Хоча він пропонує технології, такі як ASP.NET, MVC Core та Blazor для створення інтерфейсу, ці технології можуть відстати за гнучкістю та функціональністю від таких фреймворків на основі JavaScript, як Angular, React.js та Vue.js;

– витік пам’яті (memory leak): ця проблема поширена практично в будь-якій технології, оскільки іноді фреймворк не вивільняє пам’ять, яка йому вже не потрібна. .NET має певну репутацію, коли йдеться про проблеми, пов’язані з пам’яттю та витоками пам’яті. Розробники, які обирають працювати з цією технологією, повинні вкладати додаткові зусилля та час у належне управління ресурсами. Розробники та інженери повинні бути уважними, оскільки проекти стають більшими, а витік пам’яті збільшується разом з розміром додатка.

Висновок за розділом 1

В першому розділі дипломної роботи було досліджено основні характеристики технології .NET, а саме: визначення, історія виникнення та розвитку, основні компоненти та функції, а також використання в створенні різноманітних додатках. Після огляду характеристик були виділені переваги та недоліки використання технології .NET.

Визначено, що технологія .NET є однією з найважливіших та широко використовуваних платформ для розробки програмного забезпечення. Її історія виникнення та розвитку підтверджує постійну еволюцію та адаптацію до вимог сучасного програмування. Аналіз основних компонентів та функцій технології .NET дозволив зрозуміти, як різноманітні елементи цієї платформи взаємодіють між собою та сприяють розробці різноманітних програмних продуктів.

З огляду на досліджену інформацію, особливу увагу було приділено визначенню переваг технології .NET, такі як широка спільнота розробників, високий рівень безпеки та надійності, спрощення процесу розробки, підтримка різноманітних мов програмування, а також багато інших переваг. Однак недоліки в технології .NET також є, такі як скомпільоване програмування, обмежена розробка інтерфейсу та витік пам’яті.

Отже, розділ показує важливість технології .NET у сучасному програмуванні, її постійний розвиток та застосування у різних сферах. Ці дані будуть корисними для подальших розділів роботи та вивчення її застосування в конкретних випадках.

РОЗДІЛ 2
МЕТОДИ ЗАХИСТУ В ТЕХНОЛОГІЇ .NET

2.1 Автентифікація та авторизація

Автентифікація – це процес перевірки особистості користувача за допомогою надання та підтвердження його ідентифікаційних даних. Цей процес має на меті переконатися, що користувач є тим, за кого себе видає, і має право на доступ до системи або ресурсів [31].

Авторизація – це процес надання користувачеві доступу до певних ресурсів або виконання певних дій у системі. Цей процес ґрунтується на результатах автентифікації, і визначає, що саме може робити користувач після успішного входу в систему [32].

2.1.1 Windows Authentication

Windows Authentication – це метод автентифікації, який використовується для перевірки ідентифікованих користувачів Windows у вебдодатках ASP.NET. Він ґрунтується на інтеграції з Active Directory (AD), службою каталогів, яка використовується в ОС Windows для керування користувачами та групами. Автентифікація на основі Windows здійснюється між сервером Windows і клієнтською машиною, зокрема використовуючи ідентифікатори домену AD або облікові записи Windows для ідентифікації [33, 34].

Автентифікація Windows зазвичай використовується в корпоративних мережах, де користувачі, клієнтські програми та вебсервери належать до одного домену Windows. Вона є сценарієм зі збереженням стану, який передбачено для внутрішніх мереж, де проксі-сервери або балансувальники навантаження зазвичай не обробляють трафік між клієнтами та серверами. У випадку використання проксі-сервера або балансувальника навантаження, автентифікація Windows виконується лише в разі,

якщо вони: здійснюють автентифікацію або передають інформацію про автентифікацію користувача в програму (наприклад, через заголовок запиту), яка діє на основі цієї інформації [33].

Альтернативою автентифікації Windows у середовищах з використанням проксі та балансувальників навантаження є об’єднані служби Active Directory (ADFS) з протоколом OpenID Connect (OIDC). Також альтернативою є автентифікація за формою, яка може бути використана у випадку, якщо необхідно реалізувати власний процес автентифікації через внутрішню базу даних і спеціальну сторінку. Проте, для вебпрограм, призначених для обмеженої кількості користувачів, які вже є частиною мережевого домену, автентифікація Windows є кращим і вигіднішим вибором [33, 34].

2.1.2 Forms Authentication

Forms Authentication – це метод, який використовується у вебдодатках для керування автентифікацією та авторизацією користувачів. Він передбачає збір облікових даних користувача (зазвичай імені користувача та пароля) через форму HTML і перевірку їх у базі даних користувачів або каталозі. Після автентифікації користувачеві надається доступ до обмежених ресурсів або функцій програми [35].

Forms Authentication є альтернативою Windows Authentication і використовується для автентифікації користувачів у вебзастосунках. На відміну від Windows Authentication, Forms Authentication не залежить від ОС чи домену. За допомогою автентифікації за формами, дані про користувача зберігаються у зовнішньому ресурсі, такому як база даних членства або файл конфігурації програми. Після проходження процедури автентифікації, система зберігає токен автентифікації у файлі cookie або у URL-адресі, щоб у подальших запитах автентифікованому користувачеві не потрібно було повторно вводити облікові дані [36].

Під час запиту користувача до сторінки ASP.NET через браузер, ASP.NET проводить перевірку наявності маркера автентифікації форми. У випадку позитивного результату середовище виконання направляє користувача на сторінку входу та запускає процес перевірки ідентифікатора користувача та пароля. Після

успішної автентифікації користувача середовище виконання автоматично конфігурується з використанням файлу cookie автентифікації, який містить фактичний квиток [37].

2.1.3 Claims-based Authentication

Claims-based Authentication (автентифікація на основі тверджень) – це метод автентифікації, який використовує твердження для представлення ідентифікованих користувачів та їхніх дозволів. Claims (твердження) – це невеликі фрагменти даних, які містять інформацію про користувача, наприклад, ім’я користувача, роль та інші атрибути. Ці дані передаються як потік байтів під час передачі через мережу, для перевірки на стороні одержувача claims мають цифровий підпис [38].

Claims-based Authentication є стандартним протоколом безпеки, призначеним для перевірки ідентичності користувача на хост-системі. Автентифікація на основі твердження складається з набору стандартів WS-*, які описують використання токенів безпеки мови розмітки тверджень (Security Assertion Markup Language, SAML) у режимі пасивної або активної інтеграції. У пасивному режимі (коли WS-Federation використовується з вебдодатками Dynamics 365 для взаємодії з клієнтами) або у активному режимі (коли WS-Trust використовується з клієнтами Windows Communication Foundation (WCF)). Ця система автентифікації взаємодіє з WCF, забезпечуючи безпечну перевірку ідентичності користувачів та безпечний канал зв’язку з сервером Dynamics 365. Усі версії Dynamics 365 Customer Engagement (взаємодія з клієнтами) підтримують автентифікацію на основі тверджень [39].

Автентифікація на основі тверджень передбачає наявність служби маркерів безпеки (Security Token Service, STS), яка діє на сервері. Сервер STS може базуватися на Active Directory Federation Services (AD FS) V2 або на будь-якій платформі, що підтримує офіційний протокол STS [38, 39].

Процес автентифікації на основі тверджень полягає у тому, що запит на перевірку ідентичності користувача надсилається з Dynamics 365 для взаємодії з клієнтами або користувацького застосунку на сервер STS. Сервер STS визначає, чи

потрібно автентифікувати користувача, і, в разі позитивної відповіді, виділяє підписаний і зашифрований токен SAML, що містить інформацію про автентифікацію користувача. Токен має обмежений строк дії, і його може знадобитися періодично оновлювати, залежно від тривалості використання програмою [39].

2.1.4 OAuth 2.0

OAuth 2.0 – це протокол авторизації, який дозволяє програмам сторонніх розробників отримувати доступ до вебресурсів від імені користувача, не відкриваючи його особистих облікових даних. Ця система спроектована для забезпечення контрольованого доступу до захищених ресурсів. OAuth 2.0 випускає маркери доступу стороннім програмам після одержання згоди користувача. Ці маркери надають обмежений доступ до ресурсів користувача, розміщених на сервері ресурсів. Протокол визначає різні ролі, такі як клієнт, власник ресурсу, сервер ресурсів і сервер авторизації, а також кілька типів надання, таких як код авторизації, неявний, облікові дані клієнта та облікові дані власника ресурсу, призначених для різних типів клієнтів і сценаріїв [40].

Основна концепція ролей є ключовою частиною структури авторизації OAuth 2.0, яка визначає основні складові цієї системи: власник ресурсу та клієнт. Власник ресурсу – це користувач або система, що має контроль над захищеними ресурсами та може дозволити їх використання. Клієнт – це програма або система, яка потребує доступу до захищених ресурсів, для отримання доступу до цих ресурсів клієнт повинен мати відповідний маркер доступу [41].

Крім того, важливими компонентами протоколу OAuth 2.0 є: сервер авторизації і сервер ресурсів. Сервер авторизації – це сервер, який отримує запити на маркери доступу від клієнта та видає їх після успішної автентифікації та згоди власника ресурсу. Сервер авторизації надає дві основні точки доступу: точку авторизації, яка використовується для інтерактивної автентифікації та отримання згоди від користувача, і точку отримання токенів, яка взаємодіє з клієнтом без участі користувача. А сервер ресурсів – це сервер, який захищає ресурси користувача та

обробляє запити на доступ до них від клієнта. Він перевіряє маркер доступу від клієнта та надає відповідні ресурси [40, 41].

Процес взаємодії за протоколом OAuth 2.0 передбачає, що клієнт отримує свої облікові дані від сервера авторизації та взаємодіє з ним для отримання доступу до ресурсів. Запити на доступ ініціюються клієнтом, наприклад, мобільною програмою або вебсайтом, і перевіряються сервером авторизації та сервером ресурсів. Типи надання включають різні стратегії отримання доступу до ресурсів, такі як надання кодів авторизації, неявне надання маркерів доступу або надання облікових даних власника ресурсу чи клієнта [41].

2.1.5 OpenID Connect

OpenID Connect (OIDC) – це розширення для рівня автентифікації, розташоване над протоколом OAuth 2.0. Хоча OAuth 2.0 зосереджений на авторизації, OIDC розширює його функціональність, додавши компонент автентифікації. Одним із ключових аспектів OIDC є концепція ID Token, яка не включена до стандарту OAuth 2.0. Цей токен містить інформацію про особу користувача і використовується в парі з маркером доступу OAuth. OIDC стандартизує процес перевірки особи користувача та механізми, якими клієнти можуть запитувати й отримувати дані про автентифіковані сесії та кінцевих користувачів [40].

OIDC Connect надає можливість розробникам програмних засобів та вебсайтів запускати процеси автентифікації та отримувати перевірені ствердження про користувачів у вебклієнтах, мобільних додатках та додатках на JavaScript. Цей набір специфікацій можна розширити для підтримки різноманітних додаткових функцій, таких як шифрування ідентифікаційних даних, визначення постачальників OpenID та завершення сеансів користувача [42].

2.1.6 Role-Based Authorization

Role-Based Authorization (авторизація на основі ролей) – це метод контролю доступу, який використовується для обмеження доступу до ресурсів на основі ролей користувачів. Ролі – це групи користувачів, яким надано спільні дозволи на доступ до ресурсів [43].

Ролі часто використовуються у фінансових або бізнес-додатках для забезпечення виконання політики. Наприклад, програма може накласти обмеження на розмір транзакції, що обробляється, залежно від того, чи є користувач, який робить запит, членом певної ролі. Захист на основі ролей також можна використовувати, коли програма потребує кількох схвалень для виконання дії. Таким випадком може бути система закупівель, у якій будь-який працівник може створити запит на купівлю, але лише агент із закупівель може перетворити цей запит на замовлення, яке можна надіслати постачальнику [44].

Хоча ролі часто асоціюються з претензіями, проте не кожна претензія є роллю. Відповідно до постачальника ідентифікаційної інформації, роль може відображати набір користувачів, які мають право застосовувати претензії стосовно членів групи, а також саму претензію на ідентифікаційну інформацію. Проте претензії представляють інформацію про окремого користувача. Використання ролей для надання претензій користувачеві може призвести до непорозумінь між самим користувачем та його конкретними претензіями. Це непорозуміння є причиною того, що шаблони Single Page Applications (SPA – односторонні додатки) не розвиваються навколо концепції ролей. Крім того, для організацій, що переходять зі застарілої локальної системи, збільшення кількості ролей протягом багатьох років може призвести до того, що обсяг претензій за роль може стати занадто великим для вміщення у маркері, який використовується SPA [45].

2.2 Шифрування

Шифрування – це процес перетворення даних з читабельного формату (наприклад, текст, зображення, аудіо) в нечитабельний формат (називається шифротекст) за допомогою алгоритму шифрування. Цей процес робить дані незрозумілими для будь-кого, хто не має ключа дешифрування. Ключ дешифрування використовується для зворотного перетворення зашифрованих даних у читабельний формат [46].

2.2.1 Симетричне шифрування

Симетричне шифрування, відоме також як шифрування за допомогою закритого ключа, ґрунтується на чіткій концепції: використання одного єдиного секретного ключа для як шифрування, так і розшифрування інформації. Цей єдиний ключ, який відомий як відправнику, так і отримувачу, гарантує конфіденційність даних [47].

В технології .NET симетричне шифрування виконується на потоках і тому корисно для шифрування великих обсягів даних. У механізмі симетричної криптографії, керовані класи використовують спеціальний потік, відомий як CryptoStream. Цей потік зашифровує дані, які вводяться в нього. При ініціалізації класу CryptoStream використовується керований потік, який реалізує інтерфейс ICryptoTransform, що відповідає криптографічному алгоритму, а також режим CryptoStreamMode, який визначає тип доступу до потоку CryptoStream. Клас CryptoStream може бути ініціалізований будь-яким класом, що походить від класу Stream, таким як FileStream, MemoryStream і NetworkStream. Ці класи дозволяють здійснювати симетричне шифрування різних об’єктів потоку [48].

2.2.2 Асиметричне шифрування

Зазвичай асиметричні алгоритми використовуються для шифрування невеликих обсягів даних, таких як ключі симетричного шифрування та ініціалізаційний вектор. У звичайних умовах особа, що виконує асиметричне шифрування, користується відкритим ключем, який був створений іншою стороною. Для забезпечення цієї мети, технологія .NET надає клас RSA [48].

Клас RSA використовується для реалізації асиметричного алгоритму RSA. Безпека криптографічного методу RSA ґрунтується на обчислювальній проблемі, а саме на розкладанні великих цілих чисел на їх прості множники. Отже, міцність захисту, який надає цей шифр, безпосередньо залежить від розміру ключа. З розвитком обчислювальних потужностей і вдосконаленням алгоритмів розкладання на множники зростає можливість розкладати на прості множники більших чисел [49, 50].

2.2.3 Хешування

Хешування – це процес перетворення даних (наприклад, пароля, тексту або файлу) у фіксований розмір значення, яке називається хешем. Хешування використовується в різних системах безпеки, таких як автентифікація користувачів, цілісність даних та криптографія [51].

При використанні ASP.NET Core Identity, під час реєстрації у програмі, користувач надає свій ідентифікатор (ім’я користувача) та пароль, разом з іншими необхідними даними. Після цього програма генерує хеш для пароля та зберігає його у базі даних, спільно з інформацією про користувача [51, 52].

Зберігання хешу пароля у базі даних є критично важливим аспектом з погляду безпеки. Хеш-функція є односторонньою, що означає, що з пароля можна отримати хеш, але неможливо відновити вихідний пароль за його хешем. З міркувань безпеки обрана хеш-функція має бути обчислювальною складною, щоб затримати спроби злому паролів у разі компрометації бази даних [52].

Важливо відзначити, що паролі користувачів не повинні зберігатися в базі даних у відкритому або шифрованому вигляді, що дозволяє відновлення пароля. Замість цього, рекомендується зберігати лише хеш пароля за допомогою надійної криптографічної хеш-функції, спеціально призначеної для цієї цілі [51, 52].

При спробі входу користувачі надають свій ідентифікатор та пароль. Програма перевіряє наявність облікового запису в базі даних і, якщо такий існує, витягує збережений хеш пароля, пов’язаний з обліковим записом. Після цього програма хешує введений користувачем пароль і порівнює отримані хеші. Якщо вони збігаються, то пароль введено правильно, і користувача можна автентифікувати. У разі ж незбігу хешів вважається, що користувач ввів невірний пароль, і його авторизацію відхиляють [52].

