Міністерство освіти і науки України

Одеський національний технологічний університет

Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Веб-система відображення стану мережевого обладнання*

(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Кульбаченка М.С.*

(прізвище та ініціали)

4 курсу *541(а)* групи

Керівники: *д.т.н., проф. Артеменко С.В.*

(посада, прізвище та ініціали)

ст. викл. Сіренко О.І.

(посада, прізвище та ініціали)

Консультанти: *Phd, ст.викл. Богданов О.О.*

(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *05.06* 2024 р., протокол № *8*

Завідувач кафедри *комп. інженерії* _____ *Сергій АРТЕМЕНКО*

(назва кафедри) (підпис) (ім'я та ПРІЗВИЩЕ)

Одеса – 2024 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*

Кафедра *комп'ютерної інженерії*

Ступінь вищої освіти *бакалавр*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітня програма *Мережеві технології та інтернет речей*

ЗАТВЕРДЖУЮ

Зав. кафедри *комп'ютерної інженерії*

Сергій АРТЕМЕНКО

« *30* » *серпня* 2023 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Кульбаченка Максима Сергійовича

1. Тема роботи *Веб-система відображення стану мережевого обладнання*

Затверджена наказом університету від « *30* » *серпня* 2023р., наказ № *442-03*

2. Термін здачі здобувачем закінченої роботи *1 червня 2024 р.*

3. Вихідні дані роботи

1. Інтегроване середовище розробки Visual Studio. 2. Framework ASP.NET MVC. 3. Платформа контейнеризації застосунків Docker Desktop. 4. Система управління реляційними базами даних Microsoft SQL. 5. Текстовий редактор Microsoft Word. 6. Редактор презентацій Microsoft PowerPoint.

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Збір та аналіз інформації. 3. Проектування системи. 4. Розробка системи. 5. Загальні висновки. 6. Економічні розрахунки. 7. Охорона праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайд 1. Тема проекту. Слайд 2. Актуальність. Слайд 3. Об'єкт, предмет. Слайд 4. Задачі. Слайд 5. Аналоги. Слайд 6. Функціональні вимоги. Слайд 7. Архітектура. Слайд 8. Структурна схема. Слайд 9. Розробка. Слайд 10. Реалізація. Слайд 11. Економічні розрахунки. Слайд 12. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Phd, ст.викл. Богданов О.О.*		
Охорона праці	*д.т.н., проф. Артеменко С.В.*		
Нормоконтроль	*ст. викл. Сіренко О.І.*		

7. Дата видачі завдання *30.08.2023*

Керівники *Сергій АРТЕМЕНКО*

Олександр СІРЕНКО

Завдання прийняв до виконання *Максим КУЛЬБАЧЕНКО*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Дослідження предметної області*	*17.10.2023*	
2.	*Дослідження існуючих аналогів*	*23.11.2023*	
3.	*Розробка технічного завдання*	*26.01.2024*	
4.	*Проектування*	*29.02.2024*	
5.	*Розробка демонстраційної версії ПЗ*	*24.03.2024*	
6.	*Підготовка техніко-економічної частини*	*12.04.2024*	
7.	*Підготовка розділу охорони праці*	*27.04.2024*	
8.	*Оформлення пояснювальної записки*	*23.05.2024*	
9.	*Оформлення графічної частини та*	*30.05.2024*	
	лістингу		

Здобувач - дипломник _____ *Максим КУЛЬБАЧЕНКО*

Керівники роботи _____ *Сергій АРТЕМЕНКО*

_____ *Олександр СІРЕНКО*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач - дипломник *Максим КУЛЬБАЧЕНКО* _____

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці веб-системи *NetworkMonitor* для моніторингу стану мережевого обладнання. Метою роботи є створення гнучкої, масштабованої та безпечної системи для збору та обробки мережевих даних. Основні технічні характеристики включають використання мови програмування *C#* у фреймворку *ASP.NET Core*, *JavaScript*, *HTML*, *CSS* для фронтенду, та *Microsoft SQL Server* з *Entity Framework Core* для роботи з базою даних.

У першому розділі проведено аналіз сучасних методів і технологій моніторингу мереж, що дозволило обґрунтувати вибір підходу до розробки системи.

Другий розділ присвячений проектуванню системи *NetworkMonitor*, включаючи архітектуру, методи збирання даних через *SSH*, *SNMP* та *HTTP*, а також використання шифрування *HTTPS* для забезпечення безпеки.

У третьому розділі описано практичну реалізацію системи з використанням обраних технологій, що дозволило досягти високої продуктивності та інтерактивності.

Четвертий розділ містить техніко-економічне обґрунтування проекту, що підтвердило доцільність його розробки з точки зору інвестицій та ресурсів.

Рекомендації щодо впровадження системи включають подальшу інтеграцію механізмів аутентифікації та авторизації користувачів, а також розширення функціоналу для підтримки додаткових типів мережевих пристроїв.

Ключові слова: моніторинг мереж, веб-система, *ASP.NET Core*, безпека даних, техніко-економічне обґрунтування.

ABSTRACT

The qualification work is dedicated to the development of the NetworkMonitor web system for monitoring the state of network equipment. The aim of the work is to create a flexible, scalable, and secure system for collecting and processing network data. The main technical characteristics include the use of the C# programming language in the ASP.NET Core framework, JavaScript, HTML, CSS for the frontend, and Microsoft SQL Server with Entity Framework Core for database operations.

The first chapter provides an analysis of modern methods and technologies for network monitoring, justifying the chosen approach for system development.

The second chapter is dedicated to the design of the NetworkMonitor system, including architecture, data collection methods through SSH, SNMP, and HTTP, as well as the use of HTTPS encryption for security.

The third chapter describes the practical implementation of the system using the selected technologies, achieving high performance and interactivity.

The fourth chapter contains the technical and economic justification of the project, confirming the feasibility of its development in terms of investments and resources.

Recommendations for system implementation include further integration of user authentication and authorization mechanisms, as well as expanding functionality to support additional types of network devices.

***Keywords**: network monitoring, web system, ASP.NET Core, data security, technical and economic justification.*

ЗМІСТ

Стор.

ВСТУП

У нашому сучасному світі, де мережі стають серцевиною величезної кількості діяльностей, моніторинг та управління мережевим обладнанням стає вельми важливою задачею. Важливість забезпечення безперервної роботи мережі, особливо в умовах зростаючої кількості пристроїв та залежності бізнес-процесів від них, не може бути недооціненою.

Це завдання вимагає не лише постійного контролю за станом обладнання, а й швидкого реагування на будь-які відхилення або проблеми. Відсутність ефективної системи моніторингу може призвести до великих фінансових втрат, втрати довіри клієнтів та негативного впливу на репутацію компанії.

У цьому контексті розробка та впровадження веб-системи для відображення стану мережевого обладнання набуває особливого значення. Така система дозволяє зосередити в одному місці важливу інформацію про роботу мережі, представити її у зручному для аналізу вигляді та забезпечити оперативну реакцію на будь-які проблеми.

Зростання кількості та складності мережевих інфраструктур робить ручне відстеження стану обладнання практично неможливим. Веб-система відображення стану мережевого обладнання може значно спростити цю задачу, надаючи такі переваги:

1. Централізований доступ до інформації. Вся інформація про стан мережевого обладнання буде доступна в одному місці, що значно полегшить її моніторинг та аналіз.

2. Візуалізація даних. Система може використовувати різні візуальні інструменти, такі як графіки, діаграми та карти, для наочного представлення стану мережі.

3. Сповіщення про проблеми. Система може автоматично генерувати сповіщення про проблеми з мережевим обладнанням, що дозволить швидко їх вирішувати.

4. Історія даних. Система може зберігати записи про стан мережевого обладнання протягом певного періоду часу, що дозволить відстежувати динаміку змін та проводити аналіз.

У цій роботі об'єктом мого дослідження є моніторинг та управління станом мережевого обладнання. Предмет дослідження – розробка та впровадження веб-системи для відображення стану мережевого обладнання з метою забезпечення ефективного моніторингу, швидкого реагування на проблеми та оптимізації управління мережею.

Мета мого дослідження полягає в підвищенні ефективності моніторингу мережевого обладнання, зменшенні часу простою мережі, поліпшенні проактивного управління мережею та зниженні витрат на обслуговування мережі шляхом розробки, впровадження та оптимізації веб-системи відображення стану мережевого обладнання.

Особлива увага буде приділена аналізу вимог до системи, вибору відповідних інструментів та технологій, а також процесу розробки та тестування системи. Крім того, буде досліджено питання забезпечення безпеки та конфіденційності даних у веб-системах моніторингу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження архітектури систем моніторингу

Якщо в загальному описувати архітектуру систем моніторингу мережі то вона складається з взаємопов'язаних компонентів, що забезпечують збір, аналіз та візуалізацію даних про стан мережевої інфраструктури. Основними елементами є агенти моніторингу, розміщені на мережевих пристроях, центральний сервер управління, що агрегує та обробляє зібрані дані, та інтерфейс користувача для візуалізації та аналізу.

Агенти моніторингу збирають різноманітні метрики, такі як завантаженість каналів зв'язку, втрата пакетів, час відгуку тощо. Центральний сервер обробляє ці дані, виявляє аномалії, генерує сповіщення та формує звіти. Інтерфейс користувача дозволяє адміністраторам переглядати поточний стан мережі, аналізувати історичні дані та налаштовувати параметри моніторингу.

Веб-системи, як і веб-сайти – це основа сучасного Інтернету, що дозволяє користувачам отримувати доступ до інформації, спілкуватися та виконувати різні дії в онлайн-режимі. Розуміння принципів роботи вебсайтів та протоколів, які їх підтримують, є ключовим для розробки та використання веб-технологій.

Сайти на мережевому рівні працюють за принципом взаємодії між клієнтськими та серверними компонентами через інтернет-протоколи, такі як *HTTP* (*Hypertext Transfer Protocol*) або *HTTPS* (*Hypertext Transfer Protocol Secure*). Нижче описав декілька ключових принципів їхньої роботи.

Клієнт-серверна архітектура, коли сайти базуються на моделі клієнт-сервер, де клієнти (наприклад, веб-браузери) взаємодіють з серверами (комп'ютери, які обслуговують запити) для отримання веб-сторінок та ресурсів.