2.3 Контроль доступу

Контроль доступу – це набір методів та механізмів, які використовуються для регулювання доступу до ресурсів, таких як інформація, системи та фізичні об’єкти. Контроль доступу використовується для захисту ресурсів від несанкціонованого доступу, використання та модифікації [53].

2.3.1 Role-Based Access Control

RBAC (Role-Based Access Control) є широко використовуваною моделлю контролю доступу, яка використовує ролі та дозволи для визначення того, хто має доступ до яких ресурсів. Працівники мають доступ лише до інформації, необхідної для ефективного виконання своїх посадових обов’язків. Доступ може ґрунтуватися на кількох факторах, таких як повноваження, відповідальність і професійна компетентність. Крім того, доступ до ресурсів комп’ютера може бути обмежений певними завданнями, такими як можливість перегляду, створення або зміни файлу [54].

Як наслідок, працівники нижчого рівня зазвичай не мають доступу до конфіденційних даних, якщо вони їм не потрібні для виконання своїх обов’язків. Це особливо корисно, якщо у компанії багато співробітників і вона користуєтеся послугами третіх осіб і підрядників, які ускладнюють ретельний моніторинг доступу до мережі. Використання RBAC допоможе захистити конфіденційні дані компанії та важливі програми [54].

RBAC – це стратегічний метод обмеження доступу до ресурсів на основі ролей, які призначені користувачам в організації. Замість надання дозволів безпосередньо користувачам, вони отримують визначені ролі, які регулюють їх рівні доступу. Механізм контролю доступу на основі ролей працює відповідно до принципу найменших привілеїв (PoLP), забезпечуючи користувачам мінімальний необхідний рівень доступу для виконання їх робочих завдань. Однак цей метод не єдиний у своєму роді. Інші варіанти контролю доступу, такі як контроль доступу на основі атрибутів (ABAC), контроль доступу на основі політики (PBAC) та списки контролю доступу (ACL), також є доступними [55].

2.3.2 Token-Based Authentication

Протокол автентифікації на основі токена (token-Based Authentication) визначає процедуру, за допомогою якої користувачі можуть підтвердити свою ідентичність та отримати унікальний токен доступу. Під час дії токена користувачі мають можливість отримувати доступ до вебсайту або програми, для яких був виданий токен, без необхідності повторного введення облікових даних при кожному відвідуванні тієї ж самої вебсторінки, програми або будь-якого іншого ресурсу, захищеного цим токеном. У контексті управління доступом сервери використовують маркери автентифікації для верифікації ідентичності користувача, API, комп’ютера чи іншого сервера [56, 57].

Токени автентифікації функціонують аналогічно проштампованим квиткам. Користувач може скористатися доступом, доки токен залишається дійсним. При виході користувача з системи або закритті програми токен стає недійсним [56].

Можна уявити токени як символічний предмет, який надається надійним джерелом – подібно до значка, який носять правоохоронці, виданий їхнім агентством, що підтверджує їх повноваження. Токени можуть мати фізичний вигляд (наприклад, USB-ключ) або цифровий (комп'ютерне повідомлення чи цифровий підпис) [57].

2.3.3 Безпечний протокол WebSocket

Безпечний протокол WebSocket – це розширення протоколу WebSocket, яке використовує шифрування та автентифікацію для забезпечення безпечного зв'язку між клієнтом і сервером. Це робить його більш стійким до атак типу "людина посередині", крадіжки даних та інших загроз безпеці [58].

Атака типу "людина посередині" (Man-in-the-Middle Attack) – це вид кіберзлочину, де зловмисник, виступаючи як таємний посередник, перехоплює і маніпулює передачею даних між двома сторонами. У цьому сценарії обидві сторони мають враження, що спілкуються безпосередньо одна з одною, тоді як насправді всі дані проходять через зловмисника, який контролює та може змінювати передану інформацію [59].

Протокол WebSocket в ASP.NET Core забезпечує двосторонні постійні канали зв'язку через TCP. Цей протокол використовується в програмах, яким потрібне швидке спілкування в реальному часі, таких як чати, інформаційні панелі та ігри [58, 60].

Застосування WebSockets через HTTP/2 використовує інноваційні можливості, такі як: стиснення заголовків; мультиплексування, яке ефективно зменшує час та ресурси, необхідні для передачі кількох запитів до сервера [58].

2.4 Захист від SQL-ін'єкцій

SQL-ін'єкція (SQLi) являє собою серйозну вразливість в системах веббезпеки, що дозволяє зловмисникам маніпулювати запитами, що виконуються програмою до бази даних. Ця вразливість може викликати непередбачені наслідки, такі як доступ до

конфіденційної інформації, що належить іншим користувачам, або зміну, видалення чи втрату даних [61].

Також зловмисники можуть покращити атаку SQL-ін’єкцій для компрометації базового сервера або іншої серверної інфраструктури. Це може відкрити шлях для подальших атак, таких як атаки відмови в обслуговуванні. Зазвичай атаки SQL-ін’єкції виникають у випадках, коли запитується введення даних від користувача, таких як ім’я користувача або ідентифікатор користувача, і замість очікуваних даних отримується SQL-оператор, який несвідомо виконується у базі даних [61, 62].

Протягом багатьох років атаки SQL-ін’єкції стали причиною багатьох відомих випадків витоку даних, що призвели до серйозної шкоди репутації організацій та суттєвих фінансових втрат через нарахування штрафів. В окремих випадках успішних вторгнень може призвести до створення постійного доступу до систем організації, що може залишатися непоміченим на тривалий період часу, ставши потенційним джерелом довгострокових інцидентів з безпекою [61].

Хоча SQL-ін’єкція є однією з найпоширеніших загроз API, існують ефективні стратегії запобігання цьому типу атаки. До корисних підходів для запобігання SQL-ін’єкціям відносяться: обмеження процедур баз даних, перед тим як використати в системі необхідне проведення процедури фільтрації даних для їх очищення, а також забезпечення доступу з найменшими привілеями [63].

2.4.1 Параметризовані запити

Один із широко використовуваних методів захисту від SQL-ін’єкції полягає у використанні інкапсуляції та параметризації SQL-команд. Параметризовані запити використовуються для відокремлення SQL-запиту від введених користувачем даних, які передаються як параметри. Це дозволяє уникнути виконання потенційно небезпечного коду, оскільки кожен параметр розглядається як літеральне значення та перевіряється на тип і довжину. Коротко кажучи, параметризовані запити – це запити, які вказують параметри замість значень [64, 65].

Метою параметризованих запитів є надання параметрів, потім підключення значень до цих параметрів, а потім виконання запиту. Запити, виконані таким чином, не будуть сприйнятливими до ін’єкційних атак, оскільки запит і параметри будуть надсилатися на виконання окремо [65].

2.4.2 Перевірка вхідних даних

Перевірка вхідних даних – важливий метод захисту від SQL-ін’єкцій в .NET. Перевірка введених даних є процесом, який передбачає перевірку введених користувачем даних на відповідність певним критеріям, перш ніж ці дані будуть оброблені програмою. Цей процес включає кілька методів, таких як перевірка типу даних, перевірка довжини, перевірка діапазону та перевірка формату [66, 67].

Важливість перевірки введених даних полягає у забезпеченні безпеки вебдодатків, що гарантує, що дані, введені користувачем, є безпечними та придатними для використання в програмі. Це перешкоджає атакам ін’єкції зловмисного коду, таких як ін’єкція SQL, яка може ставити під загрозу програму та дані користувача [66].

Методи перевірки вхідних даних включають перевірку типу даних, довжини, діапазону та формату. Наприклад, перевірка типу даних гарантує відповідність формату введених користувачем даних вимогам програми. Перевірка довжини перевіряє, чи не занадто довгі або короткі дані, а перевірка діапазону перевіряє, чи вони відповідають певному діапазону значень. Крім того, перевірка формату впевнюється, що введені дані мають правильний формат для подальшої обробки [66, 67].

Часто методи вашої програми можуть приймати лише певний тип даних, які вважаються прийнятними. Наприклад, у випадку програми ASP.NET MVC, яка отримує ціле число як параметр, вона має приймати лише цілі числа, а не рядки. Вказання правильного типу даних у методі дозволяє забезпечити, що неприпустимі дані не будуть передані, зменшуючи потенційні ризики безпеки [67].

2.5 Захист від XSS-атак

Cross-Site Scripting (XSS) є типом атак на вебдодатки, коли зловмисник впроваджує шкідливі сценарії на надійних вебсайтах. Під час XSS-атак зловмисник використовує вебдодаток для передачі шкідливого коду, який зазвичай має вигляд сценарію, на стороні клієнта, іншому користувачеві, який навіть не підозрює про це. Уразливості, які допомагають здійснити ці атаки, поширені та можуть виникнути в будь-якому місці, де вебдодаток використовує вхідні дані від користувача без їх кодування або перевірки [68].

Уразливості, що стосуються міжсайтових сценаріїв (XSS), зазвичай допомагають зловмисникові приховатися під ідентичність користувача-жертви, виконувати різноманітні дії, доступні для користувача, і отримувати доступ до всіх даних, що належать користувачеві. У випадку, якщо користувач-жертва має привілейований статус у межах програми, зловмисник може здійснювати повний контроль над усіма функціями та ресурсами цієї програми [69].

У випадку XSS зловмисник надсилає шкідливий сценарій користувачеві, який вважає, що це безпечний вміст. У браузері кінцевого користувача може виконувати шкідливі сценарії, вважаючи їх безпечними, що може призвести до отримання доступу зловмисними сценаріями до маркерів сеансу, файлів cookie та іншої конфіденційної інформації, що зберігається вебпереглядачем та використовується на відповідному вебсайті. Ці зловмисні сценарії можуть внести зміни у вміст HTML-сторінок. Цей зловмисний вміст, який передається вебпереглядачу, зазвичай складається з сегментів JavaScript, але може також включати HTML, Flash або будь-який інший тип коду, що спрямований на виконання шкідливих дій. Атаки на основі XSS можуть використовуватися для отримання особистих даних, направлення жертв на контент, що контролюється зловмисником, або виконання інших зловмисних операцій на комп’ютері користувача [68].

В окремих випадках запобігання XSS-атакам може здаватися легким завданням, але в інших випадках воно може стати значно більш складним, залежно від складності

програмного забезпечення та методів обробки даних, що вводяться користувачем [69].

2.5.1 Валідація вхідних даних і кодування виводу

Першим етапом у протидії атакам XSS є очищення всіх введених користувачами даних перед їх відображенням на вебсторінці. Це можна здійснити різними методами, включаючи перевірку вхідних даних (inpot validation) і кодування виводу (output encoding) [70].

Валідація введених даних передбачає аналіз їх на відповідність певним критеріям. Наприклад, якщо очікується введення користувачем адреси електронної пошти, перевірка включатиме перевірку наявності символу "@" та наявності доменного імені. У випадку несумісності вхідних даних з вказаними критеріями, вони будуть відхилені [70, 71].

Кодування виводу полягає в перетворенні введених даних користувача з метою того, щоб вони не інтерпретувалися як код HTML або JavaScript. Це може бути досягнуто шляхом використання різноманітних методів, таких як екранування HTML та кодування JavaScript [70].

Для запобігання атакам XSS вебінтерфейси API мають використовувати як перевірку вхідних даних, так і кодування виводу. Перевірка введених даних гарантує відповідність їхнього формату очікуваним критеріям і відсутність в них шкідливого коду. Кодування виводу забезпечує належну обробку даних, що повертаються API, щоб уникнути їхнього виконання як коду в браузері користувача [71].

2.5.2 HttpOnly cookies

Файли HttpOnly cookies є одним з типів файлів cookie, що передаються та отримуються через протокол HTTP. Вони відокремлені від інших типів файлів cookie і недоступні для зчитування або модифікації через API-інтерфейс document.cookie вебпереглядача. Ця особливість підвищує їх рівень безпеки, оскільки зменшує

вразливість до атак, спрямованих на маніпуляцію файлами cookie з боку шкідливих сценаріїв [72].

Файли cookie лише HTTP часто використовуються для зберігання конфіденційної інформації, такої як маркери сеансу та облікові дані автентифікації, які не повинні бути доступні на клієнтській стороні. Ці файли не запобігають атакам міжсайтового скриптування (XSS), але сприяють у зменшенні можливого впливу цих атак та уникненні необхідності виходу користувачів після виявлення XSS [73].

Наприклад, у разі спроби зловмисника використати XSS для отримання доступу до значень document.cookie та їх передачі на зловмисний сервер, ця спроба буде неуспішною, оскільки файли cookie лише HTTP не доступні через document.cookie. Також, спроби змінити або видалити файли cookie лише HTTP також будуть безрезультатними, оскільки браузер ігнорує будь-які такі зміни. Ці особливості роблять файли cookie лише HTTP ефективними в забезпеченні захисту вебдодатків від крадіжки сесій, витоку особистих даних та інших несанкціонованих дій, які можуть бути здійснені за допомогою файлів cookie [72].

Важливо зауважити, що файли cookies HttpOnly не замінюють заходів захисту від XSS, але зменшують їх вплив та спрощують процес реагування на такі атаки. Ці файли сприяють у запобіганні конкретної загрози, відомої як ексфільтрація (викрадення) маркера сеансу. Ця ситуація виникає, коли зловмисник отримує доступ до маркера сеансу користувача за допомогою атаки XSS. В цьому випадку, коли маркер сеансу зберігається в файлі cookie без прапорця HttpOnly, зловмисник може скористатися цим для виконання дій від імені користувача, що створює серйозні наслідки для безпеки. Але коли маркер сеансу зберігається в файлі cookie з прапорцем HttpOnly, його не можна викрасти безпосередньо через XSS, що допомагає у мінімізації загрози XSS та полегшує реагування на такі атаки [72, 73].

2.5.3 Content-Security-Policy

Політика безпеки вмісту (Content-Security-Policy, CSP) являє собою додатковий рівень захисту, спрямований на виявлення та зменшення певних типів атак, зокрема

XSS-атаки і атаки з впровадженням даних. Ці види атак можуть призвести до крадіжки конфіденційної інформації, пошкодження вебсайтів або поширення шкідливого програмного забезпечення [74].

Основною метою CSP є зменшення ризику XSS-атак і забезпечення відповідного звітування про них. XSS-атаки використовують довіру браузера до вмісту, що надходить з сервера, та виконують шкідливі сценарії в браузері потерпілого. Це стає можливим через довіру браузера до джерела вмісту, навіть якщо воно не є автентичним [74, 75].

Завдяки CSP адміністратори серверів можуть зменшити або усунути вектори атак, вказавши домени, які браузер повинен вважати дійсними джерелами виконуваних сценаріїв. Браузер, що підтримує CSP, буде виконувати лише ті сценарії, які були завантажені з цих дозволених доменів, і ігнорувати всі інші (включаючи вбудовані сценарії та атрибути HTML обробки подій) [74].

CSP є механізмом безпеки, який застосовується вебдодатками для зменшення ризику XSS-атак. Цей механізм дозволяє вебсайтам встановлювати політики, які визначають, які джерела вмісту є безпечними для завантаження на сторінку. Ці політики передаються браузеру через HTTP-заголовок відповіді Content-Security-Policy або метатег у HTML-документі [74, 75].

Після визначення політики безпеки вмісту розробники вебдодатків можуть обмежити типи вмісту, який може бути завантажений та виконаний на сторінці. Це допомагає запобігти виконанню шкідливих сценаріїв, введених зловмисниками. CSP надає кілька директив, таких як "default-src", "script-src", "style-src", "img-src", "frame-src" і "connect-src", які дозволяють налаштовувати ці обмеження зокрема для різних типів вмісту [75].

Важливо зазначити, що CSP працює на основі підходу білого списку, що означає, що дозволені лише визначені джерела, а всі інші будуть автоматично заблоковані. Впровадження CSP дозволяє розробникам значно знизити ризик XSS-атак, оскільки браузер буде блокувати виконання шкідливих сценаріїв, якщо вони не походять з довірених джерел [74, 75].

2.6 Захист від CSRF-атак

Підробка міжсайтового запиту (Cross-Site Request Forgery, CSRF) являє собою вид атаки, яка використовується для надання небажаних дій у вебдодатку, на якому користувач вже автентифікований. За допомогою соціальної інженерії, такої як надсилання спеціальних посилань електронною поштою або через чати, зловмисник змушує користувачів (вебдодатку) виконувати небажані дії. У разі, якщо жертва є звичайним користувачем, успішна атака CSRF може спонукати його виконати запити на зміну стану, такі як переказ коштів або зміна контактної інформації. У випадку, коли об’єктом атаки стає обліковий запис адміністратора, CSRF може призвести до компрометації всього вебдодатку [76, 77].

CSRF – це атака, яка за використанням підроблених запитів намагається виконати дії від імені автентифікованого користувача. Зловмисник використовує ідентифікаційні дані та привілеї жертви для виконання дій, несанкціонованих користувачів. Оскільки браузер автоматично включає облікові дані, пов’язані з вебсайтом, до кожного запиту, включаючи файли cookie сесії користувача, IP-адресу та облікові дані домену Windows, вебдодаток не може розрізнити між легітимними та підробленими запитами [76].

CSRF-атаки націлені на функції, які змінюють стан сервера, такі як зміна особистих даних чи проведення платежів. Оскільки зловмиснику не потрібно отримувати відповідь на запит, а саме жертва отримує її, атаки спрямовані на зміну стану. Зловмисник може використовувати CSRF для отримання особистих даних жертви, змушуючи її ввести їх до вебдодатка через спеціальний запит. Також, CSRF може бути використаний для зламу облікового запису жертви, щоб переглянути її особисті дані та активність у додатку [77].

Іноді CSRF-атаку можна виконати безпосередньо на вразливому вебсайті. Це відомо як "збережена CSRF-помилка". Наприклад, це можна здійснити, вставивши тег IMG або IFRAME у поле для введення HTML або використовуючи складні міжсайтові атаки. Враховуючи те, що жертва з більшою ймовірністю перегляне вміст

атаки на своєму вебсайті та вже автентифікована, це може підвищити серйозність атаки [76].

Атака CSRF, попри те, що вона потенційно катастрофічна, є застарілим типом загрози безпеці, і більшість мов/фреймворків уже мають вбудований захист від неї. Основна ідея полягає в тому, щоб придумати спосіб зробити запити неможливими передбачити, включивши в сам запит деяку інформацію, яку неможливо підробити та змінюється з кожним запитом. В технології .NET для захисту від CSRF-атак використовуються анти-CSRF-токени, HTTP-заголовки, а також SamSite cookies [77, 78].

2.6.1 Анти-CSRF-токен

Анти-CSRF-токени, також відомі як токени запобігання міжсайтовому підробленню запитів, є важливим елементом забезпечення безпеки вебдодатків, оскільки вони допомагають зменшити ризик використання атак CSRF. Атаки цього типу експлуатують довіру вебдодатка до браузера користувача з метою виконання несанкціонованих дій від імені останнього. Ця ситуація може призвести до серйозних наслідків, включаючи несанкціонований доступ до ресурсів, витік конфіденційної інформації або навіть фінансові втрати [79, 80].

Анти-CSRF-токени являє собою випадково згенеровані унікальні значення, які вбудовуються в HTML-форми або URL-адреси вебдодатка. Ці токени асоційовані з сеансом користувача та використовуються для перевірки автентичності наступних запитів, надісланих від імені цього користувача. Головна мета анти-CSRF-токенів полягає в переконанні, що всі запити, що надходять від браузера користувача, є законними та не були підроблені зловмисниками [80].

Наприклад, користувач авторизується на вебсайті банку. Вебсайт використовує анти-CSRF-токени для захисту від атак CSRF. При виконанні конфіденційних операцій, наприклад, переказу коштів, вебдодаток генерує унікальний анти-CSRF-токен та включає його у приховане поле HTML-форми. Цей токен також асоційований із сеансом користувача на серверному боці [79].

Якщо зловмисник спробує перехопити запит від імені користувача, анти-CSRF-токен виконає функцію захисту. Зловмисник не матиме дійсного анти-CSRF-токена, тому підроблений запит буде відхилений. При отриманні запиту вебдодаток порівнює надісланий токен з тим, який асоційований із сеансом користувача. Якщо вони не збігаються, запит вважається неавторизованим і відкидається [77, 79].

Крім того, анти-CSRF-токени ефективно захищаються від атак, таких як фіксація та керування сеансами. Фіксація сеансу відбувається, коли зловмисник намагається використати попередньо визначений сеанс, тоді як керування сеансами передбачає використання активного сеансу. Анти-CSRF-токени, які є унікальними та випадковими для кожного сеансу, ускладнюють спроби зловмисників передбачити або маніпулювати ними, тим самим унеможливлюючи подібні атаки [79].

2.6.2 SameSite cookies

SameSite є одним з атрибутів файлів cookie (схожий на HTTPOnly, Secure тощо) і спрямований на зменшення вразливості до атак CSRF. Цей атрибут визначений у RFC6265bis. SameSite допомагає браузеру вирішити, чи надсилати файли cookie разом з міжсайтовими запитами. Для цього атрибута можливі значення: Lax, Strict або None [77, 80].

Значення Strict заважає браузеру надсилати файли cookie на цільовий сайт у будь-якому міжсайтовому контексті перегляду, навіть якщо користувач переходить за звичайним посиланням. Наприклад, якщо вебсайт, подібний до GitHub, використовує значення Strict, і користувач GitHub намагається перейти за посиланням на приватний проект GitHub, розміщений на корпоративному форумі чи в електронній пошті, то користувач не отримає доступ до проекту, оскільки GitHub не отримає сеансовий файл cookie. Для вебсайту банку, який не дозволяє посилання на сторінки транзакцій з зовнішніх джерел, строгий режим (Strict) є найбільш відповідним [77].

Значення Lax SameSite за замовчуванням забезпечує розумний баланс між безпекою та зручністю використання. Наприклад, у сценарії з GitHub, якщо

використовується значення Lax, файли cookie сеансу будуть надсилатися при переході за звичайним посиланням з зовнішніх вебсайтів, але будуть блокуватися в методах запиту, що можуть бути використані для CSRF, таких як POST [77].

SameSite є проектом стандарту IETF, який був оновлений у 2019 році. Оновлений стандарт несумісний з попереднім і має деякі відмінності, такі як обробка файлів cookie без заголовка SameSite як Lax за замовчуванням, а також потреба у маркуванні файлів cookie SameSite=None, які також повинні бути позначені як Secure. Крім того, фрейми можуть мати проблеми з використанням файлів cookie з атрибутами SameSite=Lax або SameSite=Strict, оскільки вони розглядаються як міжсайтові [80].

2.7 Порівняння методів захисту технології .NET

Порівняння методів захисту технології .NET, за допомогою їх особливостей, переваг і недоліків зображено в табл. 2.1.

Таблиця 2.1 – Переваги та недоліки методів захисту в технології .NET

Метод захисту	Опис	Переваги	Недоліки
1. Автентифікація та авторизація			
Windows Authentication	Використовує вбудовані механізми автентифікації Windows для перевірки користувачів	Проста реалізація та налаштування; високий рівень безпеки шляхом інтеграції з Windows; не потребує збереження паролів на стороні клієнта.	Обмежене налаштування та контроль автентифікації, не підходить для незалежних вебсистем

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
Forms Authentication	Користувачі вводять свої імена користувачів та паролі на вебформі; паролі зберігаються на стороні сервера.	Простий у реалізації; гнучкий, оскільки можна налаштувати сторінки входу та помилок; не потребує Active Directory	Вразливий до атак типу перебору та фішингу, якщо не вжито додаткових заходів безпеки; ризик витоку паролів; потребує додаткової конфігурації
Claims-based Authentication (автентифікація на основі тверджень)	Використовує токени тверджень для автентифікації користувачів; токени містять інформацію про користувача	Дозволяє включати додаткові атрибути користувача в токен автентифікації, що дозволяє більш гнучкий контроль доступу; можливість реалізувати єдиний вхід між різними застосунками	Потребує додаткової розробки для інтеграції та керування заявками користувача; можливі складнощі для користувачів у розумінні токенів
OAuth 2.0	Надання вебсайтам доступу до даних користувачів з інших вебсайтів	Безпечність та надійність; зручність для користувачів; підходить для розподілених систем	Складність реалізації; залежність від сторонніх сервісів; вразливість до певних атак

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
OpenID Connect (OIDC)	OIDC є надбудовою над OAuth 2.0, яка надає додаткові можливості для автентифікації та авторизації; OIDC використовує JSON Web Tokens (JWT) для передачі інформації про користувача між вебдодатком та IdP	OIDC надає вебдодаткам більше інформації про користувача, наприклад, ім’я, адресу електронної пошти та зображення профілю; OIDC спрощує реалізацію OAuth 2.0; OIDC використовує JWT для захисту інформації про користувача	OIDC потребує реалізації OAuth 2.0, не всі IdP підтримують OIDC
Role-Based Authorization (авторизація на основі ролей)	Контроль доступу на основі ролей користувачів; призначення ролей користувачам з визначенням доступу до ресурсів.	Легкість керування доступом до ресурсів; зниження навантаження на адміністраторів; підходить для великих систем	Потребує додаткової конфігурації; можлива складність реалізації; не підходить для простих вебсайтів
2. Шифрування			
Симетричне шифрування	Застосовує один ключ для як шифрування, так і розшифрування даних	Швидкий та ефективний процес шифрування та розшифрування даних	Потребує безпечної передачі та зберігання спільного ключа

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
Асиметричне шифрування	Застосовує пару ключів: публічний і приватний для шифрування та розшифрування даних	Вища безпека, оскільки використовується пара ключів, один з яких залишається приватним	Швидкодія обміну даними зменшується порівняно з симетричним шифруванням
Хешування	Використовується для генерації унікального фіксованого рядка з великого набору даних	Незворотний процес, що гарантує безпеку відновлення даних	Вразливий до атак перебору, оскільки немає можливості розшифрувати хешований результат
3. Контроль доступу			
Role-Based Access Control (ролі та дозволи)	Надає різні рівні доступу для користувачів на основі їх ролей і дозволів	Легко додавати, видаляти та змінювати ролі та дозволи; підходить для великих систем з багатьма користувачами та ресурсами; є можливість створювати складні правила доступу, ґрунтуючись на різних факторах	Потребує додаткової конфігурації та налаштування; складні правила доступу можуть бути важкими для розуміння та керування; може бути надмірним для невеликих систем

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
Token-Based Authentication (автентифікація на основі токенів)	Використовує токени для автентифікації користувачів	Можна використовувати з різними постачальниками автентифікації та протоколами; підходить для великих систем; токени захищені цифровим підписом, що гарантує їх надійність; автентифікація та авторизація обробляються окремо, що робить систему більш гнучкою	Потребує додаткової конфігурації та налаштування; користувачам може знадобитися зрозуміти, як працюють токени; не всі вебсайти та постачальники автентифікації підтримують автентифікацію на основі токенів
Безпечний протокол WebSocket	Протокол двосторонньої комунікації, який дозволяє встановлювати постійні з’єднання між клієнтом і сервером	WebSocket забезпечує низьку затримку; використовує менше ресурсів, ніж традиційні HTTP-запити, що робить його більш ефективним для потокових даних	Складність впровадження: WebSocket потребує більше знань з розробки; WebSocket не сумісний з багатьма браузерами та серверами

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
4. Захист від SQL-ін’єкцій			
Параметризовані запити	Використовується для передачі параметрів в SQL-запити, що ускладнює їх вплив на базу даних	Найефективніший метод захисту від SQL-ін’єкцій; запобігає виконанню довільного SQL-коду: простий у реалізації.	Може бути складним у використанні на початковому етапі розробки, особливо для великих систем; необхідно трохи змінити наявний код
Перевірка вхідних даних	Використовується для перевірки вхідних даних на наявність небезпечних символів або конструкцій	Може використовуватися разом з іншими методами захисту; забезпечує додатковий рівень безпеки; відносно простий у реалізації	Може бути не таким ефективним, як параметризовані запити, потребує ретельного тестування, щоб гарантувати, що він не блокує легітимні дані, може вплинути на продуктивність
5. Захист від XSS-атак			
Перевірка вхідних даних (Input Validation)	Перевіряє вхідні дані на наявність HTML або JavaScript коду, який може виконатися	Ефективність проти XSS-атак; простий у реалізації; може використовуватися для захисту різних типів даних	Може бути складним у визначенні всіх потенційно небезпечних символів та конструкцій

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
Кодування виводу (Output encoding)	Перетворення вхідних даних у безпечний формат перед виведенням їх у браузері користувача	Ефективність проти XSS-атак; простий у реалізації; може використовуватися для захисту різних типів даних	Може зробити вихідні дані нечитабельними; може призвести до проблем з форматуванням
HttpOnly cookies	Дозволяє обмежити доступ JavaScript до cookies, зменшуючи ризик XSS-атаки	Застосування HttpOnly атрибута для cookies у браузері, що унеможливлює доступ JavaScript до цих cookies; простий у реалізації	Деякі старі браузери не підтримують HttpOnly cookies; не захищає від інших типів XSS-атак; може ускладнити доступ до cookie за допомогою JavaScript на стороні клієнта.
Content-Security-Policy (CSP)	Заголовок HTTP, який дозволяє вебсайту вказувати, які джерела дозволено використовувати для завантаження контенту (скрипти, стилі, зображення тощо)	Потужний метод захисту від XSS-атак; може захистити від інших типів атак; забезпечує гнучкий контроль над тим, які ресурси дозволено завантажувати на вебсторінку	Складний у реалізації: потрібно ретельно налаштувати, щоб уникнути порушення роботи сторінки; не всі браузери підтримують CSP

Продовження таблиці 2.1

Метод захисту	Опис	Переваги	Недоліки
6. Захист від CSRF-атак			
Анти-CSRF-токен	Генерує унікальний токен для кожного запиту, що використовується для перевірки відправника	Ефективний проти CSRF-атак; простий у реалізації	Може ускладнити форми та запити; потрібно ретельно тестувати, щоб уникнути помилок передачу їх на сторону клієнта
SameSite cookies	SameSite – це атрибут cookie, який можна використовувати для контролю того, як cookie надсилається разом із HTTP-запитами	Зменшує ризик використання CSRF-атак, оскільки cookie із сеансовою інформацією не надсилаються стороннім вебсайтам, також простий у реалізації	Не повністю захищає від CSRF-атак; може призвести до проблем із сумісністю з деякими вебсайтами

Висновок за розділом 2

У другому розділі дипломної роботи було ретельно досліджено методи захисту в технології .NET. Розділ включав огляд різноманітних засобів автентифікації та авторизації, включаючи Windows Authentication, Forms Authentication, Claims-based Authentication, Role-Based Authorization, OAuth 2.0 та OpenID Connect. Також розглядалися методи шифрування (симетричне, асиметричне, хешування), контролю доступу (Role-Based Access Control, Token-Based Authentication та використання протокол WebSocket), захисту від SQL-ін’єкцій (параметризовані запити, перевірка

вхідних даних), XSS-атак (валідація вхідних даних і кодування виводу, HttpOnly cookies, Content-Security-Policy) та CSRF-атак (анти-CSRF-токен, SameSite cookies).

Аналізуючи різні методи автентифікації, виявлено, що кожен з них має свої переваги та обмеження. Наприклад, Windows Authentication дозволяє ідентифікувати користувачів з використанням їхнього Windows облікового запису, тоді як Forms Authentication надає більшу гнучкість, але вимагає додаткової уваги до забезпечення безпеки. Тому було виявлено, що використання комбінації різних методів захисту є ключовим для забезпечення високого рівня безпеки вебдодатків на базі технології .NET. Наприклад, для захисту від XSS-атак рекомендується використовувати валідацію вхідних даних, кодування виводу, HttpOnly cookies та інші заходи, які були описані в цьому розділі. Крім того, важливо постійне оновлення та вдосконалення методів захисту з урахуванням останніх тенденцій у цій сфері.

Отже, дослідження технології .NET в контексті її захисту від потенційних загроз підкреслює необхідність глибокого розуміння різноманітних методів та їх інтеграції, а також постійного оновлення та вдосконалення методів захисту для забезпечення високого рівня безпеки вебдодатків.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ

3.1 Створення додатка ASP.NET Core Web App

Розроблена комплексна система захисту інформації за допомогою створення вебдодатка, використовуючи середовище Visual Studio 2022. Для реалізації було створено новий проект за шаблоном ASP.NET Core Web App (Razor Pages) під назвою “WebApp1”.