За допомогою протоколів передачі даних, таких як *HTTP* (або його безпечна версія *HTTPS*), відбувається взаємодія між клієнтом і сервером. Клієнти надсилають *HTTP*-запити на сервер, а сервери відповідають на ці запити,

надсилаючи *HTML* (*HyperText Markup Language*) -сторінки, *CSS* (*Cascading Style Sheets*) - стилі, *JavaScript*-скрипти та інші ресурси.

Сайти можуть містити як статичний, так і динамічний контент. Статичний контент, такий як *HTML*-файли та зображення, зберігається на сервері і відправляється клієнтам без змін. Динамічний контент генерується на сервері в реальному часі відповідно до запитів клієнтів, наприклад, шляхом виконання скриптів на мовах програмування, таких як *PHP, Python* або *JavaScript*.

Механізми сесій та куків використовуються для збереження стану між послідовними запитами. Сесії дозволяють зберігати дані на сервері для конкретного користувача протягом його візиту на сайт, тоді як
куки – це невеликі файли, які зберігаються на браузері клієнта і надсилаються з кожним запитом, щоб ідентифікувати та відстежувати користувачів.

Розподілені систем, такі як *CDN* (*Content Delivery Network* – мережа доставки контенту), можуть використовувати для підтримки великого обсягу запитів. А також можуть використовувати техніки балансування навантаження, щоб рівномірно розподілити трафік між різними серверами.

З метою захисту конфіденційності та цілісності даних використовуються різні заходи безпеки, такі як шифрування даних за допомогою *SSL* (*Secure Sockets Layer*) / *TLS* (*Transport Layer Security*) протоколів, аутентифікація користувачів та захист від атак, таких як перехоплення даних та введення шляхом використання механізмів, таких як брандмауери та системи виявлення вторгнень.

1.1.1 Мережеві протоколи

Сьогодні для роботи в мережі використовуються мережеві протоколи, оскільки майже всі комп’ютери в усьому світі підключені до Інтернету. Але багато користувачів не знають, що таке мережевий протокол, його призначення та особливості.

Мережевий протокол – це набір правил, що визначають, як пристрої в мережі спілкуються один з одним. Він встановлює стандарти для форматування

даних, їх передачі та інтерпретації. Мережеві протоколи є основою ефективної та надійної комунікації в комп'ютерних мережах.

Як розповідається в роботі [1], мережеві протоколи поділяються на сім рівнів відповідно до моделі *OSI* (*Open Systems Interconnection*). Кожен рівень має своє призначення та взаємодіє з іншими рівнями для забезпечення чіткої передачі даних у мережі. Виділяють наступні рівні протоколів:

- фізичний рівень;
- канальний рівень;
- мережевий рівень;
- транспортний рівень;
- сеансовий рівень;
- репрезентативний рівень;
- прикладний рівень.

Далі наведено опис тих рівнів на, яких працюють протоколи, які в свою чергу використовуються моніторинговими системами:

1. Канальний рівень (*PDU*):
   - цей рівень відповідає за надійну передачу даних через одне фізичне середовище;
   - він використовує адреси *MAC* для ідентифікації мережевих пристроїв.
2. Мережевий рівень (*IP, ICMP*):
   - цей рівень відповідає за логічну адресацію та маршрутизацію даних через мережу, яка може складатися з кількох фізичних сегментів;
   - він використовує I*P*-адреси для ідентифікації мережевих пристроїв.
3. Транспортний рівень (*TCP, UDP, SSL, TLS*):
   - цей рівень забезпечує надійну передачу даних між двома кінцевими точками в мережі;
   - він використовує порти для ідентифікації різних програм на одному пристрої.

4. Прикладний рівень (*HTTP, HTTPS, DNS, SMTP, SNMP, SSH, Telnet, IPMI, RMI, DCOM*):
   - цей рівень забезпечує мережеві служби для користувачів та прикладних програм, таких як електронна пошта, передача файлів, доступ до веб-сайтів тощо;
   - він використовує протоколи, які зрозумілі конкретним програмам.

Тепер розглянемо трохи детальніше протоколи, які приймають участь у роботі веб-системи. В роботі [2] зазначено:

1. *HTTP* (*Hypertext Transfer Protocol*) – це основний протокол, який використовується для передачі даних між веб-браузерами та веб-серверами. Він визначає правила форматування та передачі інформації, а також коди помилок, що гарантує чітку взаємодію. *HTTPS* – це безпечна версія *HTTP*, яка використовує шифрування для захисту конфіденційності даних під час передачі.

2. *TCP/IP* (*Transmission Control Protocol/Internet Protocol*) – протокол керування передачею даних/протокол Інтернету. Це сімейство протоколів, яке використовується в Інтернеті для передачі даних. *TCP* відповідає за надійну доставку даних, розбиваючи їх на пакети, нумеруючи їх і гарантуючи, що вони дійдуть у правильному порядку (рис. 1.1). *IP* відповідає за адресацію та маршрутизацію пакетів даних через мережу, використовуючи *IP*-адреси.

3. *UDP* (*User Datagram Protocol*) – використовується для моніторингу сервісів, що працюють на *UDP*-портах (*DNS*, *SNMP* тощо). Детальніше про цей протокол буде згадано в підрозділі1.2.1.

4. *DNS* (*Domain Name System*) – система доменних імен. *DNS* перетворює доменні імена (наприклад, "*https://example.com/*") на *IP*-адреси (наприклад, "192.0.2.44"), що робить доступ до веб-сайтів зручнішим для користувачів. Користувачі вводять доменне ім'я в браузер, а *DNS* направляє їх на правильний веб-сервер за допомогою *IP*-адреси.

5. *SMTP* (*Simple Mail Transfer Protocol*) – простий протокол передачі пошти. *SMTP* використовується для надсилання електронної пошти з веб-сайтів. Він переносить електронні листи від веб-сервера до сервера отримувача (*SMTP*-

сервера). *SMTP* використовує текстовий формат для представлення повідомлень електронної пошти.

6. *ICMP* (*Internet Control Message Protocol*) – це мережевий протокол, що використовується для передачі службових повідомлень про стан мережі та її елементів. Він допомагає діагностувати проблеми з підключенням, виявляти недоступні хости та маршрути, а також вимірювати час відгуку мережевих вузлів. ICMP не передає дані користувача, а лише службову інформацію,

7. *IPMI* (*Intelligent Platform Management Interface*) – це спеціалізований протокол, призначений для моніторингу та управління апаратними компонентами серверів та робочих станцій. Він надає доступ до сенсорів, журналів подій та функцій управління живленням, дозволяючи адміністраторам віддалено контролювати стан обладнання, отримувати повідомлення про апаратні помилки та виконувати дії з перезавантаження та вимкнення системи. IPMI незалежний від операційної системи, що забезпечує його роботу навіть у разі збоїв програмного забезпечення.

Рис. 1.1 – Рівні стеку протокола TCP/IP

Також розглянемо дві технології, які також необхідні для роботи моніторингових систем та, які використовують свої протоколи.

JMX (*Java Management Extensions*) – це технологія, що надає інтерфейс для управління та моніторингу додатків, написаних на мові *Java*. Вона дозволяє збирати статистику про використання ресурсів, контролювати стан додатків та виконувати різноманітні операції управління. *JMX* використовує протокол *RMI* (*Remote Method Invocation*) – це технологія *Java*, що дозволяє об'єктам однієї *Java Virtual Machine* (*JVM*) викликати методи об'єктів в іншій *JVM*, навіть якщо вони знаходяться на різних машинах.

WMI (*Windows Management Instrumentation*) – це набір технологій від *Microsoft*, призначений для управління та моніторингу систем на базі *Windows*. Він надає уніфікований інтерфейс для доступу до інформації про апаратне та програмне забезпечення, дозволяючи адміністраторам збирати дані про стан системи, конфігурацію, продуктивність та виконувати різноманітні адміністративні завдання. *WMI* використовує модель *COM* (*Component Object Model*) для представлення об'єктів управління та їх властивостей. І також використовує модель *DCOM* (*Distributed Component Object Model*) – це технологія Microsoft, що розширює *COM* (*Component Object Model*) для підтримки розподілених об'єктів. *DCOM* дозволяє об'єктам на різних комп'ютерах взаємодіяти, ніби вони знаходяться на одній машині.

1.1.2 Клієнт-серверна архітектура

В роботі [3] вказано, що клієнт-серверна архітектура – це модель побудови комп'ютерних мереж, де ролі розподілені між двома типами програм: клієнтами та серверами (рис. 1.2). Клієнти ініціюють запити на ресурси або послуги, а сервери їх надають.

Переваги клієнт-серверної архітектури:

- масштабованість – легко додавати нових клієнтів без значних змін у серверній інфраструктурі;
- централізоване управління – легко керувати даними на серверах;

- централізоване управління – легко керувати програмним забезпеченням;
- спеціалізація – клієнти та сервери можуть виконувати спеціалізовані функції, підвищуючи ефективність;
- відмовостійкість – можна використовувати резервні сервери для забезпечення безперебійної роботи.

Приклади використання клієнт-серверної архітектури:

1. Веб-сайти – клієнти (веб-браузери) надсилають запити до веб-серверів, які отримують та обробляють їх, а потім надсилають відповіді у вигляді веб-сторінок.
2. Електронна пошта – клієнти електронної пошти надсилають та отримують повідомлення через сервери електронної пошти.
3. Бази даних – клієнти баз даних надсилають запити до серверів баз даних, які отримують та обробляють їх, а потім надсилають відповіді у вигляді даних.

Важливо зазначити, що клієнт-серверна архітектура не є універсальною. Вона добре підходить для завдань, де потрібне чітке розділення функцій, масштабованість та централізоване управління.

Інші типи архітектур комп'ютерних мереж:

- однорангова архітектура – всі комп'ютери в мережі є рівноправними та можуть надавати та використовувати ресурси один одного;
- розподілена архітектура – завдання розподіляються між кількома комп'ютерами в мережі, що підвищує загальну продуктивність.

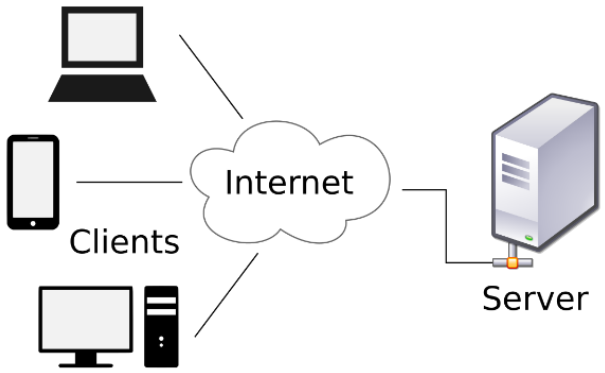


Рис. 1.2 – клієнт-серверна архітектура

1.2 Опис принципів отримання даних про стан мережевого обладнання

В роботі [4] розповідається, що отримання даних про стан мережевого обладнання є ключовим етапом у моніторингу та управлінні мережами. Цей процес включає в себе використання різноманітних методів та інструментів для збору, аналізу та візуалізації інформації про роботу мережевих пристроїв, таких як маршрутизатори, комутатори, фаєрволи, сервери тощо. За допомогою цих даних можна вчасно виявляти проблеми та вирішувати їх, забезпечуючи надійну та ефективну роботу мережі.

Один з основних принципів отримання даних про стан мережевого обладнання полягає в використанні протоколів моніторингу мережі. Серед найпоширеніших протоколів можна виділити *SNMP* (*Simple Network Management Protocol*), який дозволяє здійснювати збір інформації про стан пристроїв у реальному часі. За допомогою *SNMP* адміністратор мережі може отримувати дані про статус пристроїв, використання ресурсів, поточні навантаження та інші параметри.

Ще одним важливим методом є використання технологій моніторингу мережевого трафіку, таких як *NetFlow* або *sFlow*. Ці технології дозволяють отримувати дані про обсяги та характер трафіку, що проходить через мережеві пристрої. Аналіз цих даних дозволяє виявляти аномалії, визначати причини перевантажень та вчасно реагувати на них.

Окрім цього, можуть використовуватися інструменти командного рядка, такі як *Ping*, *Traceroute*, або *Telnet*, для діагностики конкретних проблем зі з'єднанням чи доступом до мережевих пристроїв. Вони дозволяють виконувати тестування зв'язку та визначати маршрути, якими проходить пакетна інформація в мережі.

Необхідно також зазначити, що для збору даних про стан мережевого обладнання можуть використовуватися спеціалізовані апаратні засоби, такі як сенсори або монітори, що підключаються безпосередньо до пристроїв. Ці засоби дозволяють отримувати додаткову інформацію про температуру, вологість.

В роботі [5] вказано, що *SNMP* є протоколом, який використовується для моніторингу та управління мережевими пристроями. Основні компоненти протоколу:

1. Агенти – вони запускаються на мережевих пристроях (наприклад, маршрутизаторах, комутаторах) та забезпечують інформацію про стан пристрою.

2. Менеджери – це програмне забезпечення, яке використовується адміністраторами для моніторингу та керування мережею. Менеджери взаємодіють з агентами, запитуючи їх стан та виконуючи управлінські функції.

3. *MIB* (*Management Information Base*) – це база даних, яка зберігає інформацію про доступні параметри та значення, які можуть бути запитані агентами.

4. Оператори *SNMP* – Вони визначають правила доступу до *MIB* та механізми повідомлень для сповіщення про події у мережі.

Протокол використовує *UDP* (*User Datagram Protocol*) для передачі даних. Комунікація відбувається за допомогою *PDU* (*Protocol Data Units*), які містять запити або відповіді на запити.

SSH (*Secure Shell*) є криптографічним протоколом, який забезпечує безпечний доступ до мережевих пристроїв для керування ними та передачі даних. Основні характеристики:

1. Шифрування – *SSH* використовує асиметричне шифрування для захисту даних від несанкціонованого доступу під час передачі.

2. Автентифікація – це процес підтвердження ідентичності користувача перед наданням доступу. *SSH* підтримує різні методи автентифікації, такі як пароль, ключі *SSH*, а також автентифікація на основі сертифікатів.

3. Тунелювання – *SSH* дозволяє створювати зашифровані тунелі для безпечної передачі даних між двома вузлами мережі.

Далі також будуть наведені та описані інші протоколи для отримання даних про стан мережевого обладнання такі, як *Telecommunication Network*, *NetFlow* та *Syslog*.

Telnet (*Telecommunication Network*) є простим текстовим протоколом, який використовується для віддаленого доступу до мережевих пристроїв. Основні особливості *Telnet*:

1. *Telnet* дозволяє віддалено підключатися користувачам до мережевих пристроїв, таких як маршрутизатори, комутатори або сервери, і виконувати команди через текстовий інтерфейс.

2. Має простий текстовий інтерфейс, де користувачі взаємодіють з мережевим пристроєм шляхом введення команд у текстовому форматі.

3. Обмін даними незашифрований, що є мінусом. *Telnet* передає дані у відкритому текстовому форматі, тому інформація може бути легко перехоплена. Оскільки дані передаються у незашифрованому вигляді, *Telnet* є менш безпечним в порівнянні з іншими протоколами, такими як SSH.

4. *Telnet* використовує *TCP*-порт 23 для встановлення з'єднання між клієнтом і сервером.

У зв'язку зі своєю низькою безпекою, рекомендується використовувати більш безпечні альтернативи, такі як *SSH*, для віддаленого доступу до мережевих пристроїв.

NetFlow є протоколом моніторингу трафіку, розробленим компанією *Cisco*, і використовується для збору та аналізу даних про трафік, що проходить через мережеві пристрої. Основні характеристики *NetFlow*:

1. Дозволяє збирати дані про обсяги, напрямок та характер трафіку, що проходить через мережеві пристрої, такі як маршрутизатори та комутатори.

2. Зібрані дані можуть бути використані для аналізу трафіку в мережі, виявлення аномалій та оптимізації ресурсів.

3. Реалізовано ресурсозбереження, тобто використання *NetFlow* допомагає економити пропускну здатність мережі та зменшувати навантаження на пристрої за рахунок збору статистики використання трафіку.

4. *NetFlow* підтримує як *IPv4*, так і *IPv6*, що дозволяє використовувати його для моніторингу різних типів мереж. Це забезпечує гнучкість у налаштуванні та адаптацію до різноманітних мережевих середовищ.

Syslog є протоколом журналювання подій, який використовується для збору та аналізу журналів подій з мережевого обладнання. Основні особливості *Syslog*:

1. Збір журналів подій реалізований наступним чином, *Syslog* дозволяє мережевим пристроям надсилати журнали подій (логи) на центральний сервер для зберігання та аналізу.

2. Зібрані логи можуть бути централізовано аналізовані та моніторені адміністраторами для виявлення проблем та подій у мережі.

3. *Syslog* підтримує різні рівні журналювання, такі як інформаційні повідомлення, попередження, помилки тощо, що дозволяє класифікувати події за їх важливістю.

4. Логи *Syslog* зазвичай представлені у простому текстовому форматі, що дозволяє легко їх аналізувати та обробляти.

5. Протокол *Syslog* підтримується більшістю мережевих пристроїв та операційних систем, що робить його універсальним інструментом для журналювання подій у різних мережевих інфраструктурах.

6. *Syslog* дозволяє налаштовувати фільтри та правила для відправки лише певних типів повідомлень, що допомагає зменшити обсяг даних та зосередитися на найбільш критичних подіях.

І також *HTTP*/*HTTPS*, опис цього протоколу було вже наведено вище, але ще можна зауважити, що деякі мережеві пристрої можуть мати веб-інтерфейс, який дозволяє отримувати дані про їх стан за допомогою протоколів *HTTP* або *HTTPS*.

У висновку, отримання даних про стан мережевого обладнання є складним та багатогранним процесом, який включає в себе використання різноманітних методів та інструментів. Правильне збирання, аналіз та використання цих даних дозволяє забезпечити надійну та ефективну роботу мережі, зменшити час простою та виявити потенційні проблеми заздалегідь.

1.3 Огляд поточних систем відображення стану мережевого обладнання

SolarWinds NPM (*Network Performance Monitor*) – це комерційна система моніторингу, що пропонує широкий спектр функцій для моніторингу мережевих пристроїв, серверів, додатків та віртуальних машин, як зазначено в джерелі [6].

Система масштабована, що робить її привабливою для організацій будь-якого розміру, від малого та середнього бізнесу до великих підприємств. Вона здатна адаптуватися до зростання мережі, забезпечуючи стабільну роботу навіть при значному збільшенні кількості пристроїв та обсягу трафіку. *SolarWinds NPM* підтримує різноманітні платформи, включаючи такі операційні системи, як *Windows, Linux* та *macOS*, що дозволяє інтегрувати її в будь-яке середовище *IT*-інфраструктури.

Ця система використовує агентів, які встановлюються на мережевих пристроях для збору даних. Ці дані потім аналізуються та візуалізуються на дашбордах, звітах та картах мережі. Як виглядає інтерфейс цієї системи можна побачити на рисунку 1.4.

Характеристики:

1. *SolarWinds NPM* надає широкий спектр можливостей для моніторингу різних типів мережевого обладнання, включаючи маршрутизатори, комутатори, сервери, фаєрволи та інші.
2. Система забезпечує гнучкі інструменти для візуалізації даних про стан мережі, включаючи графіки, діаграми, теплові карти та інші.
3. *SolarWinds NPM* дозволяє аналізувати мережевий трафік за допомогою *NetFlow* та інших протоколів, що допомагає виявляти проблеми та оптимізувати ресурси. Система підтримує також протоколи *sFlow, J-Flow, IPFIX* та інші, що надає адміністраторам можливість отримувати детальну інформацію про використання мережі та її стан.
4. Система надає можливості налаштування сповіщень про виникнення проблем, а також генерацію звітів про стан мережі.