Вибраний шаблон ASP.NET Core Web App (Razor Pages), тому що він простий у використанні та добре документований, а також сумісний з різними інструментами й бібліотеками. Проект використовує останню версію технології .NET, а саме .NET 8.0. Для написання програми для виконання методів захисту використана мова C#. А для структури додатка використана мова HTML.

Для даних користувачів була використала база даних Microsoft SQL Server. Тому для зручності перед налаштовуванням додатка її підключили і налаштували (рис. 3.1)

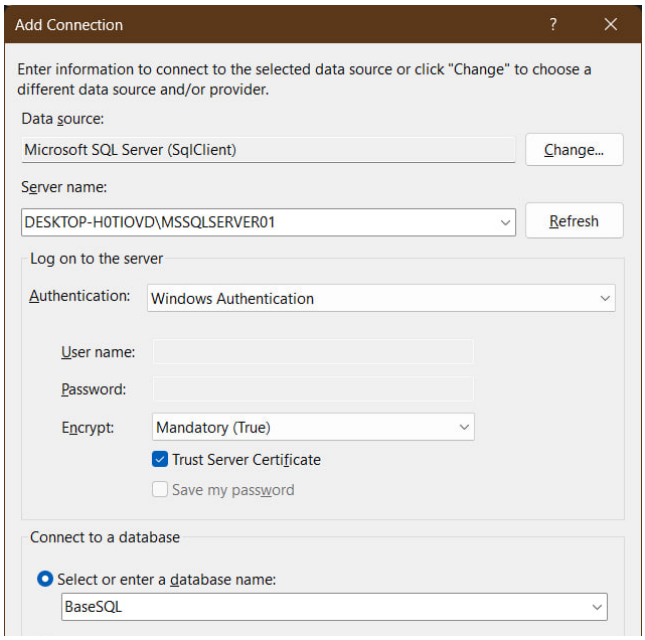

Рисунок 3.1 – Підключення бази даних

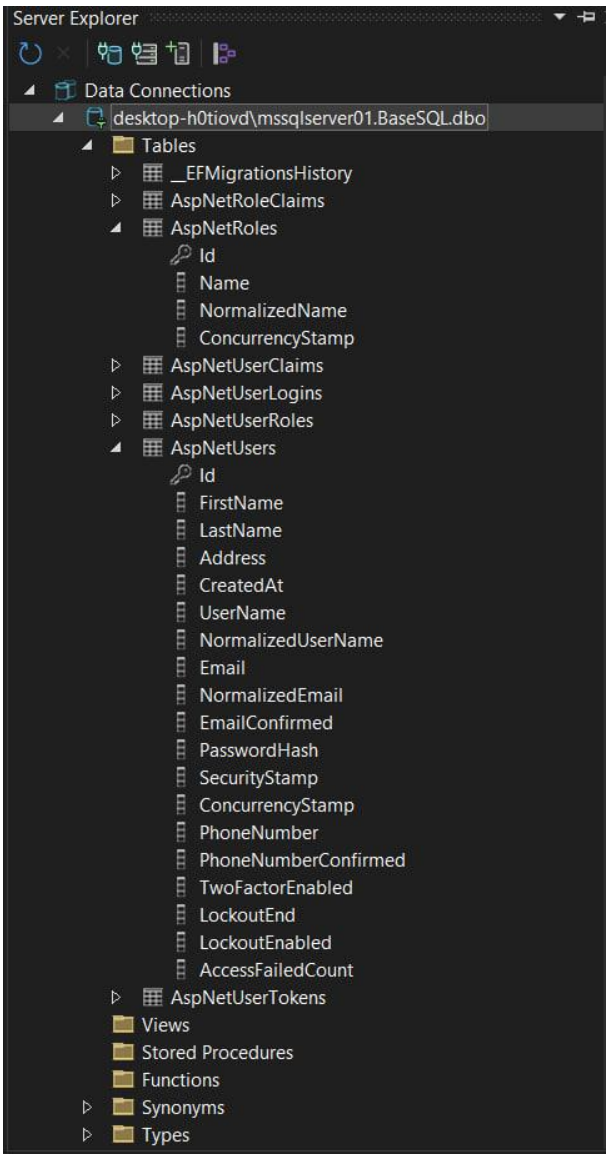

Рисунок 3.2 – Створена і налаштована база даних

Для коректної роботи додатка, підключеної бази, а також всіх можливих методів захисту завантажили пакети, які показано на рис. 3.3.

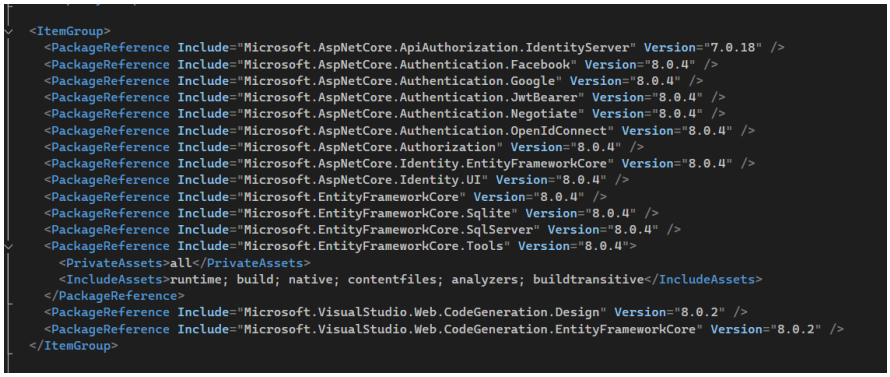

Рисунок 3.3 – Інстальовані пакети

Microsoft.AspNetCore.ApiAuthorization.IdentityServer допомагає захистити вебдодаток за допомогою протоколу OpenID Connect сервера IdentityServer. Він інтегрує управління користувачами ASP.NET Core Identity із цим протоколом для безпечного контролю доступу.

Microsoft.AspNetCore.Authentication.Facebook дозволяє користувачам входити до вашого застосунку за допомогою облікових записів Facebook, що спрощує процес входу.

Microsoft.AspNetCore.Authentication.Google дозволяє користувачам входити до вашого застосунку за допомогою облікових записів Google, що спрощує процес входу.

Microsoft.AspNetCore.Authentication.JwtBearer реалізує автентифікацію за допомогою JSON Web Token (JWT) – популярного механізму для захисту API. Користувачі обмінюють свої дані для входу на токени, які потім можна використовувати для доступу до захищених ресурсів.

Microsoft.AspNetCore.Authentication.Negotiate дозволяє інтегрувати автентифікацію Windows (WIA) у ваш застосунок. Користувачі автоматично автентифікуються на основі своїх облікових даних Windows, забезпечуючи безпроблемний вхід у середовищах Windows.

Microsoft.AspNetCore.Authentication.OpenIdConnect пропонує загальний клієнт OpenID Connect (OIDC) для автентифікації користувачів із різними постачальниками OIDC, окрім Facebook і Google. Це розширює можливості вибору постачальників послуг авторизації.

Microsoft.AspNetCore.Authorization є базовим пакетом для визначення логіки авторизації в програмах ASP.NET Core. Він дозволяє створювати правила, що контролюють доступ до певних контролерів, дій або ресурсів на основі ролей користувачів, претензій або інших критеріїв.

Microsoft.AspNetCore.Identity.EntityFrameworkCore інтегрує ASP.NET Core Identity з Entity Framework Core для управління користувачами в реляційній базі даних. Він пропонує функції реєстрації, входу в систему, керування паролями та авторизації на основі ролей.

Microsoft.AspNetCore.Identity.UI надає готові компоненти користувальницького інтерфейсу (Razor Pages) для таких задач, як реєстрація, вхід в систему, скидання пароля та керування профілем. Це гарна starting point для створення інтерфейсів, що взаємодіють із системою ASP.NET Core Identity.

Microsoft.EntityFrameworkCore – ядро для роботи з Entity Framework Core, об'єктно-реляційним відображенням (ORM), що спрощує доступ до даних у програмах ASP.NET Core. Воно дозволяє взаємодіяти з реляційними базами даних за допомогою коду C#.

Microsoft.EntityFrameworkCore.Sqlite, Microsoft.EntityFrameworkCore.SqlServer – постачальники для Entity Framework Core, що дозволяють використовувати відповідно SQLite або SQL Server як базу даних для вашого застосунку.

Microsoft.EntityFrameworkCore.Tools – пакет командних рядків для роботи з Entity Framework Core із командного рядка.

Після завантаження пакетів і підключення бази даних було додано Identity Pages для подальшого налаштування вигляду додатка, а також методів безпеки додатка (рис. 3.4-3.6). Це набагато спрощує створення макетів, а також дає можливість використовувати HTML, CSS та JavaScript для налаштування дизайну.

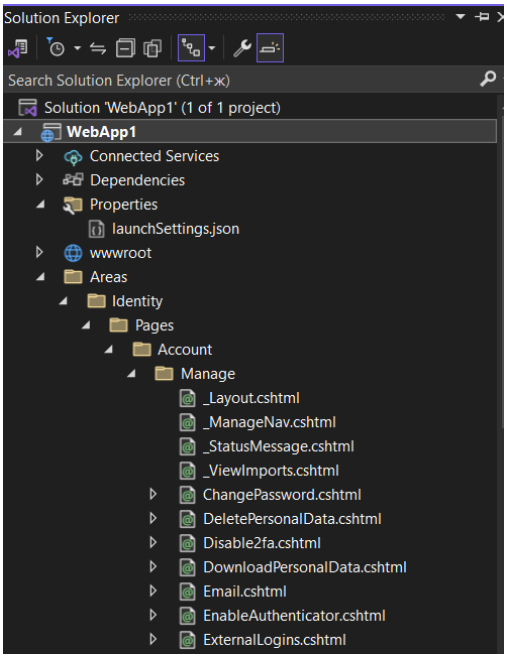

Рисунок 3.4 – Структура створених файлів (частина 1)

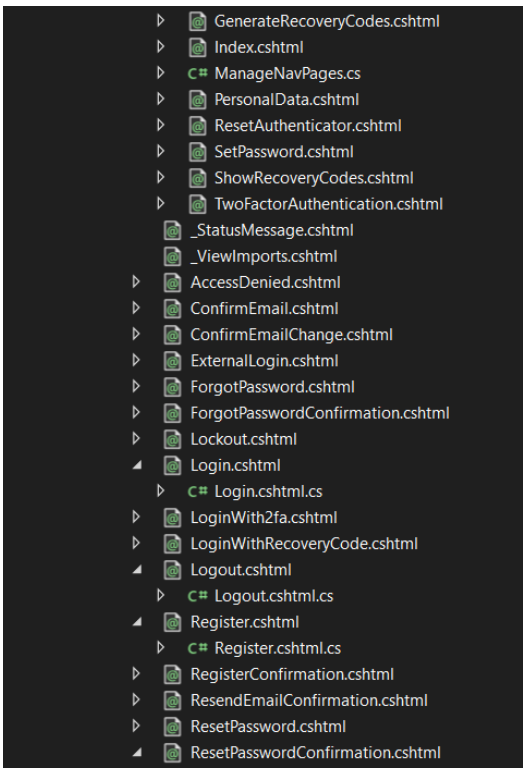

Рисунок 3.5 – Структура створених файлів (частина 2)

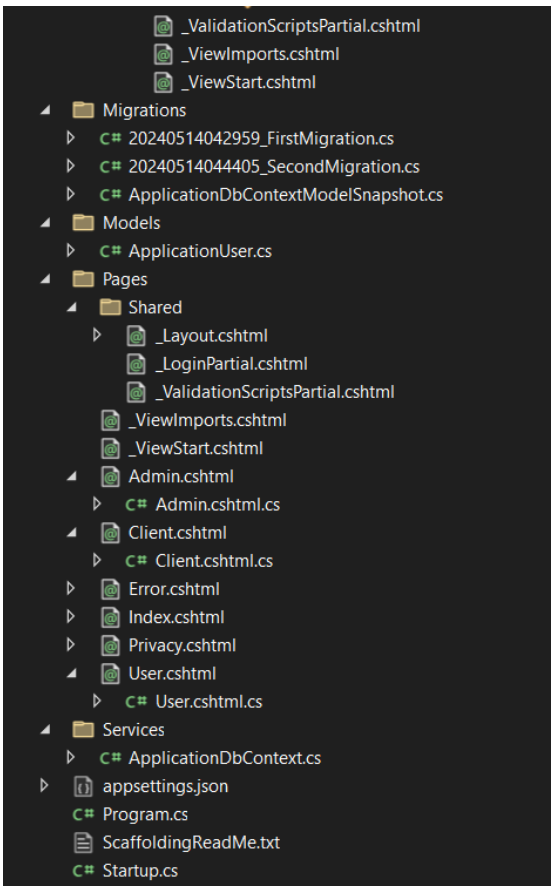

Рисунок 3.6 – Структура створених файлів (частина 3)

Вигляд головної сторінки було налаштовано за допомогою файлу Index.cshtml, а структура панелі керування за допомогою _LoginPartial.cshtml та _Loyout.cshtnl як показано на рис. 3.7. Код програми цього файлу та подальших файлів налаштувань буде знаходитись в Додатку Б.

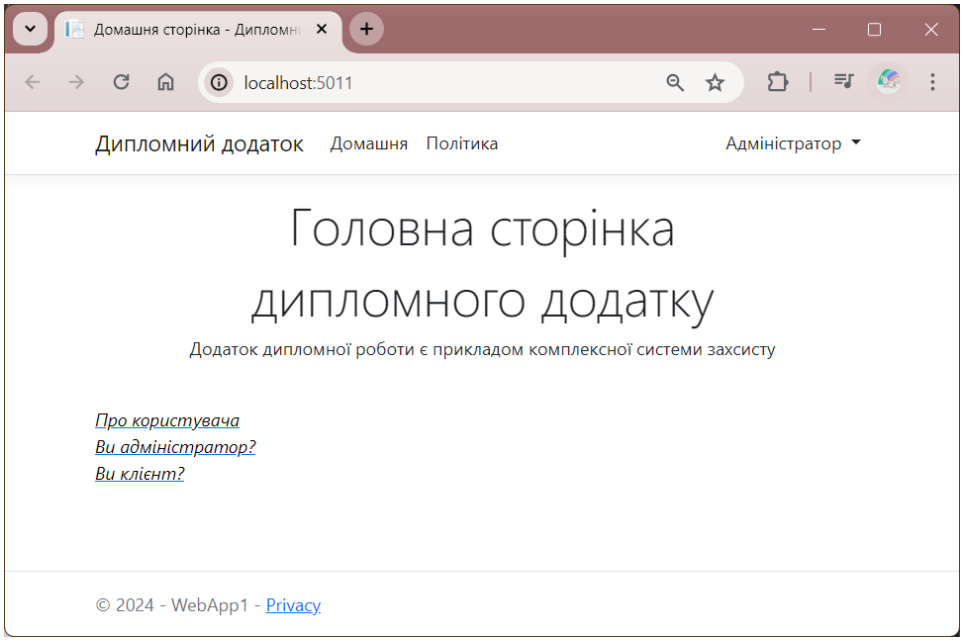

Рисунок 3.6 – Інтерфейс головної сторінки

У додатку є також додаткова сторінка крім головної під назвою «Політика», яка налаштована за допомогою файлу Privacy.cshtml (рис. 3.7)

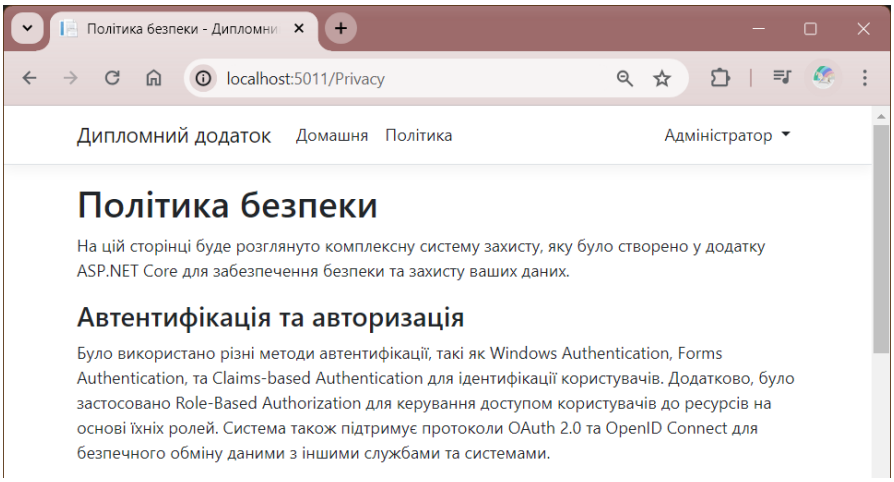

Рисунок 3.7 – Інтерфейс додаткової сторінки

Реєстрація користувача створена за допомогою файлу Register.cshtml та Register.cshtml.cs, було створено категорії для основних даних реєстрації, а саме: ім’я, прізвище, пошта, пароль і повтор пароля як показано на рис. 3.8. Всі дані повинні бути введені, а також пароль повинен бути достатньо захищений, щоб зареєструватись як показано на рис. 3.9.