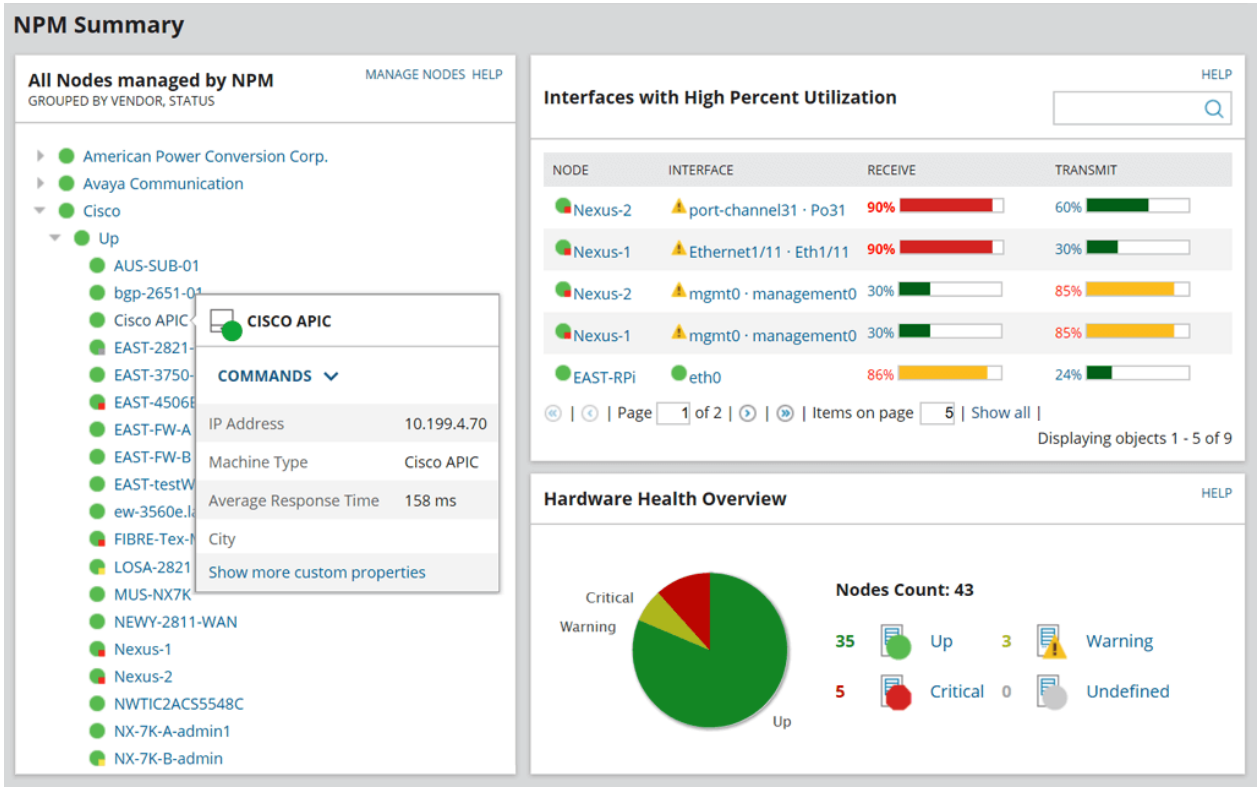


Рис. 1.3 – приклад інтерфейсу *SolarWinds NPM*

PRTG Network Monitor – це ще одна комерційна, але інтегрована система моніторингу, яка забезпечує відображення стану мережевого обладнання, аналіз трафіку, сповіщення про проблеми та генерацію звітів, як наведено в джерелі [7].

Система масштабована та підтримує різноманітні платформи, включаючи *Windows*, *Linux* та *macOS*. *PRTG Network Monitor* використовує агентів для збору даних з мережевих пристроїв. Приклад, як виглядає інтерфейс системи можна побачити на рисунку 1.4.

Характеристики:

1. *PRTG* має інтуїтивний інтерфейс, що дозволяє швидко налаштовувати моніторинг та отримувати інформацію про стан мережі.
2. Система може моніторити мережі будь-якої складності, починаючи з невеликих офісних мереж до великих корпоративних інфраструктур.
3. *PRTG* забезпечує широкі можливості налаштування сповіщень про виникнення проблем через різні канали зв'язку.
4. Система дозволяє користувачам визначати власні параметри моніторингу та налаштовувати їх відповідно до власних потреб.

5. *PRTG* тепер доступний на мобільних пристроях, що дозволяє користувачам відстежувати стан своїх мереж з будь-якого місця.
6. *PRTG* пропонує нові та вдосконалені опції візуалізації даних, які дозволяють користувачам легко отримувати доступ до інформації та аналізувати тенденції.

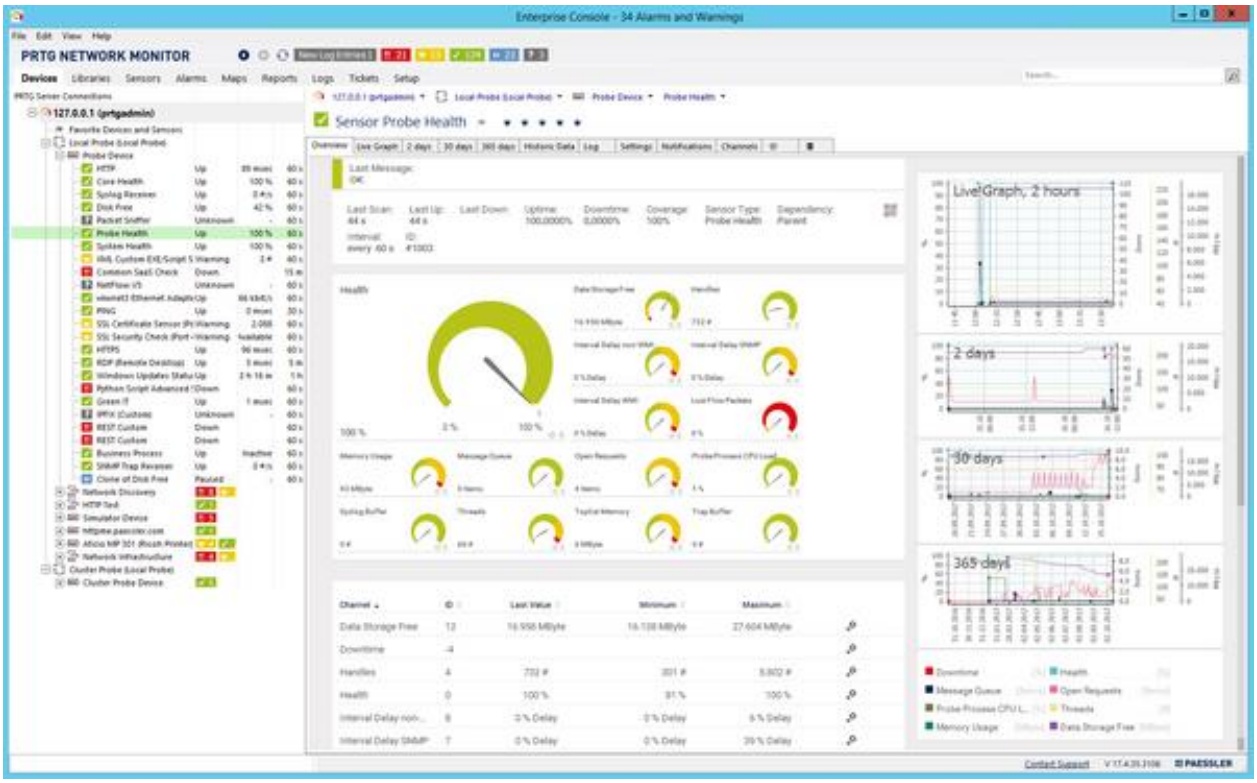


Рис. 1.4 – Приклад інтерфейсу *PRTG Network Monitor*

Zabbix – це відкрите програмне забезпечення для моніторингу мережі та інфраструктури, яке надає можливості моніторингу різних типів пристроїв і систем. Система підтримує різноманітні платформи, включаючи *Linux*, *Unix* та *Windows*. *Zabbix* може використовувати або не використовувати агентів для збору даних з мережевих пристроїв. Крім того, *Zabbix* забезпечує можливості налаштування повідомлень, інтеграцію з різними системами оповіщення, а також візуалізацію даних через інтерактивні графіки та дашборди. Він також підтримує автоматичне виявлення мережевих пристроїв і самонавчання, що робить його потужним інструментом для великих мережевих інфраструктур. Приклад того, як виглядає інтерфейс системи наведено на рисунку 1.5.

Характеристики:

1. *Zabbix* дозволяє користувачам налаштовувати різноманітні параметри моніторингу та створювати власні шаблони для відслідковування потрібних метрик.

2. Система може легко масштабуватися для відслідковування мереж будь-якої складності, від невеликих офісних мереж до великих корпоративних інфраструктур.

3. *Zabbix* надає засоби візуалізації даних, такі як графіки та діаграми, для аналізу стану мережі.

4. Система дозволяє налаштовувати сповіщення про події та генерувати звіти про стан мережі.