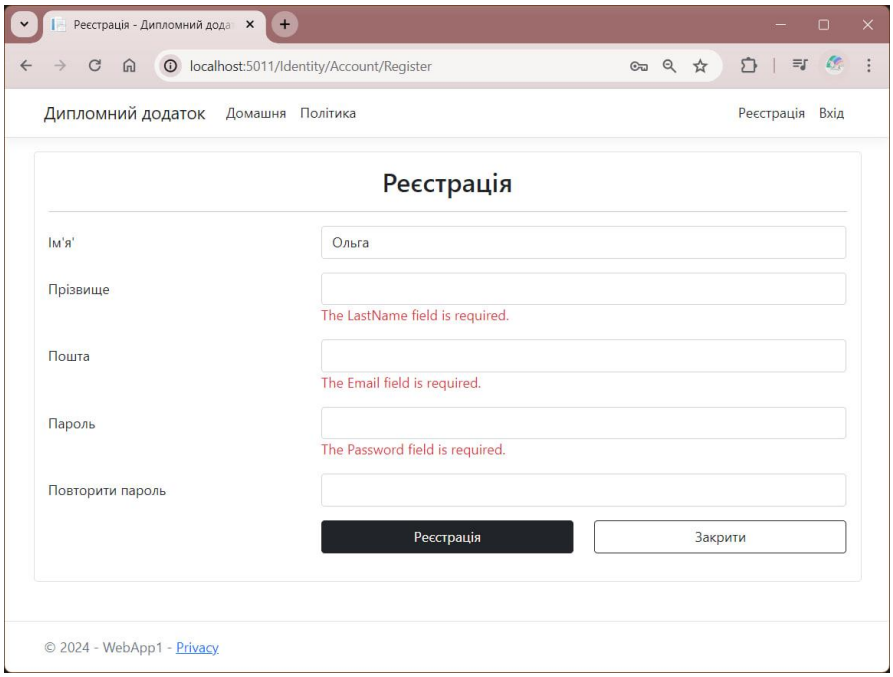

Рисунок 3.9 – Інтерфейс сторінки реєстрації

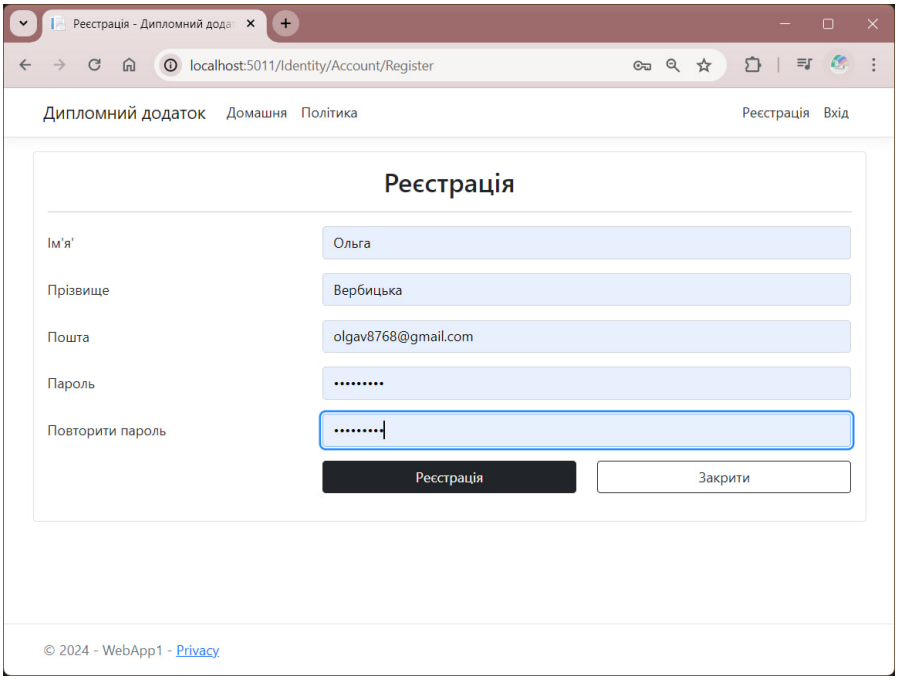

Рисунок 3.10 – Приклад коректної реєстрації користувача

Вхід в систему створений за допомогою Login.cshtml та Login.cshtml.cs як показано на рис. 3.11-12.

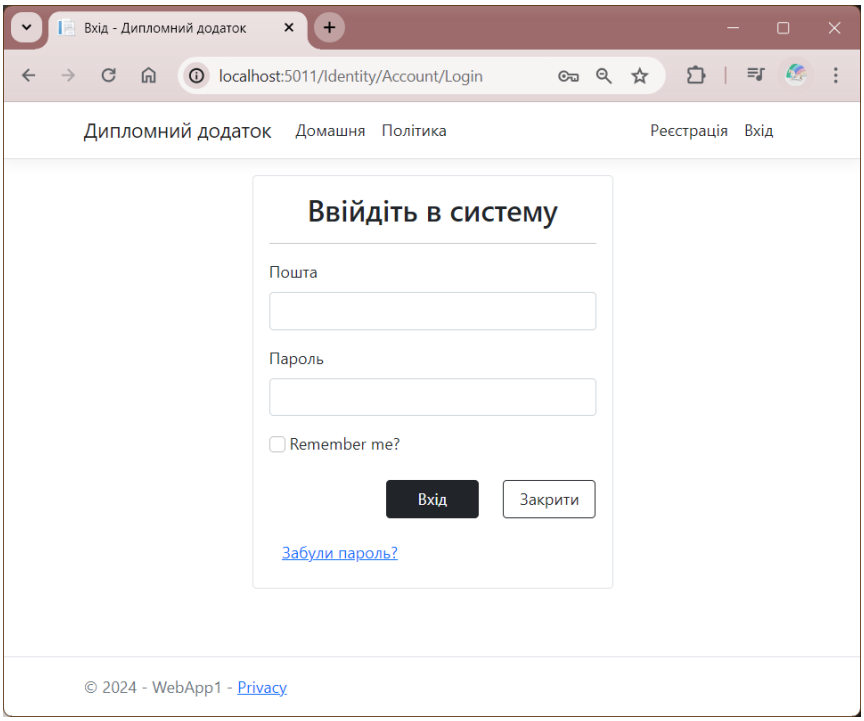

Рисунок 3.11 – Інтерфейс входу в систему

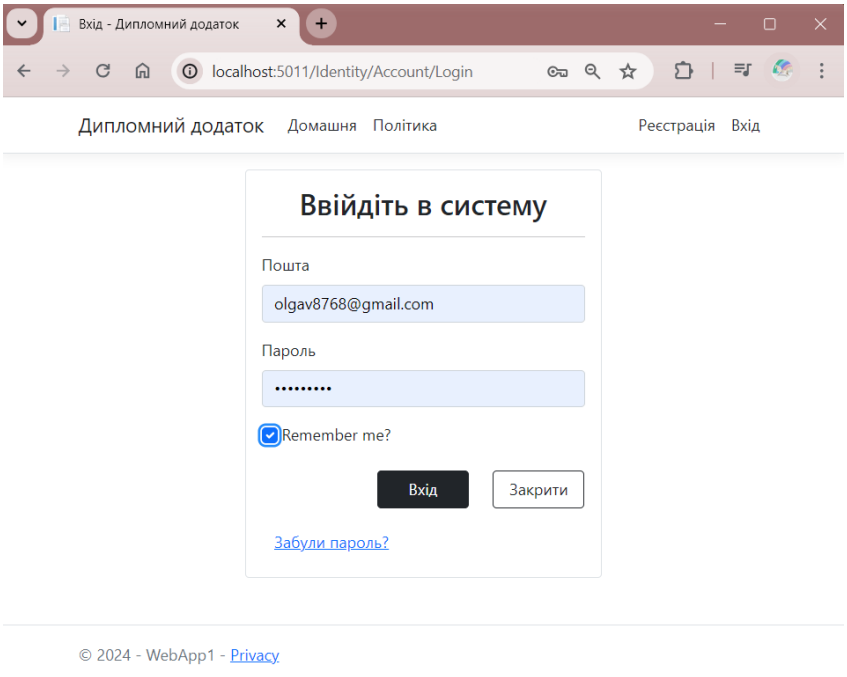

Рисунок 3.12 – Приклад коректного входу в систему

Якщо користувач не пам’ятає пароля, то його можна відправити на пошту як показано на рис. 3.13-3.15. Для надсилання електронних листів було змінено файл appsettings.Development.json. Для налаштування функції надсилання листів було створено SendSettings.cs та EmailSender.cs., а також у файлі Program.cs було створено потрібний клас для надсилання листів.

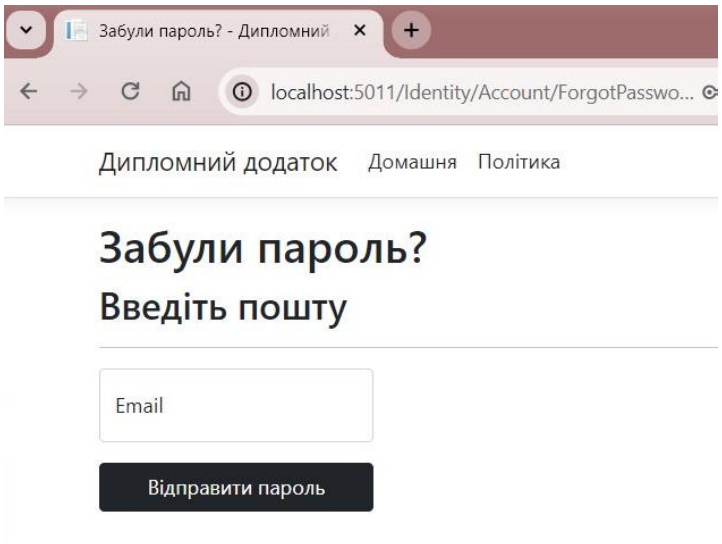

Рисунок 3.13 – Інтерфейс для посилання пароля

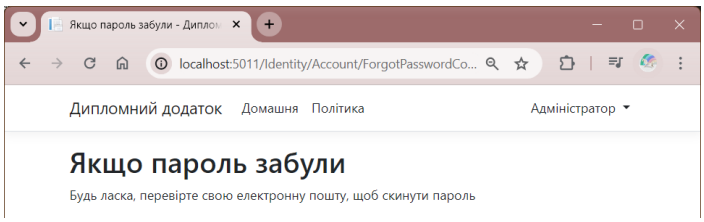

Рисунок 3.14 – Підтвердження посилання пароля на пошту

Були налаштовані інтерфейс та функціонал виходу з акаунту за допомогою файлів Logout.cshtml та Logout.cshtml.cs (рис. 3.15).

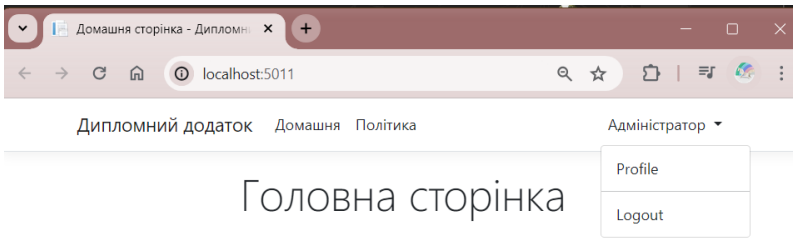

Рисунок 3.15 – Приклад виходу з системи

Також в додатку можна переглянути профіль користувача, де можна відредагувати профіль і додати номер телефону (рис. 3.16), змінити і підтвердити пошту (рис. 3.17), змінити пароль (рис. 3.18), додати двофакторну автентифікацію (рис. 3.19), а також користувач може завантажити або видалити особисті дані (рис. 3.20).

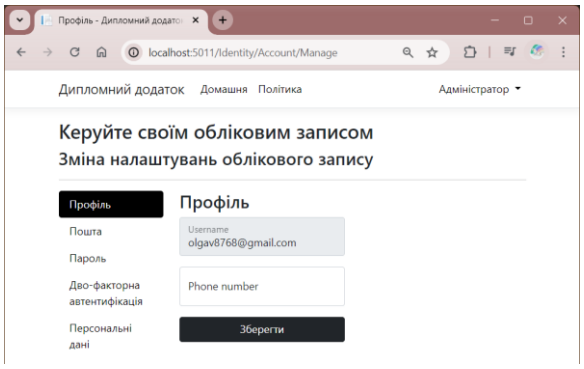

Рисунок 3.16 – Інтерфейс редагування профілю користувача

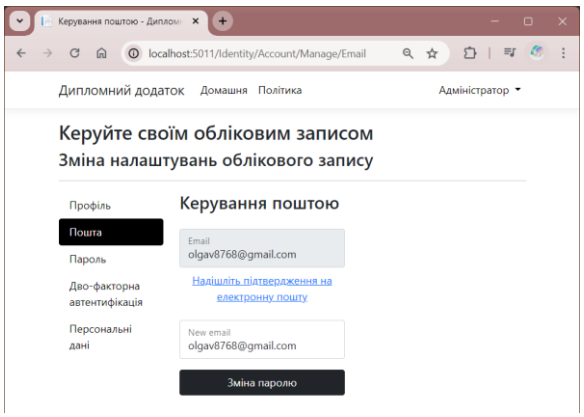

Рисунок 3.17 – Інтерфейс керування поштою користувача

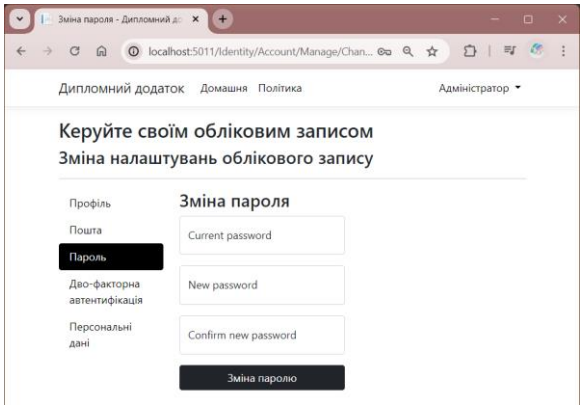

Рисунок 3.17 – Інтерфейс керування паролем

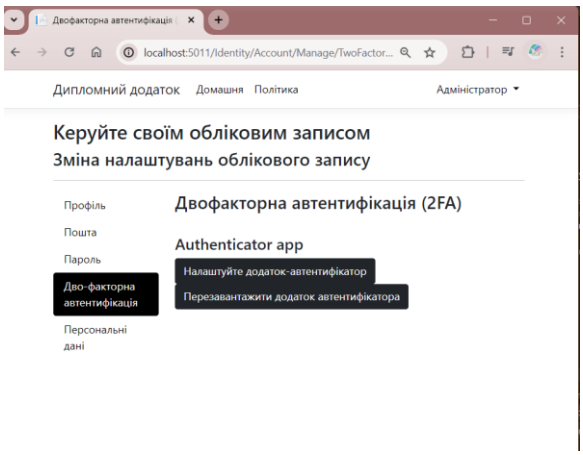

Рисунок 3.17 – Інтерфейс додавання двофакторної автентифікації

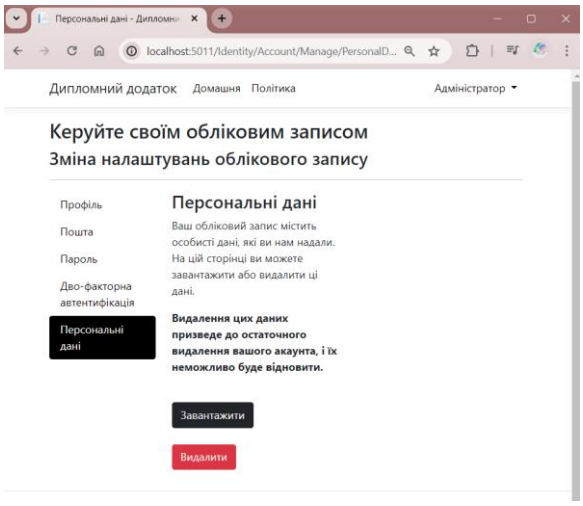

Рисунок 3.18 – Інтерфейс керування особистими даними

3.2 Реалізація методів захисту

У додатку було реалізовано Forms Authentication і Claims-based Authentication через форми реєстрації та входу (як показано вище), а також збереження інформації у базі даних (рис. 3.19) за даними, які показані на рис. 3.20.

	Id	FirstName	LastName	Address	CreatedAt	UserName	NormalizedUs...	Email	NormalizedEm...	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencySt...	PhoneNumber	PhoneNu
	10378bd0-76fa...	Вол	Лlic		01.01.0001 0:00:...	vovfvvr8768@g...	VOVFVVR8768...	vovfvvr8768@g...	VOVFVVR8768...	False	AQAAAAIAAYa...	WYTHKQYJ7AN...	4040110c-9491...	NULL	False
	46dc3b75-864e...	Bill	KLc		01.01.0001 0:00:...	olgavfvdvfgv87...	OLGAVFVDVFG...	olgavfvdvfgv87...	OLGAVFVDVFG...	False	AQAAAAIAAYa...	BD5SNBAMFKS...	e95b238e-9fc7-...	NULL	False
	9a1cbf30-9503-...	Ольга	Вербицька		01.01.0001 0:00:...	olgav8768@gm...	OLGAV8768@G...	olgav8768@gm...	OLGAV8768@G...	False	AQAAAAIAAYa...	XAC4L7536YUM...	05a530eb-05dd...	NULL	False
	9e07a3b2-0d11...	Вол	Лlic		01.01.0001 0:00:...	vov8768@gmai...	VOV8768@GM...	vov8768@gmai...	VOV8768@GM...	False	AQAAAAIAAYa...	J7D4WFH6KKA...	57c23a92-0573...	NULL	False
	a2300fed-f9b6-...	Billfre	KLcfere		01.01.0001 0:00:...	olgadecvfvdvfg...	OLGADECVFVD...	olgadecvfvdvfg...	OLGADECVFVD...	False	AQAAAAIAAYa...	ZMK7HOK7CG...	24cf1fd6-8cd1-...	NULL	False

Рисунок 3.19 – Вигляд бази даних

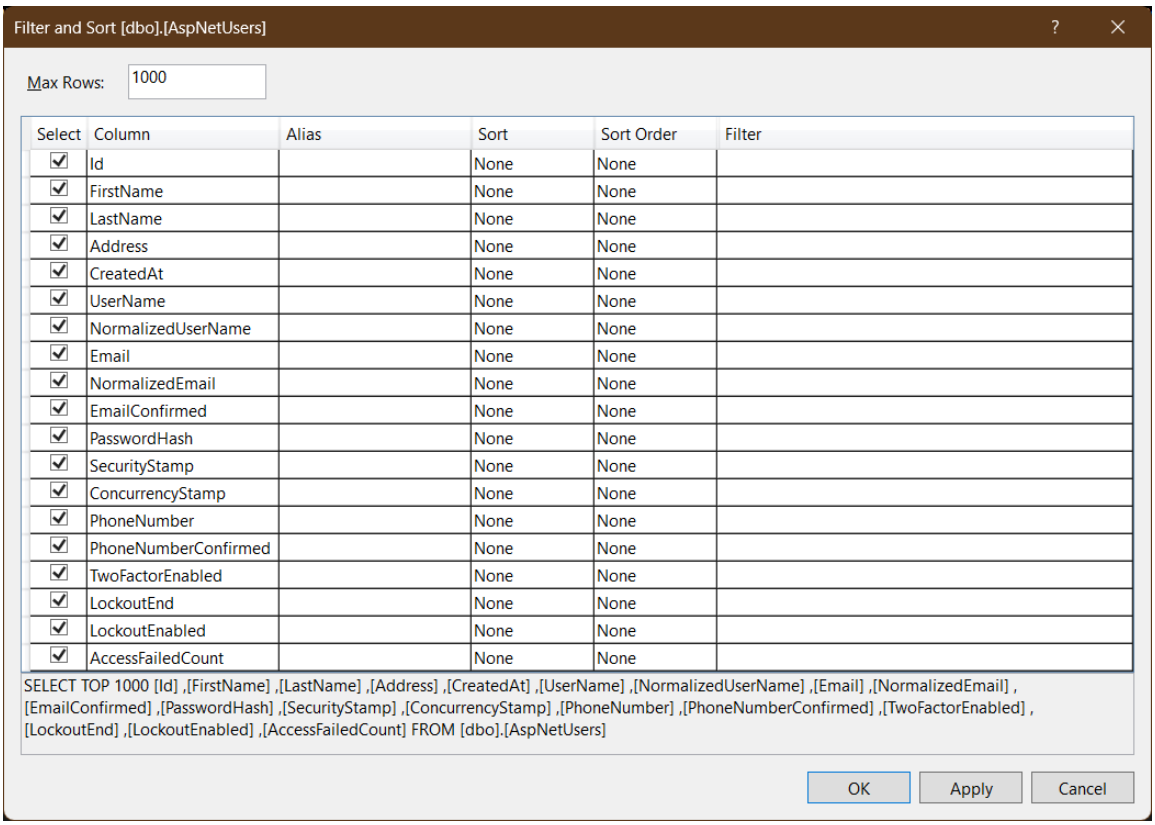

Рисунок 3.20 – Види інформації бази даних

Для реалізації Role-Based Authorization і Role-Based Access Control було створено 3 сторінки типу Razor Pages для користувача, адміністратора та клієнта (User.cshtml, User.cshtml.cs; Admin.cshtml, Admin.cshtml.cs; Client.cshtml, Client.cshtml.cs). Файли зовнішнього вигляду було змінено, так щоб відображало вітальне повідомлення, а також було додано різні типу доступу до сторінок: [AllowAnonymous] – дозвіл для всіх; [Authorize(Roles = "client")] – дозволено тільки клієнтам, [Authorize] – дозволено авторизованим користувачам, [Authorize(Roles = "admin")] – дозволено тільки адміністраторам як показано на рис. 3.21.

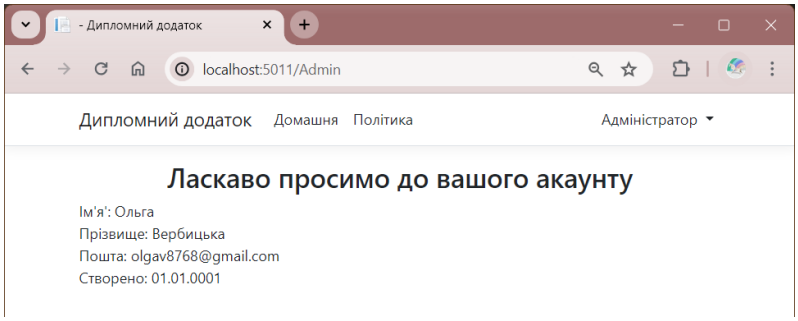

Рисунок 3.21 – Доступ до ресурсу Admin

Для забезпечення авторизації на основі ролей було змінено налаштування Program.cs, додаючи функцію AddRoles<IdentityRole>.

Також для реалізації Role-Based Authorization і Role-Based Access Control було оновлено таблицю AspNetRoles за допомогою налаштування класу ApplicationDbContext. Було ініціалізовано три ролі: seller, admin, client. Оновлена таблиця показана на рис. 3.22.

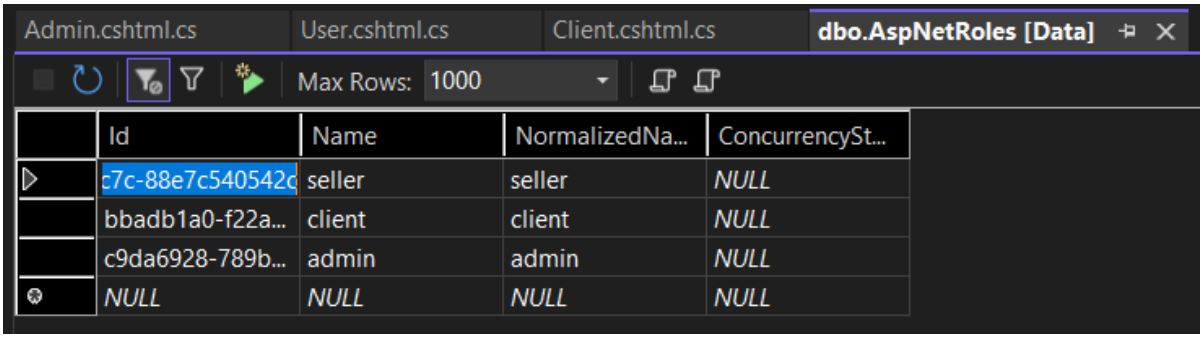

Рисунок 3.22 – Таблиця AspNetRoles

Хешування застосовано для зберігання паролів, що можна побачити в базі даних на рис. 3.23. PasswordHash – хеш пароля користувачу, SecurityStamp – сіль пароля. Це забезпечує безпеку пароля, оскільки пароль користувачів не зберігається у відкритому вигляді, а лише його хешований варіант, який є важким для дешифрування. При автентифікації користувача в системі, введений пароль перевіряється на відповідність до збереженого хешу. SecurityStamp – це додатковий захисний елемент, який використовується для підвищення безпеки автентифікації користувачів. Сіль – це випадкове значення, яке зазвичай генерується під час створення облікового запису або при зміні пароля.

Id	FirstName	CreatedAt	UserName	PasswordHash	SecurityStamp
10378bd0-76fa...	Вол	01.01.0001 0:00:...	vovfvvr8768@g...	AQAAAAIAAYagAAAAEOvYJqL7aQns8p9N7cnu/LzXMfvAX/VvlzPqAFQJ32yrNaNN92S4gMfRbac2BKozng==	WYTHKQYJ7ANNT4O2RFL3MJEBU4H3GRPF
46dc3b75-864e...	Bill	01.01.0001 0:00:...	olgavfvdvfgv87...	AQAAAAIAAYagAAAAENIS3U08e8cJ/LNoxbc7i3736z08vquj33ymBDkzA0BEOvBEvswL65+86wo0dkSTxw==	BD5SNBAMFKSIVKUJ3334C2IXOJR6QXPY
9a1cbf30-9503-...	Ольга	01.01.0001 0:00:...	olgav8768@gm...	AQAAAAIAAYagAAAAEKnfg/rMJ6i8+ERSc+1fTdo+rlNOdQLAHpxOeTACnJLfp+82h41NpXruARJMmToe/A==	XAC4L7536YUMMJGGQG55UJKPG5PDIG4A
9e07a3b2-0d11...	Вол	01.01.0001 0:00:...	vov8768@gmai...	AQAAAAIAAYagAAAAEG5kLjTUN5ahYgu/DFOgojPbHs8hxN6hJ3/a6frzR+N10kGX84rSUjhY57Cbm4fPcw==	J7D4WFH6KKAG25UX6G7WEYJYJ744RMS4
a2300fed-f9b6-...	Billfre	01.01.0001 0:00:...	olgadecvfvdvfg...	AQAAAAIAAYagAAAAEKmAFM/wf46VVXqmtNaS7PNIECoJW1QU5H2zQCSTgI7clIW5467xqq/QoA3CbLVIYQ==	ZMK7HOK7CGGQMP6AWAR43UQFABDPQP7E
NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 3.23 – База даних з хешами паролів

Реалізація методів захисту від SQL-ін’єкцій, XSS-атак і CSRF-атак знаходиться у файлі Startup.cs, а саме: параметризовані запити та перевірка вхідних даних; перевірка на наявність шкідливих символів; валідація вхідних даних та кодування виводу, очищення від HTML-тегів та кодування спеціальних символів; генерація анти-CSRF-токена.

3.3 Рекомендації щодо захисту вебдодатків

Після проведення практичної реалізації удосконалених методів системи захисту вебдодатків можна виділити наступні загальні рекомендації для покращення захисту в технології .NET:

- комплексний підхід: потрібно обов’язково використовувати комбінацію різних методів захисту, таких як автентифікація, авторизація, шифрування, контроль доступу та захист від атак, для забезпечення повноцінного захисту додатка;
- регулярні оновлення: проводити оновлення програмного забезпечення, бібліотек, фреймворків та компонентів, які використовуються у додатку, з метою виправлення виявлених вразливостей, підвищення безпеки та стабільності системи;
- системи моніторингу та аудиту: використовувати системи моніторингу та аудиту для виявлення та аналізу аномальної активності користувачів, вразливостей в системі та потенційних загроз безпеці;
- політики паролів: потрібно ставити критерії на складність паролів, а також використовувати двофакторну автентифікацію для додаткового забезпечення.

Висновки за розділом 3

У третьому розділі дипломної роботи було детально розглянуто та реалізовано методи захисту для додатків, розроблених з використанням технології .NET. Зокрема, у розділі було розглянуто процес створення додатка ASP.NET Core Web App,

включаючи налаштування середовища розробки та встановлення необхідних пакетів для забезпечення безпеки.

Реалізація методів захисту містила в собі використання різноманітних криптографічних протоколів, механізмів контролю доступу та автентифікації, що спрямовані на забезпечення конфіденційності, цілісності та доступності даних у додатку. Було детально описано та реалізовано заходи безпеки, такі як обробка введених даних, захист від SQL-ін’єкцій та перехоплення сесій, що забезпечило високий рівень безпеки програмного продукту.

Рекомендації щодо захисту вебдодатків в технології .NET, надані у даному розділі, містять в собі практичні поради та стратегії для забезпечення безпеки програмних засобів.

Тестування функціональних можливостей реалізованих методів безпеки показує, що заходи, запропоновані та реалізовані у цьому розділі, виконують необхідні умови для забезпечення безпеки вебдодатків, розроблених з використанням технології .NET.

ВИСНОВКИ

Під час написання дипломної роботи було детально досліджено технології .NET, включаючи її визначення, історію виникнення, основні компоненти, використання, а також переваги та недоліки. Аналізуючи методи захисту в технології .NET, були проаналізовані та описані такі методи безпеки, як автентифікація та авторизація (з використанням різних підходів, таких як Windows Authentication, Forms Authentication, Claims-based Authentication, OAuth 2.0, Role-Based Authorization, OpenID Connect), шифрування (включаючи симетричне та асиметричне шифрування, а також хешування), контроль доступу (з використанням Role-Based Access Control, Token-Based Authentication і безпечного протоколу WebSocket), захист від SQL-ін’єкцій (за допомогою параметризованих запитів, і перевірки вхідних даних), захист від XSS-атак (використовуючи валідацію вхідних даних і кодування виводу, HttpOnly cookies, Content-Security-Policy) та захист від CSRF-атак (впроваджуючи анти-CSRF-токенів та SameSite cookies).

Проаналізувавши і порівнявши методи захисту, було виявлено, що кожен з них має свої переваги та обмеження, тому для ефективного захисту додатків, розроблених з використанням технології .NET, необхідно впроваджувати комплексний підхід, включаючи різноманітні методи автентифікації, шифрування та контролю доступу. Наприклад, використання різних методів автентифікації та авторизації дозволяє гнучко налаштувати систему безпеки під конкретні потреби проекту, а застосування заходів захисту від SQL-ін’єкцій, XSS-атак та CSRF-атак допомагає запобігти різноманітним вразливостям програмного забезпечення. Крім того, важливо враховувати специфіку конкретного додатка та його потенційні загрози для вибору найбільш ефективних методів захисту.

В останньому розділі дипломної роботи була проведена практична реалізація деяких удосконалених методів захисту шляхом створення ASP.NET MVC вебдодатку, що дозволило практично застосувати та перевірити ефективність використаних методів.