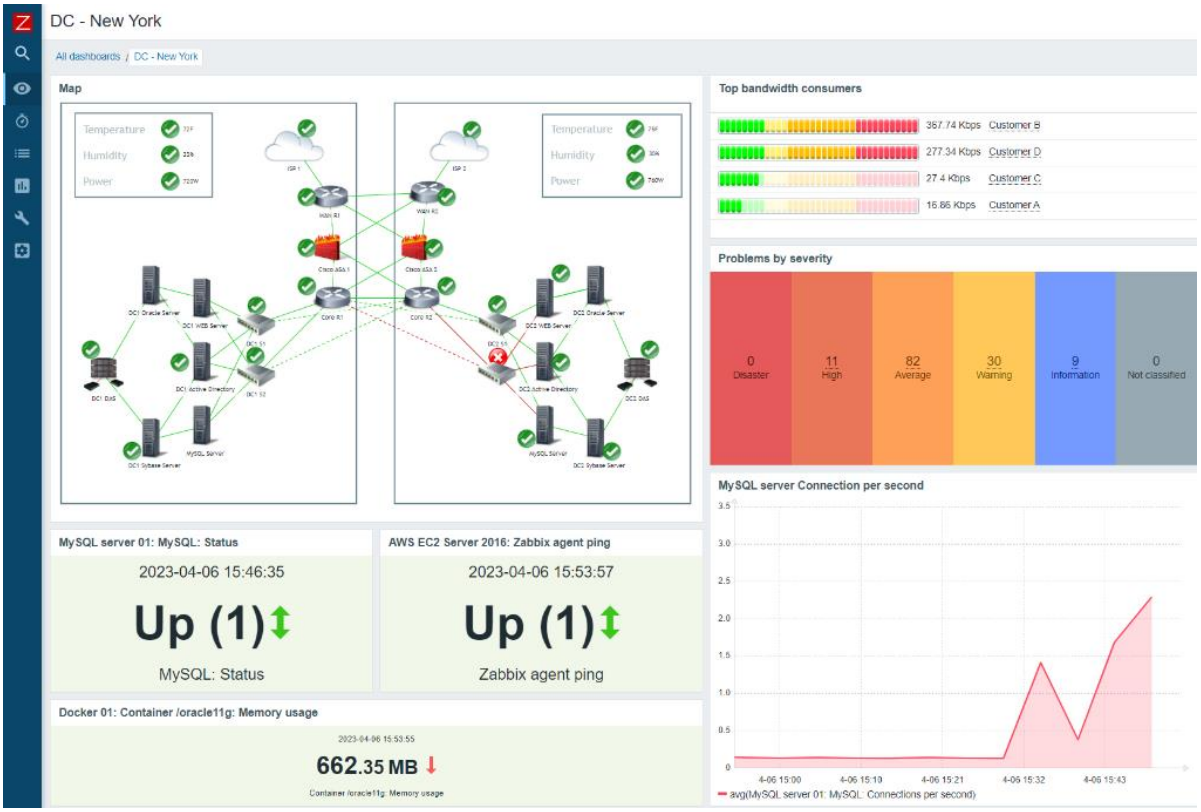


Рис. 1.5 – Приклад інтерфейсу Zabbix network monitoring

Nagios – це відкрите програмне забезпечення для моніторингу систем, мереж та інфраструктури, яке надає можливості моніторингу різних типів пристроїв та систем, включаючи сервери, маршрутизатори, комутатори та програмне забезпечення, згідно із джерелом [9]. Так само, як і попередня система

Nagios підтримує різноманітні платформи, включаючи *Linux*, *Unix* та *Windows*. І також може використовувати або не використовувати агентів для збору даних з мережевих пристроїв. Приклад, як виглядає інтерфейс цієї системи наведено на рисунку 1.6.

Характеристики:

1. *Nagios* дозволяє користувачам налаштовувати різноманітні параметри моніторингу та створювати власні шаблони для відслідковування потрібних метрик.

2. Система може легко масштабуватися для відслідковування мереж будь-якої складності, від невеликих офісних мереж до великих корпоративних інфраструктур.

3. *Nagios* надає можливості візуалізації даних за допомогою графіків, діаграм та інших інструментів.

4. Система дозволяє налаштовувати сповіщення про події та генерувати звіти про стан мережі.

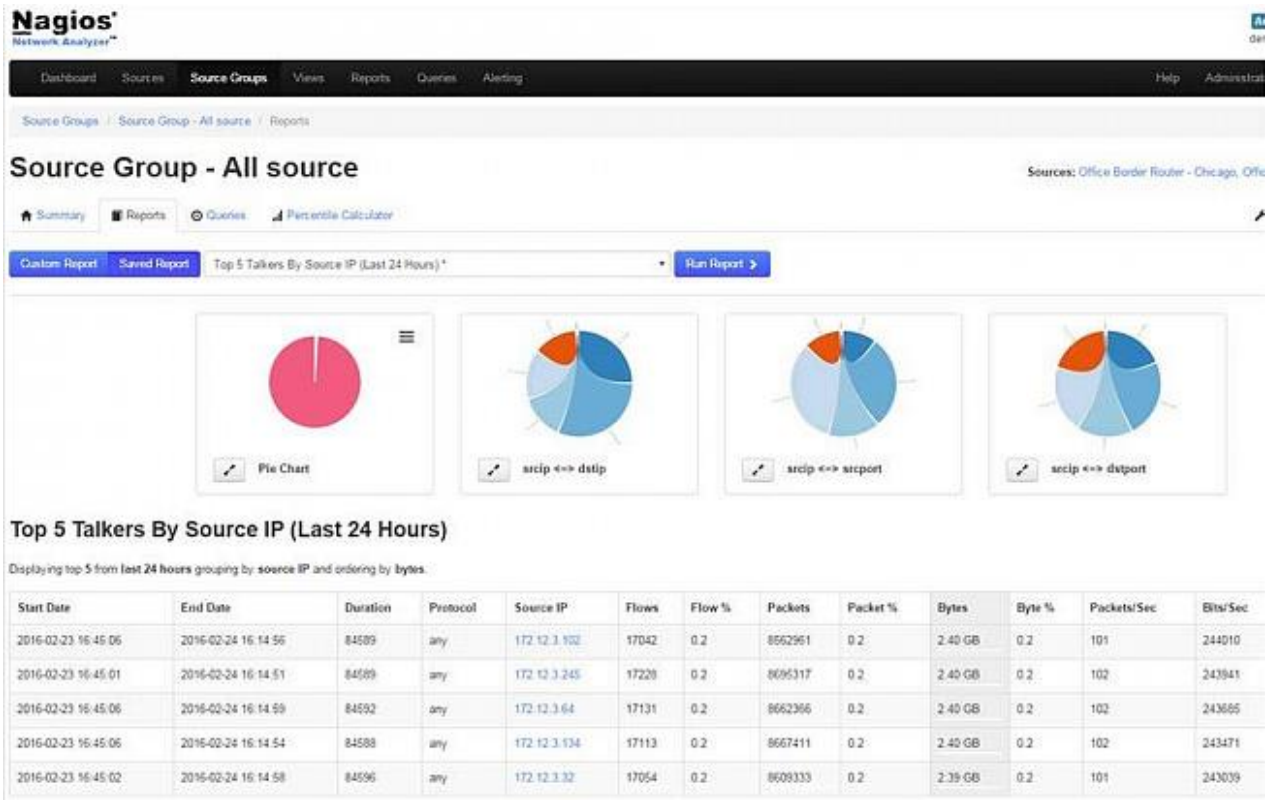


Рис. 1.6 – Приклад інтерфейсу *Niagos network analyzer*

Cisco Prime Infrastructure – це комерційна система моніторингу, призначена для управління мережевими пристроями *Cisco*, як зазначено в джерелі [10].

Система пропонує комплексні можливості моніторингу, включаючи видимість мережі, аналіз продуктивності, управління конфігурацією та автоматизацію. *Cisco Prime Infrastructure* призначена для організацій, що використовують переважно обладнання *Cisco* у своїй мережі. Приклад, як виглядає інтерфейс системи наведено на рисунку 1.7.

Характеристики:

1. *Cisco Prime Infrastructure* спеціалізується на інтеграції з обладнанням *Cisco*, що дозволяє забезпечити високу сумісність та функціональність.
2. Система пропонує автоматизовані засоби управління та моніторингу, що спрощує адміністрування мережі.
3. *Cisco Prime Infrastructure* надає можливості для аналізу безпеки мережі та виявлення загроз.
4. Система дозволяє генерувати звіти та аналізувати дані про стан мережі для прийняття ефективних управлінських рішень.

Рис. 1.7 – Приклад інтерфейсу *Cisco Prime Infrastructure*

ManageEngine OpManager – це також комерційна та інтегрована система моніторингу мережі, розроблена компанією *ManageEngine*, як зазначено в джерелі [11]. Вона надає інструменти для відображення стану мережевого обладнання, моніторингу трафіку, аналізу пропускної здатності та виявлення проблем. Система масштабована та також підтримує різноманітні платформи, включаючи *Windows* та *Linux* і використовує агентів для збору даних з мережевих пристроїв. Приклад, як виглядає інтерфейс системи наведено на рисунку 1.8.

Характеристики:

1. *ManageEngine OpManager* пропонує різноманітні можливості для моніторингу різних типів пристроїв та систем у мережі.
2. Система дозволяє користувачам налаштовувати різні параметри моніторингу та адаптувати їх до власних потреб.
3. *ManageEngine OpManager* надає автоматизовані засоби для виявлення проблем, що дозволяє забезпечити швидку реакцію на них.
4. Система може інтегруватися з іншими продуктами компанії *ManageEngine* та іншими сторонніми системами для розширення функціональності.

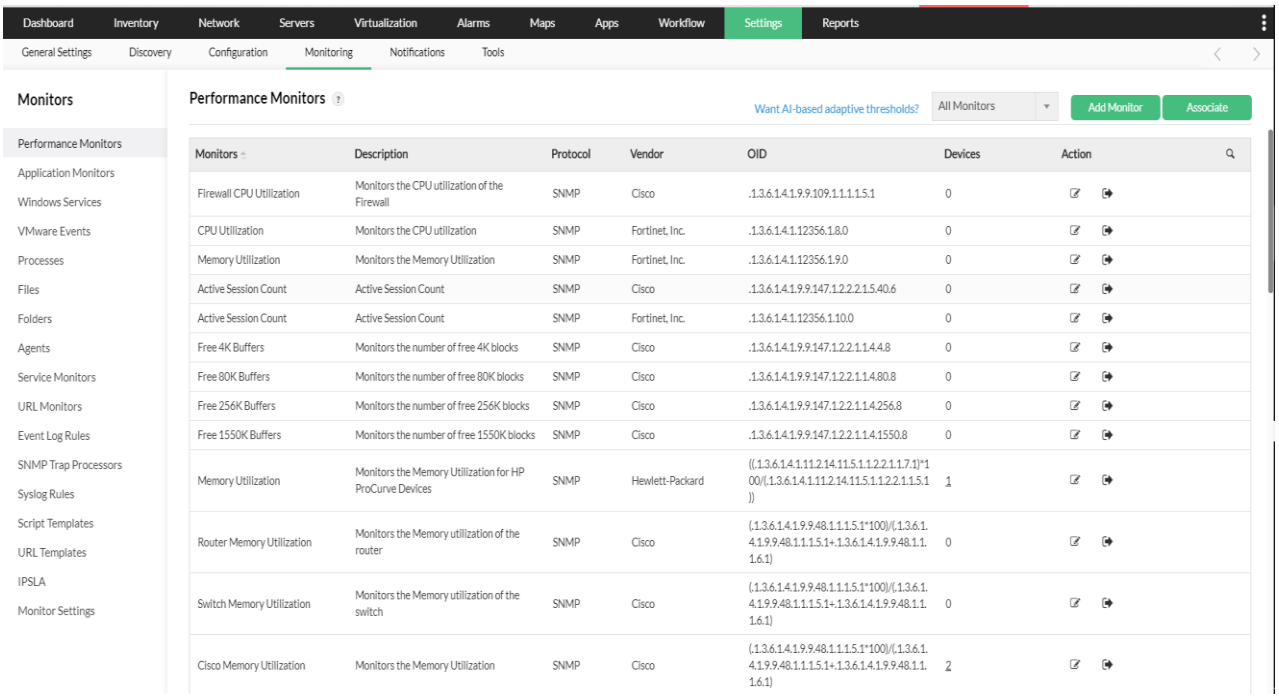


Рис. 1.8 – Приклад інтерфейсу *ManageEngine OpManager*

Детальні, порівняльні характеристики вище описаних, перших трьох моніторингових веб-систем наведені у таблиці 1.1.