Отже, було досягнуто основної мети дипломної роботи – досліджено методи безпеки в технології .NET і розробка комплексної системи захисту для вебдодатка. Результати досліджень і розробки дозволяють зробити висновок про необхідність комплексного та індивідуалізованого підходу до захисту технології .NET від різноманітних загроз і визначити напрямки подальших досліджень у цій області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Good and the Bad of .NET Framework Programming [Електронний ресурс] – Режим доступу: https://www.altexsoft.com/blog/the-good-and-the-bad-of-net-framework-programming/

2. C#. Концепція та синтаксис. Навч. Посібник. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2006ю – 136 с. [Електронний ресурс] – Режим доступу: https://ami.lnu.edu.ua/wp-content/uploads/2017/05/C_sharp.pdf

3. Л.С. Глоба. «Розробка інформаційних ресурсів та систем». – 2013. [Електронний ресурс] – Режим доступу: https://duikt.edu.ua/ua/lib/1/category/96/view/1690

4. Базові поняття .Net. Конспект лекцій. / укладачі: В. В. Булатецький, Л. В. Булатецька; ВНУ ім. Лесі Українки. – Електронні текстові данні (1 файл: 0,99 Мбайт). Луцьк : ВНУ ім. Лесі Українки, 2023. 37 с. [Електронний ресурс] – Режим доступу: https://evnuir.vnu.edu.ua/bitstream/123456789/21666/1/.Net.pdf

5. Що таке .NET і чим займаються .NET-розробники? [Електронний ресурс] – Режим доступу: https://training.epam.ua/ua/blog/301

6. A Brief History of .NET Framework [Електронний ресурс] – Режим доступу: https://softteco.com/blog/a-brief-history-of-net-framework

7. A Brief History of .NET (dotnet) [Електронний ресурс] – Режим доступу: https://medium.com/calvin-codes/a-brief-history-of-net-ec4c14adf441

8. .NET Core vs .NET Framework vs .NET Standard: A Guided Tour [Електронний ресурс] – Режим доступу: https://auth0.com/blog/navigating-dotnet-maze/

9. The History of .NET [Електронний ресурс] – Режим доступу: https://www.omnitech-inc.com/blog/the-history-of-net/

10. What’s new in .NET Framework [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/framework/whats-new/

11. What Is .NET Used For? [Електронний ресурс] – Режим доступу: https://existek.com/blog/what-is-net-used-for/

12. Xamarin Support Policy [Електронний ресурс] – Режим доступу: https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin

13. Introduction to .NET [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/core/introduction#components

14. Introduction to .NET Framework [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/introduction-to-net-framework/

15. Introduction to .NET Framework [Електронний ресурс] – Режим доступу: https://www.javatpoint.com/vb-net-dot-net-framework-introduction

16. Базові поняття технології .NET Framework [Електронний ресурс] – Режим доступу: https://www.bestprog.net/uk/2016/12/20/%d0%b1%d0%b0%d0%b7%d0%be%d0%b2%d1%96-%d0%bf%d0%be%d0%bd%d1%8f%d1%82%d1%82%d1%8f-%d1%82%d0%b5%d1%85%d0%bd%d0%be%d0%bb%d0%be%d0%b3%d1%96%d1%97-net-framework/#q01

17. Cross platform, open source .NET framework Електронний ресурс] – Режим доступу: https://www.mono-project.com/

18. .NET implementations [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/fundamentals/implementations

19. Documentation [Електронний ресурс] – Режим доступу: https://www.mono-project.com/docs/

20. .NET Core, .NET Framework, Xamarin – The “WHAT and WHEN to use it” [Електронний ресурс] – Режим доступу: https://devblogs.microsoft.com/cesardelatorre/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/

21. What Is .NET Core? [Електронний ресурс] – Режим доступу: https://www.c-sharpcorner.com/article/what-is-dot-net-core/

22. What is .NET, and why should you choose it? [Електронний ресурс] – Режим доступу: https://devblogs.microsoft.com/dotnet/why-dotnet/

23. Releases and support for .NET [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/core/releases-and-support

24. Common Language Runtime (CLR) overview [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/clr#clr-versions

25. What Makes .NET a Popular Choice Among Our Clients? [Електронний ресурс] – Режим доступу: https://inwedo.com/blog/using-dotnet-why-and-when/

26. Що таке ASP.NET і для чого використовується? [Електронний ресурс] – Режим доступу: https://hyperhost.ua/info/uk/shho-take-aspnet-i-dlya-cogo-vikoristovujetsya

27. Що таке .NET в IT: особливості і сфера застосування [Електронний ресурс] – Режим доступу: https://shalabodin.com/shho-take-net-v-it-osoblyvosti-i-sfera-zastosuvannya/

28. What Is .NET? [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/net/

29. Що таке платформи моделей додатків .NET? [Електронний ресурс] – Режим доступу: https://lemon.school/blog/shho-take-platformy-modelej-dodatkiv-net

30. PROS AND CONS OF .NET SOFTWARE DEVELOPMENT [Електронний ресурс] – Режим доступу: https://leobit.com/blog/pros-and-cons-of-net-software-development/

31. What is Authentication? [Електронний ресурс] – Режим доступу: https://auth0.com/intro-to-iam/what-is-authentication

32. What is Authorization? [Електронний ресурс] – Режим доступу: https://auth0.com/intro-to-iam/what-is-authorization

33. Configure Windows Authentication in ASP.NET Core [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/windowsauth?view=aspnetcore-8.0&tabs=visual-studio

34. Understanding Windows Authentication in Detail [Електронний ресурс] – Режим доступу: https://www.c-sharpcorner.com/UploadFile/84c85b/understanding-windows-authentication-in-detail/

35. Unleashing the Power of Forms Authentication: Step-by-Step Guide to Securely Implementing it in .NET! [Електронний ресурс] – Режим доступу:

https://medium.com/@shekhartarare/unleashing-the-power-of-forms-authentication-step-by-step-guide-to-securely-implementing-it-in-net-57122d9d078d

36. FormsAuthentication Class [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.web.security.formsauthentication?view=netframework-4.8.1

37. Form Authentication: ASP.NET Security Part 3 [Електронний ресурс] – Режим доступу: https://www.infosecinstitute.com/resources/application-security/form-authentication-asp-net-security-part-3/

38. Security: a Brief Review of Claims-Based Authentication [Електронний ресурс] – Режим доступу: https://www.baeldung.com/cs/security-claims-based-authentication

39. Active Directory and claims-based authentication [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/active-directory-claims-based-authentication?view=op-9-1

40. Securing Modern .NET Core Apps [Електронний ресурс] – Режим доступу: https://medium.com/code-factory-berlin/securing-modern-net-core-apps-bd18dafe24f8

41. What is OAuth 2.0? [Електронний ресурс] – Режим доступу: https://auth0.com/intro-to-iam/what-is-oauth-2

42. What is OpenID Connect [Електронний ресурс] – Режим доступу: https://openid.net/developers/how-connect-works/

43. Role-based authorization [Електронний ресурс] – Режим доступу: https://www.ibm.com/docs/en/ftmfm/4.0.4?topic=center-role-based-authorization

44. Role-Based Security [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/security/role-based-security

45. Role-based authorization in ASP.NET Core [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-8.0

46. What is encryption? [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/ssl/what-is-encryption/

47. Introduction to symmetric encryption [Електронний ресурс] – Режим доступу: https://forum.huawei.com/enterprise/en/introduction-to-symmetric-encryption/thread/712399236424941568-667213854934970368

48. Encrypting data [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/security/encrypting-data

49. RSA Class [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsa?view=net-8.0

50. Алгоритм шифрування RSA, види атак на нього. Реалізація мовою Python [Електронний ресурс] – Режим доступу: https://dou.ua/forums/topic/43026/

51. What Is Hashing? A Guide With Examples. [Електронний ресурс] – Режим доступу: https://builtin.com/articles/what-is-hashing

52. Exploring the ASP.NET Core Identity PasswordHasher [Електронний ресурс] – Режим доступу: https://andrewlock.net/exploring-the-asp-net-core-identity-passwordhasher/

53. What is access control? [Електронний ресурс] – Режим доступу: https://www.microsoft.com/en-us/security/business/security-101/what-is-access-control

54. What is Role-Based Access Control (RBAC)? Examples, Benefits, and More [Електронний ресурс] – Режим доступу: https://www.digitalguardian.com/blog/what-role-based-access-control-rbac-examples-benefits-and-more

55. The Power of Role-Based Access Control in Network Security [Електронний ресурс] – Режим доступу: https://www.portnox.com/blog/network-security/the-power-of-role-based-access-control-in-network-security/

56. What Is Token-Based Authentication? [Електронний ресурс] – Режим доступу: https://www.okta.com/identity-101/what-is-token-based-authentication/

57. What is token-based authentication? [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/access-management/token-based-authentication/

58. WebSockets support in ASP.NET Core [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/websockets?view=aspnetcore-8.0

59. Man in the middle (MITM) attack [Електронний ресурс] – Режим доступу: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

60. WebSocket Security: How to secure connections? [Електронний ресурс] – Режим доступу: https://apidog.com/blog/websocket-security/

61. What is SQL injection (SQLi)? [Електронний ресурс] – Режим доступу: https://portswigger.net/web-security/sql-injection

62. SQL Injection [Електронний ресурс] – Режим доступу: https://www.w3schools.com/sql/sql_injection.asp

63. How to prevent SQL injection [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/security/threats/how-to-prevent-sql-injection/

64. Using parameterized queries to avoid SQL injection [Електронний ресурс] – Режим доступу: https://www.sqlshack.com/using-parameterized-queries-to-avoid-sql-injection/

65. Parameterized Queries in SQL – A Guide [Електронний ресурс] – Режим доступу: https://www.dbvis.com/thetable/parameterized-queries-in-sql-a-guide/

66. Using Input Validation to Prevent SQL Injection Attacks [Електронний ресурс] – Режим доступу: https://www.softwaremovers.com/blog/using-input-validation-to-prevent-sql-injection-attacks

67. .NET SQL Injection. Guide: Examples and Prevention [Електронний ресурс] – Режим доступу: https://www.stackhawk.com/blog/net-sql-injection-guide-examples-and-prevention/

68. Cross Site Scripting (XSS) [Електронний ресурс] – Режим доступу: https://owasp.org/www-community/attacks/xss/

69. What is cross-site scripting (XSS)? [Електронний ресурс] – Режим доступу: https://portswigger.net/web-security/cross-site-scripting

70. How to Prevent Cross-Site Scripting (XSS) in JavaScript [Електронний ресурс] – Режим доступу: https://www.telerik.com/blogs/how-to-prevent-cross-site-scripting-xss-javascript

71. Prevent Cross-Site Scripting (XSS) in ASP.NET Core [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-8.0

72. What is the best way to prevent XSS attacks with HTTP-only cookies? [Електронний ресурс] – Режим доступу: https://www.linkedin.com/advice/0/what-best-way-prevent-xss-attacks-http-only-mdsme

73. How HttpOnly cookies help mitigate XSS attacks [Електронний ресурс] – Режим доступу: https://clerk.com/blog/how-httponly-cookies-help-mitigate-xss-attacks

74. Through the Digital Landscape: The Role of Content Security Policy (Why Is CSP So Important?) [Електронний ресурс] – Режим доступу: https://www.telerik.com/blogs/through-digital-landscape-role-content-security-policy-why-csp-important

75. What is Content Security Policy (CSP) and how does it help mitigate the risk of XSS attacks? [Електронний ресурс] – Режим доступу: https://eitca.org/cybersecurity/eitc-is-wasf-web-applications-security-fundamentals/cross-site-scripting/cross-site-scripting-defenses/examination-review-cross-site-scripting-defenses/what-is-content-security-policy-csp-and-how-does-it-help-mitigate-the-risk-of-xss-attacks/

76. Cross Site Request Forgery (CSRF) [Електронний ресурс] – Режим доступу: https://owasp.org/www-community/attacks/csrf

77. Cross-Site Request Forgery Prevention Cheat Sheet [Електронний ресурс] – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

78. .NET CSRF Protection Guide: Examples and How to Enable [Електронний ресурс] – Режим доступу: https://www.stackhawk.com/blog/net-csrf-protection-guide-examples-and-how-to-enable/

79. What are anti-CSRF tokens and how do they contribute to web security? [Електронний ресурс] – Режим доступу: https://eitca.org/cybersecurity/eitc-is-acss-advanced-computer-systems-security/network-security/web-security-model/examination-

review-web-security-model/what-are-anti-csrf-tokens-and-how-do-they-contribute-to-web-security/

80. Preventing Cross-Site Request Forgery (CSRF) Attacks in ASP.NET MVC Application [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/web-api/overview/security/preventing-cross-site-request-forgery-csrf-attacks

ДОДАТОК А

Копія наукової публікації

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

Методи безпеки в технології .NET

Сергій Бучик[1], Аліна Клещенко[2]

1. Кафедра кібербезпеки та захисту інформації, Київський національний Університет імені Тараса Шевченка, УКРАЇНА, м. Київ, вул. Б. Гаврилишина, 24, E-mail: buchyk@knu.ua

2. Кафедра кібербезпеки та захисту інформації, Київський національний Університет імені Тараса Шевченка, УКРАЇНА, м. Київ, вул. Б. Гаврилишина, 24, E-mail: kleshchenkoalina@knu.ua

In the dynamic realm of software development, .NET emerges as a robust and versatile framework, empowering developers to craft scalable and secure applications. However, safeguarding these applications against ever-evolving cyber threats necessitates a comprehensive understanding of the security measures embedded within the .NET framework. This article delves into the intricate spectrum of security methods provisioned by .NET, meticulously dissecting their strengths and limitations to equip developers with the knowledge to fortify their applications. By mastering and implementing these security methods, developers can transform their .NET applications into fortresses against a wide range of cyber threats, ensuring the integrity and protection of valuable data and resources.

Ключові слова – технологія .NET, автентифікація, авторизація, шифрування, хешування, XSS, CSRF.

Вступ

У сучасному світі, де все більше інформації зберігається та обробляється в цифровому середовищі, питання безпеки є ключовим для будь-якої IT-системи. Технологія .NET, що широко використовується для розробки вебдодатків та сервісів,

не є винятком. Захист даних, запобігання несанкціонованому доступу та кібератак стають все більш важливими завданнями як для приватних осіб, так і для великих організацій. У світлі постійного росту кількості кіберзагроз і зловмисних атак, ретельне вивчення та використання методів безпеки в .NET стає надзвичайно важливим завданням для розробників та адміністраторів систем.

Методи безпеки в технології .NET

.NET – платформа розробки програмного забезпечення від Microsoft, що дозволяє створювати додатки для різних пристроїв і ОС. Вона включає мови програмування, середовище виконання та класи бібліотек, сприяючи розробникам у забезпеченні функціональності своїх програм. Безпека .NET відіграє важливу роль, забезпечуючи захист від різних видів атак, через різноманітні підходи до автентифікації, авторизації, шифрування та контролю доступу, а також захист від різних видів атак, таких як XSS та CSRF [1, 2].

В .NET для автентифікації та авторизації використовуються різні методи, які базуються на комбінації проміжного програмного забезпечення, атрибутів і конфігураційних параметрів. Це надає розробникам можливість вибору підходящого механізму безпеки для їхніх додатків. Forms Authentication використовує HTML-форми для збору імен користувачів та паролів, а потім перевіряє їх з базою даних користувачів. Claims-based Authentication використовує маркери (token) для автентифікації користувачів та містить інформацію про їхні претензії (claims), такі як ім’я та роль. Role-Based Authorization використовує ролі для визначення прав доступу користувачів до ресурсів, призначаючи їм набори дозволів [2, 3].

Шифрування важливо для безпеки вебдодатків, захищаючи конфіденційні дані. У .NET є різні алгоритми для цього. По-перше, симетричне шифрування використовує один і той же секретний ключ для обох дій, що робить його швидким і ефективним для великих обсягів даних. По-друге, асиметричне шифрування використовує пару ключів, з яких один публічний (для шифрування), а інший – приватний (для дешифрування). Останнє – це хешування, яке перетворює дані на фіксований розмір, з неможливістю відновлення даних, що корисно для перевірки цілісності даних [4, 5].

Контроль доступу є ключовою складовою безпеки будь-якого вебдодатку, гарантуючи, що лише авторизовані користувачі мають доступ до певних функцій та даних. У .NET існують різні механізми контролю доступу, які надають розробникам гнучкість у налаштуванні системи безпеки. Наприклад, ASP.NET Membership надає базову систему керування користувачами та автентифікації, дозволяючи створювати облікові записи, зберігати дані та керувати дозволами. Що до протокол WebSocket, він дозволяє встановлювати двосторонні з’єднання з сервером, ідеально підходячи для реалізації чатів, онлайн-ігор та інших динамічних вебзастосунків [6-8].