Таблиця 1.1

Порівняльні характеристики вище описаних, перших трьох моніторингових веб-систем

Характеристика	***SolarWinds NPM***	***PRTG Network Monitor***	***Zabbix***
Тип ліцензування	Комерційний	Комерційний	*Open-source*
Масштабованість	Висока	Висока	Висока
Підтримка платформ	*Windows, Linux*	*Windows, Linux*	*Linux, Unix, Windows*
Агент	Необхідний	Необхідний	Необхідний
Методи моніторингу	*SNMP, WMI, CLI, Ping, Flow*	*SNMP, WMI, CLI, Ping, Flow*	*SNMP, WMI, CLI, Ping, Flow*
Візуалізація	Дашборди, звіти, карти мережі	Дашборди, звіти, карти мережі	Дашборди, звіти, карти мережі
Аналітика	Пошук несправностей, кореляція подій, прогнозування	Пошук несправностей, кореляція подій, прогнозування	Пошук несправностей, кореляція подій, прогнозування
Автоматизація	Автоматичне реагування на події, сценарії	Автоматичне реагування на події, сценарії	Автоматичне реагування на події, сценарії
Ціна	Залежно від кількості пристроїв	Залежно від кількості датчиків	Безкоштовно для до 100 хостів, платний для більшої кількості

Детальні, порівняльні характеристики трьох наступних веб-систем наведені у таблиці 1.2.

Таблиця 1.2

Порівняльні характеристики трьох наступних моніторингових веб-систем

Характеристика	***Nagios***	***Cisco Prime Infrastructure***	***ManageEngine OpManager***
Тип ліцензування	*Open-source*	Комерційний	Комерційний
Масштабованість	Висока	Висока	Висока
Підтримка платформ	*Linux, Unix, Windows,*	*Cisco*	*Windows, Linux*
Агент	Необхідний	Необхідний	Необхідний
Методи моніторингу	*SNMP, WMI, CLI, Ping*	*SNMP, WMI, CLI, Ping, Flow*	*SNMP, WMI, CLI, Ping, Flow*
Візуалізація	Дашборди, звіти	Дашборди, звіти, карти мережі	Дашборди, звіти, карти мережі
Аналітика	Пошук несправностей	Пошук несправностей, кореляція подій	Пошук несправностей, кореляція подій
Автоматизація	Автоматичне реагування на події	Автоматичне реагування на події, сценарії	Автоматичне реагування на події, сценарії
Ціна	Безкоштовно для до 50 хостів, платний для більшої кількості	Залежно від кількості пристроїв *Cisco*	Залежно від кількості пристроїв

1.4 Розробка технічного завдання

У сучасних IT-інфраструктурах *network monitoring system* (*NMS*) – системи моніторингу мережевого обладнання відіграють важливу роль, забезпечуючи адміністраторам візуалізацію, аналіз та контроль за працездатністю мережевих пристроїв.

Однак, існуючі *NMS* мають певні недоліки, які і розберемо в цьому пункті та, які можна виправити при розробці власної веб-системи відображення стану мережевого обладнання.

На основі огляду шести систем із минулого пункту можна виділити наступні їх недоліки:

1. Складність, яка заключається в тому, що деякі *NMS*, такі як *SolarWinds NPM* та *PRTG Network Monitor*, є складними у налаштуванні та користуванні, що робить їх недоступними для користувачів з обмеженими технічними знаннями.

2. Висока вартість заключається в тому, що більшість комерційних *NMS*, тих самих *SolarWinds NPM* та *PRTG Network Monitor*, мають високу вартість, що робить їх недоступними для малого та середнього бізнесу, навчальних закладів та інших організацій, яким потрібен простий, ефективний та функціональний інструмент для моніторингу своєї мережі.

3. Відсутність гнучкості, тобто деякі *NMS*, такі як *Cisco Prime Infrastructure*, призначені для роботи з обладнанням певного виробника, що робить їх негнучкими для використання в мережах з обладнанням різних виробників.

4. Складний інтерфейс командного рядка мають деякі *NMS*, такі як *Nagios*, що робить їх незручними для користувачів, які не знайомі з командним рядком.

5. Деякі *NMS* не мають мобільних додатків, що робить їх недоступними з мобільних пристроїв.

6. деякі *NMS* мають обмеження на кількість пристроїв або обсяг даних, які вони можуть обробляти.

Згідно із вищезазначеними недоліками, до можливих напрямків вдосконалення можна віднести наступні напрямки, які можна реалізувати в розробці моєї веб-системи.

Тож перше це розробка простого та інтуїтивного інтерфейсу. Інтерфейс системи повинен бути максимально простим та зрозумілим для користувачів з будь-яким рівнем технічної підготовки.

Також важливо забезпечити гнучкість та масштабованість. Система повинна бути гнучкою та масштабованою, щоб її можна було використовувати для моніторингу мереж будь-якого розміру та складності.

Реалізація візуалізації даних в режимі реального часу. Система повинна надавати візуальні уявлення даних про стан мережевого обладнання в режимі реального часу, дозволяючи користувачам відстежувати динаміку роботи мережі.

Розробка механізмів автоматичного реагування. Система може автоматично реагувати на певні події, наприклад, перезавантажувати мережеві пристрої у разі зависання, або інформувати про якісь помилки.

І звісно забезпечення необхідного рівня безпеки. Система повинна бути захищена від несанкціонованого доступу та атак. Необхідно реалізувати механізми аутентифікації та авторизації користувачів, а також шифрування даних.

Висновок до першого розділу

У цьому розділі було розглянуто основи мережевих технологій та протоколів, які є фундаментом для розробки веб-системи відображення стану мережевого обладнання. Було досліджено архітектуру систем моніторингу, описано принципи отримання даних про стан мережевого обладнання та проаналізовано існуючі системи моніторингу.

Зокрема, було розглянуто основні мережеві протоколи, такі як *TCP/IP*, *HTTP*, *HTTPS*, *SNMP*, *SMTP*, *ICMP* та *IPMI*, які використовуються для передачі даних, моніторингу та управління мережевим обладнанням. Було описано

клієнт-серверну архітектуру, яка є основою для багатьох веб-систем, та розглянуто принципи отримання даних про стан мережевого обладнання за допомогою різних протоколів та інструментів.

В результаті аналізу існуючих систем моніторингу було виявлено їх переваги та недоліки. Було встановлено, що більшість систем мають складний інтерфейс, високу вартість та обмежену функціональність. Це дозволило визначити основні напрямки вдосконалення для розробки власної веб-системи, яка буде простою у використанні, гнучкою, масштабованою та функціональною. Визначені напрямки вдосконалення будуть враховані при розробці технічного завдання на наступних етапах роботи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ВЕБ-СИСТЕМИ

2.1 Вступ до проектування веб-системи

Система *NetworkMonitor* забезпечує адміністраторів мережі інструментами для отримання вичерпної інформації про стан мережевого трафіку, відкриті порти та підключені пристрої у реальному часі. Основними завданнями системи є збір даних через *SSH*-з'єднання з віддалених серверів, аналіз отриманих даних та їх візуалізація через веб-інтерфейс.

При проектуванні системи NetworkMonitor були враховані різноманітні вимоги, які забезпечують її функціональність, надійність та зручність використання. Основні функціональні вимоги включають можливість збору даних про мережевий трафік, порти та підключені пристрої, їх аналіз та візуалізацію. Система також повинна забезпечувати реальний моніторинг стану мережевого обладнання. Нефункціональні вимоги включають високу продуктивність, безпеку, масштабованість та зручність використання системи. Крім того, важливою є підтримка багатокористувацького режиму з різними рівнями доступу.

У цьому розділі ми розглянемо такі аспекти проектування системи *NetworkMonitor*:

- загальний опис архітектури системи, включаючи основні компоненти та їх взаємодію;
- обґрунтування вибору технологій та інструментів, які були використані для реалізації системи;
- детальний опис основних модулів системи, їх функціональності та взаємодії;
- заходи, які були вжиті для забезпечення безпеки та захисту даних у системі.

2.2 Архітектура системи

Система *NetworkMonitor* має багатошарову архітектуру, яка включає кілька основних компонентів: користувацький інтерфейс, серверну частину та базу даних. Користувацький інтерфейс надає адміністраторам зручні інструменти для взаємодії з системою, серверна частина забезпечує обробку запитів та управління даними, а база даних зберігає зібрану інформацію про мережевий трафік та підключені пристрої.

Клієнтська частина включає *HTML*-сторінки, стилі *CSS* та скрипти *JavaScript*, які забезпечують взаємодію з користувачем. Вона відповідає за відображення даних, отриманих від серверної частини, та надсилання запитів до серверної частини через *AJAX*-запити за допомогою бібліотеки *jQuery*.

В ресурсі [12] розповідається, що *AJAX* (*Asynchronous JavaScript and XML*) – це рішення, яке дозволяє асинхронно запитувати чи передавати дані на сервер. Сама ідея полягає в тому, щоб не перезавантажуючи веб-сторінку та не блокуючи користувальницький інтерфейс, можна було б з *JavaScript*-сценаріїв звернутися до будь-якого серверного скрипта, або іншого ресерсу для отримання чи передачі будь-яких даних.

Для реалізації *AJAX*-запитів в браузери вбудовані компоненти, до яких можна отримати доступ з *JavaScript*-коду, але є одна неприємна проблема. Код створення *AJAX*-об’єкта в *IE* (*Internet Explorer*), відрізняється від коду створення *AJAX*-об’єкта в інших браузерах. Це, мабуть, найбільша складність. Тому що не дуже хочеться замислюватися про різні налаштування і низькорівневі реалізації *AJAX*. Виходячи з цього *jQuery* позбавляє від попередньо згаданих проблем, надаючи дуже зручний інтерфейс. Потрібно сказати, що застосування даної бібліотеки далеко не обмежується тільки зручною реалізацією *AJAX*-сценаріїв.

Другою частиною архітектури є серверна частина, яка відповідає за обробку запитів, виконання бізнес-логіки, взаємодію з базою даних та формування відповідей для клієнтської частини. Вона складається з кількох ключових компонентів:

– контролерів (*Controllers*);

– сервісів (*Services*);

– моделів (*Models*).

Далі на рисунку 2.1 буде наведено діаграму компонентів, яка ілюструє взаємодію конролерів, сервісів та моделей.

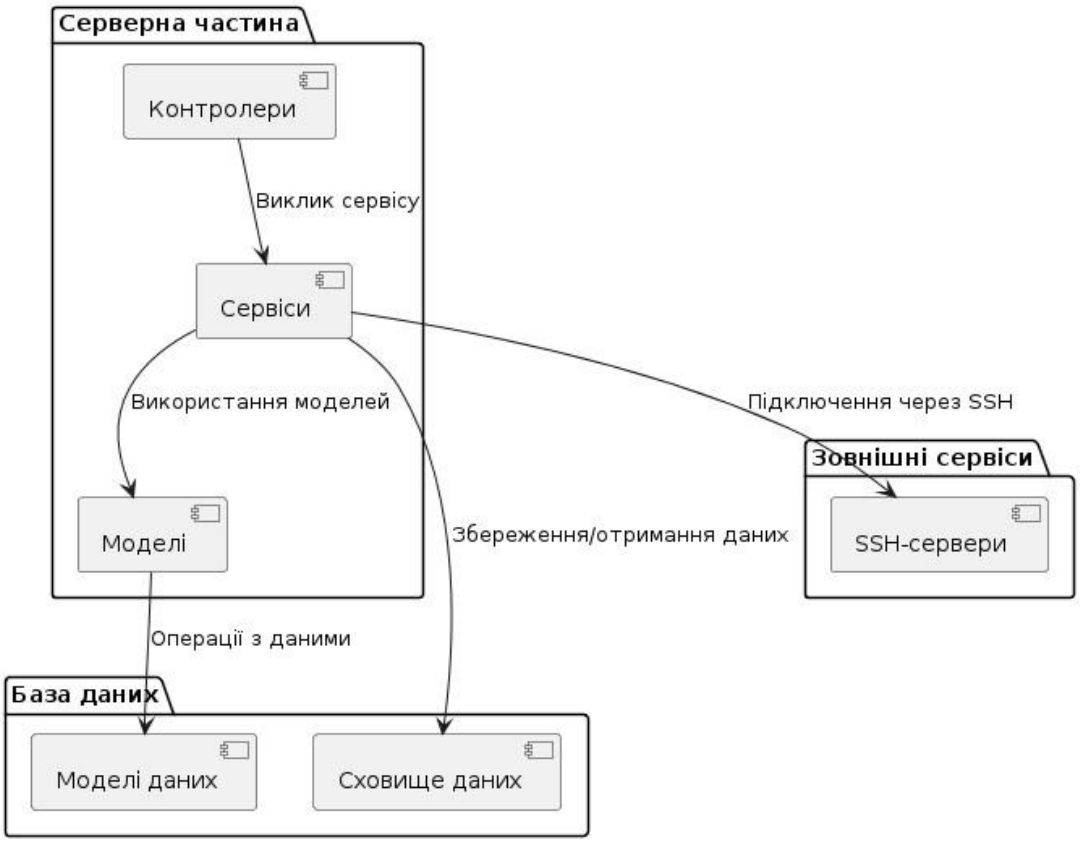


Рис. 2.1 – Діаграма компонентів, яка ілюструє взаємодію конролерів, сервісів та моделей

Контролери відповідають за прийом *HTTP*-запитів від клієнтської частини. Вони обробляють ці запити, викликають відповідні сервіси та формують відповіді. Контролери служать точками входу для користувацьких запитів, тому ці контролери є невід'ємною частиною архітектури веб-додатка.

Прикладом такого контролера в системі є *HomeController*, який обробляє запити до домашньої сторінки та сторінки конфіденційності, а також обробляє помилки.

Сервіси виконують бізнес-логіку системи. Вони обробляють дані, взаємодіють з контекстом бази даних та зовнішніми системами. Сервіси

відповідають за виконання складних операцій та забезпечення узгодженості даних.

Як приклад можна навести *SshService*, який збирає дані про мережевий трафік, порти та підключені пристрої з віддалених серверів за допомогою *SSH*-з'єднань.

Моделі представляють структуру даних, які використовуються в системі. Вони визначають поля, типи даних та зв'язки між різними сутностями.

Прикладом таких моделей є *NetworkStatistics, PortData, ConnectedDevice,* які представляють дані про мережевий трафік, порти та підключені пристрої відповідно.

Третьою частиною архітектури є база даних, що зберігає всю необхідну інформацію для роботи системи, включаючи дані про мережевий трафік, порти та підключені пристрої. Для взаємодії з базою даних використовується *Entity Framework Core*, який забезпечує ефективний доступ до даних та їх обробку. Вибір *Entity Framework Core* обумовлений його здатністю працювати з різними системами управління базами даних і забезпечувати високий рівень продуктивності та безпеки. Це дозволяє створювати складні запити та операції з даними, що необхідні для глибокого аналізу та моніторингу мережевої інфраструктури.

За допомогою вище згаданого фреймворку контексти (*DbContext*) бази даних управляють доступом до неї та забезпечують *CRUD*-операції (створення, читання, оновлення, видалення). Вони налаштовують зв'язки між моделями та таблицями бази даних.

Прикладом являється *TrafficDbContext*, який визначає таблицю *NetworkStatistics* та забезпечує доступ до бази даних.

Діаграму компонентів, яка відображає загальну архітектуру даної веб-системи наведено на рисунку 2.2.

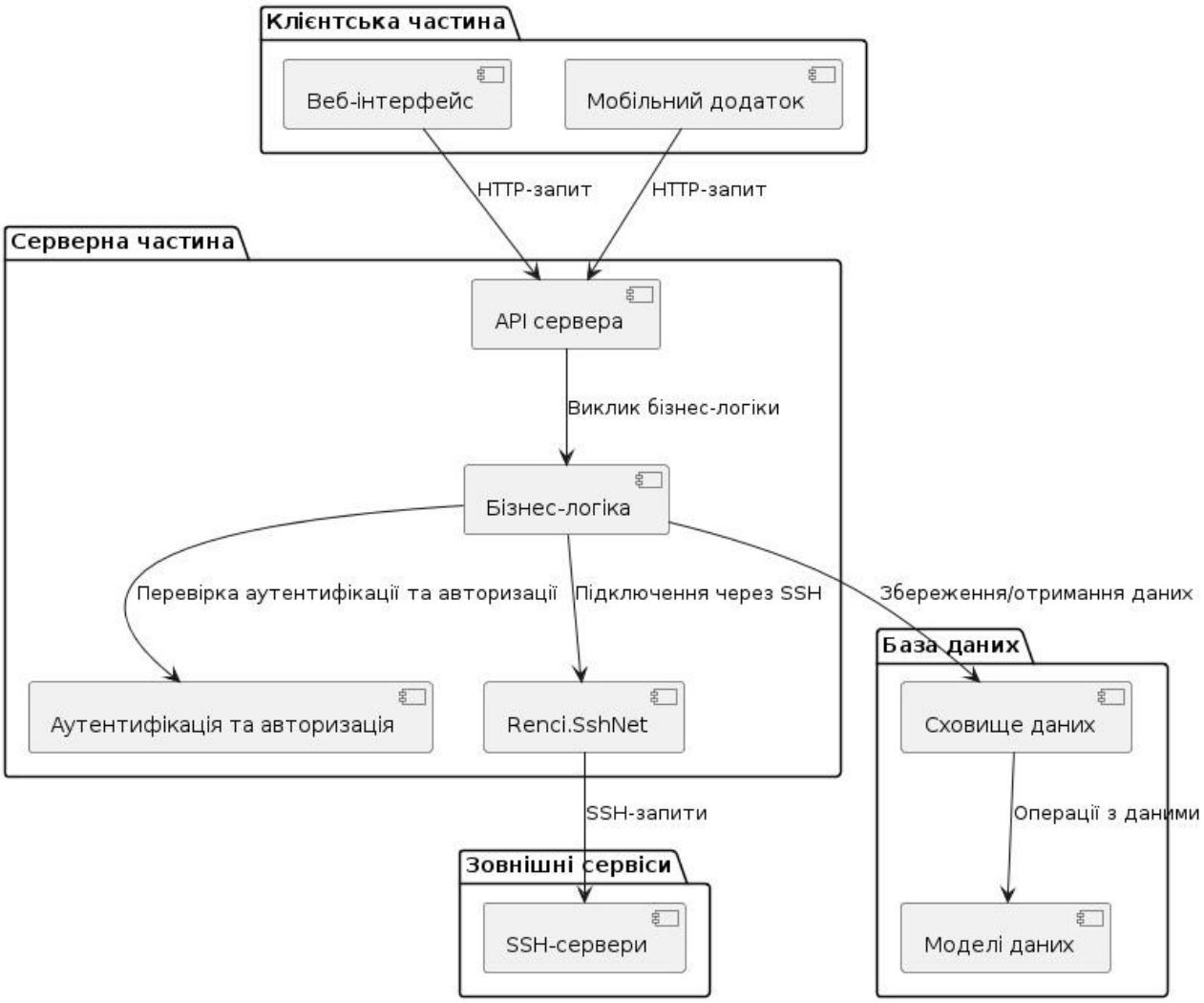


Рис. 2.2 – Діаграма компонентів, яка відображає загальну архітектуру даної веб-системи

Use-case діаграму, яка ілюструє сценарії використання системи, наведено в на рисунку 2.3, де зображені наступні елементи:

1. Актори:
   a) адміністратор – має доступ до всіх функцій системи, включаючи оновлення даних;
2. Варіанти використання:
   a) перегляд та оновлення загальної мережевої статистики;
   b) перегляд та оновлення даних про відкриті порти на сервері;
   c) перегляд та оновлення списку пристроїв, підключених до мережі.