XSS-атаки (Cross-Site Scripting) становлять серйозну загрозу для безпеки вебдодатків, дозволяючи зловмисникам впроваджувати шкідливий JavaScript-код на сторінки, що переглядаються користувачами. У .NET існують різні методи боротьби з цими атаками. Валідація вхідних даних допомагає вчасно виявляти та блокувати потенційно небезпечні запити, уникати впровадження шкідливого коду. Крім того, використання Content-Security-Policy (CSP) дозволяє контролювати джерела завантаження ресурсів на сторінці, запобігаючи завантаженню шкідливого JavaScript-коду з ненадійних джерел [9].

CSRF-атаки (Cross-Site Request Forgery) представляють серйозну загрозу для безпеки вебдодатків, дозволяючи зловмисникам змусити користувачів виконати небажані дії на вебсайті, до якого вони авторизовані. У .NET доступні різні методи захисту від CSRF-атак. Один з них – це використання анти-CSRF-токенів, що полягає у генеруванні унікального токена для кожного користувача та додаванні його до всіх форм та запитів. Токен перевіряється на сервері для підтвердження автентичності запиту. Крім того, можна використовувати різні HTTP-заголовки, такі як X-CSRF-Token та X-Forwarded-For, які надають додаткову інформацію про запит і допомагають у його перевірці [10].

В таблиці 1 представлені переваги та недоліки проаналізованих методів безпеки технології .NET.

ТАБЛИЦЯ 1

Переваги та недоліки методів безпеки технології .NET

Метод захисту	Переваги	Недоліки
1. Автентифікація та авторизація		
Forms Authentication	Простота реалізації та гнучкість в налаштуванні сторінок	Вразливість до атак типу брутфорс та фішингу
Claims-based Authentication	Гнучкий контроль доступу, додаткові атрибути	Потребує додаткової розробки
Role-Based Authorization	Гнучкість, масштабованість, керування дозволами	Складність реалізації для складних застосунків
2. Шифрування		
Симетричне шифрування	Швидкість, ефективність	Потреба у безпечному зберіганні спільного ключа
Асиметричне шифрування	Вища безпека через пару ключів	Зменшення швидкості обміну даними
Хешування	Гарантує безпеку даних	Вразливість до атак перебору
3. Контроль доступу		
ASP.NET Membership	Гнучкість у налаштуванні рівнів доступу, інтеграція	Складність реалізації та необхідність управління базою даних
Безпечний протокол WebSocket	Низька затримка та ефективність для потокових даних	Складність впровадження та обмежену сумісність
4. Захист від XSS-атак		
Input Validation	Перевірка HTML або JavaScript коду у вхідних	Складність визначення небезпечних символів

Content-Security-Policy (CSP)	Гнучке налаштування дозволених джерел	Складність реалізації та порушення роботи сторінки
5. Захист від CSRF-атак		
Анти-CSRF-токен	Ефективний проти CSRF-атак, простий у реалізації	Ускладнює форми та запити, потребує тестування
HTTP-заголовки	Не потребує змін форм та запитів, може використовуватися разом з іншими методами	Не всі браузери та сервери підтримують ці заголовки, може бути складніше реалізувати, ніж токени CSRF

Висновок

Розглянуті методи безпеки в технології .NET є ключовими складовими для забезпечення надійності та безпеки програмного забезпечення. В ході дослідження були проаналізовані та описані такі методи безпеки, як автентифікація та авторизація, шифрування, контроль доступу, захист від XSS-атак та захист від CSRF-атак.

Кожен з цих методів має свої переваги та недоліки, але вони разом утворюють комплексну систему захисту, яка забезпечує найвищий рівень безпеки програмного забезпечення на платформі .NET. Наприклад, використання різних методів автентифікації та авторизації дозволяє гнучко налаштувати систему безпеки під конкретні потреби проекту, а застосування заходів захисту від XSS-атак та CSRF-атак допомагає запобігти різноманітним вразливостям програмного забезпечення.

Отже, висновки дослідження підтверджують, що впровадження та використання різноманітних методів безпеки в технології .NET є важливою складовою для забезпечення надійності та безпеки програмного забезпечення. Ретельний аналіз та вибір оптимальних методів забезпечення безпеки допоможе забезпечити найвищий рівень захисту даних та ресурсів на платформі .NET.

Література

[1] altexsoft.com [Електронний ресурс]. – https://www.altexsoft.com/blog/the-good-and-the-bad-of-net-framework-programming/

[2] support.microsoft.com [Електронний ресурс]. – https://support.microsoft.com/en-us/topic/asp-net-security-overview-7c8562d3-7bea-306c-4c78-98dd6c6993b3

[3] medium.com [Електронний ресурс]. – https://medium.com/@kerimkkara/authentication-and-authorization-in-asp-net-core-a-comprehensive-guide-dfb8fb806ac7

[4] learn.microsoft.com [Електронний ресурс]. – https://learn.microsoft.com/en-us/dotnet/standard/security/cryptographic-services

[5] learn.microsoft.com [Електронний ресурс]. – https://learn.microsoft.com/en-us/dotnet/standard/security/encrypting-data

[6] learn.microsoft.com [Електронний ресурс]. – https://learn.microsoft.com/en-us/aspnet/web-forms/overview/moving-to-aspnet-20/membership

[7] medium.com [Електронний ресурс]. – https://medium.com/@microclip.lakeesha/net-core-6-token-based-authentication-and-middleware-43a5fc86089a

[8] learn.microsoft.com [Електронний ресурс]. – https://learn.microsoft.com/en-us/aspnet/core/fundamentals/websockets?view=aspnetcore-8.0

[9] learn.microsoft.com [Електронний ресурс]. – https://learn.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-8.0

[10] learn.microsoft.com [Електронний ресурс]. – https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-8.0

ДОДАТОК Б

Код програми

Код Index.cshtml

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Домашня сторінка";
}
<head>
    <title>Домашня</title>
</head>
<div class="text-center">
    <h1 class="display-4">Головна сторінка</h1>
    <h2 class="display-4">дипломного додатку</h2>
    <p>
    Додаток дипломної роботи є прикладом комплексної системи захисту
    </p>
</div>
<br />
<a href="/User"><span style="color: black; font-style: italic;">Про користувача</span></a>
<br />
<a href="/Admin"><span style="color: black; font-style: italic;">Ви адміністратор?</span></a>
<br />
<a href="/Client"><span style="color: black; font-style: italic;">Ви клієнт?</span></a>
```

Код Index.cshtml.cs

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

namespace WebApp1.Pages
{
    [AllowAnonymous] // Дозволяє доступ незалежно від статусу автентифікації

    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }
    }
}
```

Код _LoginPartial.cshtml

```
@using Microsoft.AspNetCore.Identity
@using WebApp1.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">
                        Адміністратор
                    </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("seller"))
            {
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle text-dark" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="true">
                        Користувач
                    </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("client"))
        {
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle text-dark" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="true">
                    Привілійований користувач
                </a>
                <ul class="dropdown-menu">
                    <li><a class="dropdown-item" asp-area="Identity" asp-page="/Account/Manage/Index">Профіль</a></li>
                    <li><hr class="dropdown-divider"></li>
                    <li><a class="dropdown-item" asp-area="Identity" asp-page="/Account/Logout">Вийти</a></li>
                </ul>
            </li>
        }
    }
    else
    {
        <li class="nav-item">
            <a class="nav-link text-dark" id="register" asp-area="Identity" asp-page="/Account/Register">Реєстрація</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" id="login" asp-area="Identity" asp-page="/Account/Login">Вхід</a>
        </li>
```

```
        }
    </ul>
```

Код _Loyout.cshtnl

```
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>@ViewData["Title"] – Дипломний додаток</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="~/WebApp1.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">Дипломний додаток</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">Домашня</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-dark" asp-area="" asp-page="/Privacy">Політика</a>
                    </li>
                </ul>
                <partial name="_LoginPartial"/>
                @*  <p class="nav navbar-text">Hello, @User.Identity?.Name!</p> *@
            </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 – WebApp1 – <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 Privacy.cshtml

```
@page
@model PrivacyModel
@{
    ViewData["Title"] = "Політика безпеки";
}
<h1>@ViewData["Title"]</h1>


<p>
    На цій сторінці буде розглянуто комплексну систему захисту, яку було створено у додатку ASP.NET Core для забезпечення безпеки та захисту ваших даних.
</p>
```

Код Register.cshtml

```
@page
@using Microsoft.AspNetCore.Identity
@model RegisterModel
@{
    ViewData["Title"] = "Реєстрація";
}
<div class="row">
    <div class="col-lq-6 mx-auto rounded border p-3">
        <h2 class="text-center mb-3">Реєстрація</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">Ім’я’</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">Прізвище</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">Пошта</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">Пароль</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">Повторити пароль</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-dark btn-primary">Реєстрація</button>
                </div>
                <div class="col-sm-4 d-grid">
                    <a class="btn btn-outline-dark" href="/" role="button">Закрити</a>
                </div>
            </div>
        </form>
    </div>
</div>
```

Код Register.cshtml.cs

```
#nullable disable
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 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;
using WebApp1.Models;
namespace WebApp1.Areas.Identity.Pages.Account
{
    [AllowAnonymous]
    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);
                }
            }
            // If we got this far, something failed, redisplay form
            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 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 SendSettings.cs

```
namespace WebApp1.Services
{
    public class SendSettings
    {
        public string ApiKey { get; set; }
```

```
        public string FromEmail { get; set; }
        public string EmailName { get; set; }

    }
}
```

Код EmailSender.cs

```
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Options;
using SendGrid;
using SendGrid.Helpers.Mail;
using WebApp1.Services;

namespace RazorPage.AspNetIdentity.Services
{
    public class EmailSender : IEmailSender
    {
        private readonly ISendGridClient _sendGridClient;
        private readonly SendSettings _sendSettings;
        public EmailSender(ISendGridClient sendGridClient,
        IOptions<SendSettings> sendGridSettings)
        {
            _sendGridClient = sendGridClient;
            _sendSettings = sendGridSettings.Value;
        }
        public async Task SendEmailAsync(string email, string subject, string
htmlMessage)
        {
            var msg = new SendGridMessage()
            {
```

```
                From = new EmailAddress(_sendSettings.FromEmail,
_sendSettings.EmailName),
                Subject = subject,
                HtmlContent = htmlMessage
            };
            msg.AddTo(email);
            await _sendGridClient.SendEmailAsync(msg);
        }
    }
}
```

Код Login.cshtml

```
@page
@model LoginModel

@{
    ViewData["Title"] = "Вхід";
}
<div class="row">
    <div class="col-md-6 mx-auto rounded border p-3">
        <section>
            <h2 class="text-center mb-3">Ввійдіть в систему</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">Пошта</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">Пароль</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-dark">Вхід</button>
                    </div>
                    <div class="col-sm-4 d-grid">
                        <a class="btn btn-outline-dark" href="/" role="button">Закрити</a>
                    </div>
                </div>
                <div>
                    <a class="btn btn-link" id="forgot-password" asp-
page="./ForgotPassword">Забули пароль?</a>
                </div>
            </form>
        </section>
    </div>
```

```
</div>
```

Код Login.cshtml.cs

```
#nullable disable
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 WebApp1.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 WebApp1.Areas.Identity.Pages.Account
{
    [AllowAnonymous]
    public class LoginModel : PageModel
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ILogger<LoginModel> _logger;
```

```
        public LoginModel(SignInManager<ApplicationUser> signInManager,
ILogger<LoginModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }
        [BindProperty]
        public InputModel Input { get; set; }
        public IList<AuthenticationScheme> ExternalLogins { get; set; }
        public string ReturnUrl { get; set; }
        [TempData]
        public string ErrorMessage { get; set; }
        public class InputModel
        {
            [Required]
            [EmailAddress]
            [Display(Name = "Email")]
            public string Email { get; set; }
            [Required]
            [DataType(DataType.Password)]
            public string Password { get; set; }
            [Display(Name = "Remember me?")]
            public bool RememberMe { get; set; }
        }
        public async Task OnGetAsync(string returnUrl = null)
        {
            if (!string.IsNullOrEmpty(ErrorMessage))
            {
                ModelState.AddModelError(string.Empty, ErrorMessage);
            }
```

```
            returnUrl ??= Url.Content("~/");
            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            ReturnUrl = returnUrl;
        }
    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            returnUrl ??= Url.Content("~/");

            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();

            if (ModelState.IsValid)
            {
                // This doesn't count login failures towards account lockout
                // To enable password failures to trigger account lockout, set lockoutOnFailure: true
                var result = await _signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User logged in.");
                    return LocalRedirect(returnUrl);
                }
                if (result.RequiresTwoFactor)
```

```
                {
                    return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl,
RememberMe = Input.RememberMe });
                }
                if (result.IsLockedOut)
                {
                    _logger.LogWarning("User account locked out.");
                    return RedirectToPage("./Lockout");
                }
                else
                {
                    ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                    return Page();
                }
            }
            // If we got this far, something failed, redisplay form
            return Page();
        }
    }
}
```

Код Program.cs

```
using Microsoft.AspNetCore.Authentication.Negotiate;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using WebApp1.Models;
using WebApp1.Services;
using SendGrid.Extensions.DependencyInjection;
using Microsoft.AspNetCore.Identity.UI.Services;
```

```
using RazorPage.AspNetIdentity.Services;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAuthentication(NegotiateDefaults.AuthenticationScheme)
  .AddNegotiate();
builder.Services.AddAuthorization(options =>
{
  // By default, all incoming requests will be authorized according to the default policy.
  options.FallbackPolicy = options.DefaultPolicy;
});
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<SendSettings>(builder.Configuration.GetSection("SendGridSettings"));
builder.Services.AddSendGrid(options => {
  options.ApiKey = builder.Configuration.GetSection("SendGridSettings")
  .GetValue<string>("ApiKey");
});
```

```
builder.Services.AddScoped<IEmailSender, EmailSender>();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapRazorPages();
app.Run();
```

Код User.cshtml

```
@page
@model WebApp1.Pages.UserModel
@{
}
<h2 class="text-center">Ласкаво просимо Адміністратор</h2>
```

Код User.cshtml.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using WebApp1.Models;
```

```
namespace WebApp1.Pages
{
    [Authorize]

    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 Admin.cshtml

```
@page
@model WebApp1.Pages.UserModel
@{
}
<h2 class="text-center">Ласкаво просимо до вашого акаунту</h2>
<div>
    Ім’я’: @Model.appUser?.FirstName
```

```
</div>
<div>
   Прізвище: @Model.appUser?.LastName
</div>
<div>
   Пошта: @Model.appUser?.Email
</div>
<div>
   Створено: @Model.appUser?.CreatedAt.ToString("MM/dd/yyyy")
</div>
```

Код 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 Client.cshtml

```
@page
@model WebApp1.Pages.UserModel
```

```
@{
}
<h2 class="text-center">Ласкаво просимо до вашого клієнтського акаунту</h2>
```

Код Client.cshtml.cs

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

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

Код Startup.cs

```
using System;
using System.Text.RegularExpressions;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
namespace WebApp1
{
        public static class SecurityHelper
      {
```

```
            // Захист від SQL-ін’єкцій: параметризовані запити та перевірка вхідних даних
            public static IActionResult ProtectFromSqlInjection(string userInput)
            {
                // Перевірка на наявність шкідливих символів
                if (Regex.IsMatch(userInput, @"[-|;|,|\/|\(|\)|\[|\]|\}|\{|%|@|\*|!|\’]", RegexOptions.IgnoreCase))
                {
                    // Вивід помилки або інша обробка, якщо знайдено потенційну SQL-ін’єкцію
                    return new BadRequestObjectResult("SQL injection detected!");
                }
                // Виконання додаткових дій з валідним введеним значенням
                return new OkObjectResult("Input is safe.");
            }
            // Захист від XSS-атак: валідація вхідних даних та кодування виводу
            public static string ProtectFromXss(string userInput)
            {
                // Очищення від HTML-тегів та кодування спеціальних символів
                return System.Net.WebUtility.HtmlEncode(userInput);}
            // Захист від CSRF-атак: генерація анти-CSRF-токена
            public static string GenerateCsrfToken(HttpContext context)
            {
                var csrfToken = Guid.NewGuid().ToString();
                context.Session.SetString("CsrfToken", csrfToken);
                return csrfToken;
            }
        }
    }
```