Рис. 2.3 – Діаграма, яка ілюструє сценарії використання системи

Описуючи серверну частину, також варто згадати про кілька ключових класів, які взаємодіють між собою для забезпечення функціональності системи. Нижче наведено опис основних класів та призначення.

HomeController є контролером, що обробляє основні запити до веб-додатку. Він забезпечує обробку запитів до головної сторінки (*Index*), сторінки конфіденційності (*Privacy*) та сторінки помилок (*Error*). Основне призначення цього класу – маршрутизація запитів та повернення відповідних представлень (*views*) користувачу.

SshService є сервісом, що відповідає за взаємодію з віддаленим сервером через *SSH*. Він збирає мережеву статистику, дані про порти та підключені пристрої за допомогою виконання команд *SSH*. Цей клас використовує налаштування *SSH* для підключення до сервера та аналізує отримані дані, наповнюючи відповідні моделі.

TrafficDbContext є контекстом бази даних, який управляє доступом до бази даних та забезпечує *CRUD*-операції. Він містить набір об'єктів *DbSet<NetworkStatistics>*, які представляють таблицю *NetworkStatistics* в базі даних. Контекст бази даних створюється автоматично, якщо він не існує.

NetworkStatistics є моделлю, що представляє мережеву статистику. Вона містить дані про кількість отриманих та надісланих байтів, кількість унікастових та неунікастових пакетів, кількість помилок та інші статистичні дані. Ця модель використовується для зберігання та передачі даних про мережеву активність. Крім того, *NetworkStatistics* є моделлю, що представляє мережеву статистику.

PortData є моделлю, що представляє дані про порти. Вона містить інформацію про протокол, локальну та зовнішню адреси, порти, стан з'єднання, ідентифікатор процесу (*PID*) та ім'я процесу. Ця модель використовується для зберігання та передачі інформації про активні мережеві з'єднання.

ConnectedDevice є моделлю, що представляє підключені пристрої. Вона містить *IP*-адресу, *MAC*-адресу та тип пристрою. Ця модель використовується для зберігання та передачі інформації про пристрої, що підключені до мережі.

Повний варіант діаграми ключових класів системи наведено в додатку А (рисунок А.1).

Далі на рисунку 2.4 буде наведено скорочену версію діаграми ключових класів веб-системи.

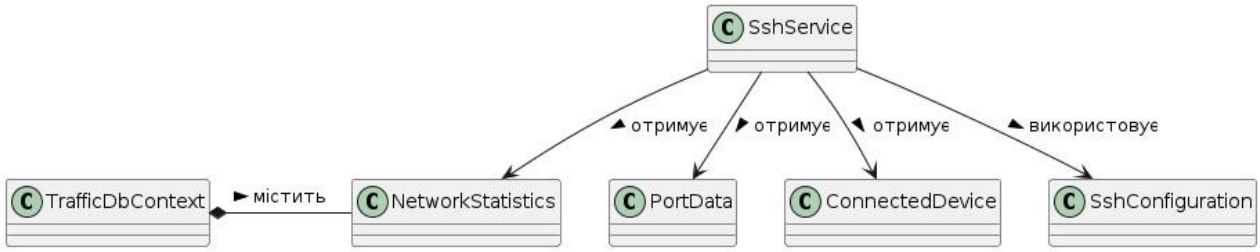


Рис 2.4 – Скорочена версія діаграми ключових класів веб-системи

2.3 Модулі веб-системи

Система *NetworkMonitor* складається з кількох основних модулів, кожен з яких виконує певні функції. Модуль збору даних відповідає за підключення до віддалених серверів через *SSH* та виконання команд для отримання інформації про мережевий трафік, порти та підключені пристрої. Модуль аналізу даних обробляє зібрані дані та зберігає їх у базі даних. Модуль візуалізації даних надає користувачам зручні інструменти для перегляду та аналізу отриманої інформації через веб-інтерфейс.

Діаграма компонентів в повному обсязі, що відображає взаємодію між модулями веб-системи зображена в додатку А (рисунок А.2). Далі на рисунку 2.5 буде наведено скорочену діаграму взаємодії модулів веб-системи.

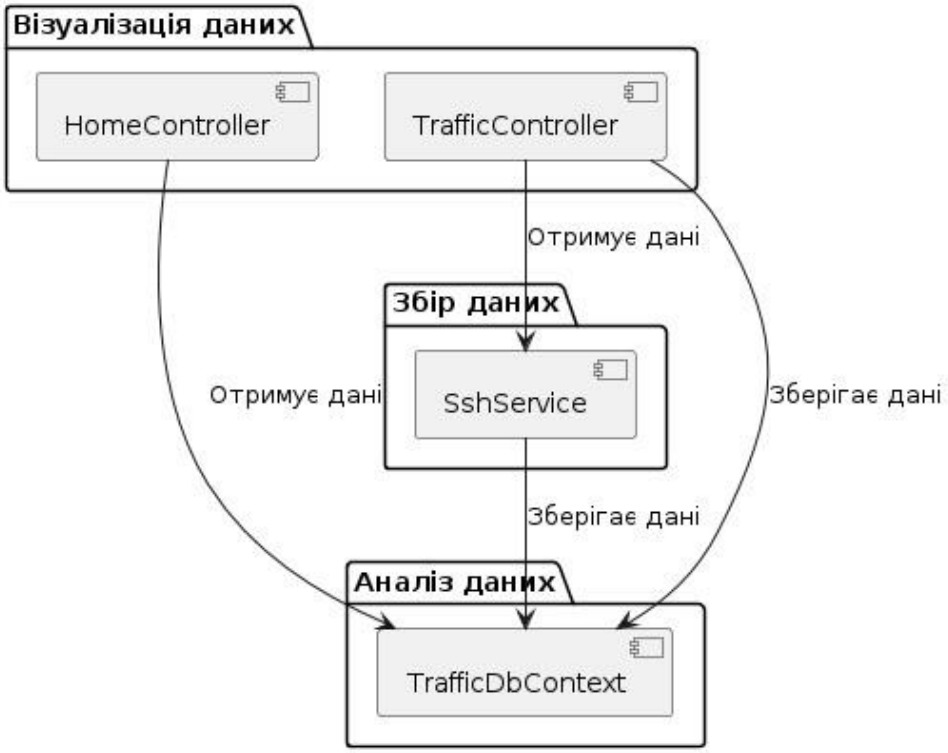


Рис. 2.5 – Скорочена діаграма компонентів, що відображає взаємодію модулів веб-системи

Модуль збору даних відповідає за підключення до віддалених серверів через *SSH* для збору мережевої статистики, даних про порти та підключені пристрої. Цей модуль реалізує основну функціональність збирання даних у реальному часі, що є ключовим для моніторингу мережі.

Модуль аналізу даних відповідає за обробку, фільтрацію та зберігання зібраних даних у базі даних. Цей модуль виконує основні операції з даними, такі як збереження нових записів, оновлення існуючих, а також виконання запитів для отримання необхідної інформації.

Модуль візуалізації даних відповідає за надання користувачам зручного інтерфейсу для перегляду зібраної та обробленої інформації. Цей модуль реалізує функціональність відображення даних у вигляді таблиць, графіків та інших візуальних елементів.

2.4 Безпека та захист даних

Безпека є критично важливим аспектом для системи *NetworkMonitor*. Для забезпечення захисту даних було реалізовано кілька рівнів безпеки, включаючи

захищені *SSH*-з'єднання для збору даних, шифрування збережених даних та даних, що передаються, за допомогою протоколу *HTTPS*. Описані нижче рівні безпеки забезпечують надійний захист від зовнішніх загроз і гарантують цілісність та конфіденційність даних.

Далі мова піде про захищені *SHH*-з'єднання. У цьому проекті, як вже було описано раніше, я використовую бібліотеку *Renci.SshNet* для встановлення захищених *SSH*-з'єднань з віддаленими серверами. Ця бібліотека забезпечує надійне шифрування та аутентифікацію, використовуючи такі основні механізми:

1. Шифрування – всі дані, що передаються через *SSH*-з'єднання, шифруються за допомогою сучасних алгоритмів шифрування (наприклад, *AES-256*), що забезпечує конфіденційність інформації під час передачі.

2. Аутентифікація – використовується ключова аутентифікація або аутентифікація за допомогою пароля. У випадку використання ключової аутентифікації, приватний ключ зберігається на клієнтському пристрої та не передається через мережу, що підвищує рівень безпеки.

Шифрування забезпечує конфіденційність і захищає дані від несанкціонованого доступу під час їх передачі та зберігання.

Дані, що передаються між клієнтом і сервером, шифруються за допомогою *HTTPS*. Це забезпечує захищений канал зв'язку та запобігає перехопленню даних.

Зібрані дані мережевої статистики та інформація про порти і підключені пристрої зберігаються у базі даних. Використання зашифрованих баз даних або додаткове шифрування чутливих даних гарантує їх безпеку у випадку компрометації бази даних.

Аутентифікація та авторизація забезпечують контроль доступу до системи та її ресурсів. У *NetworkMonitor* ці механізми поки не реалізовані так, як ще немає функціоналу, що міг би надаватись окремим користувачам. Розширифши функціонал, можна реалізувати аутентифікацію та авторизацію з використанням стандартних технологій *ASP.NET Core*.

Для аутентифікації можливе використання системи ідентифікації користувачів, що може включати аутентифікацію за допомогою пароля, *OAuth,* або інших методів. *ASP.NET Core Identity* є основним інструментом для цієї реалізації.

Авторизація забезпечує контроль доступу до різних частин системи на основі ролей користувачів або політик. *ASP.NET Core Authorization* дозволяє визначати ролі та політики доступу, які застосовуються до різних контролерів і дій.

2.5 Ілюстрація руху даних між компонентами системи та рівнями захисту

На рисунку 2.6 наведено діаграму потоку даних для ілюстрації руху даних між компонентами системи та рівнями захисту.

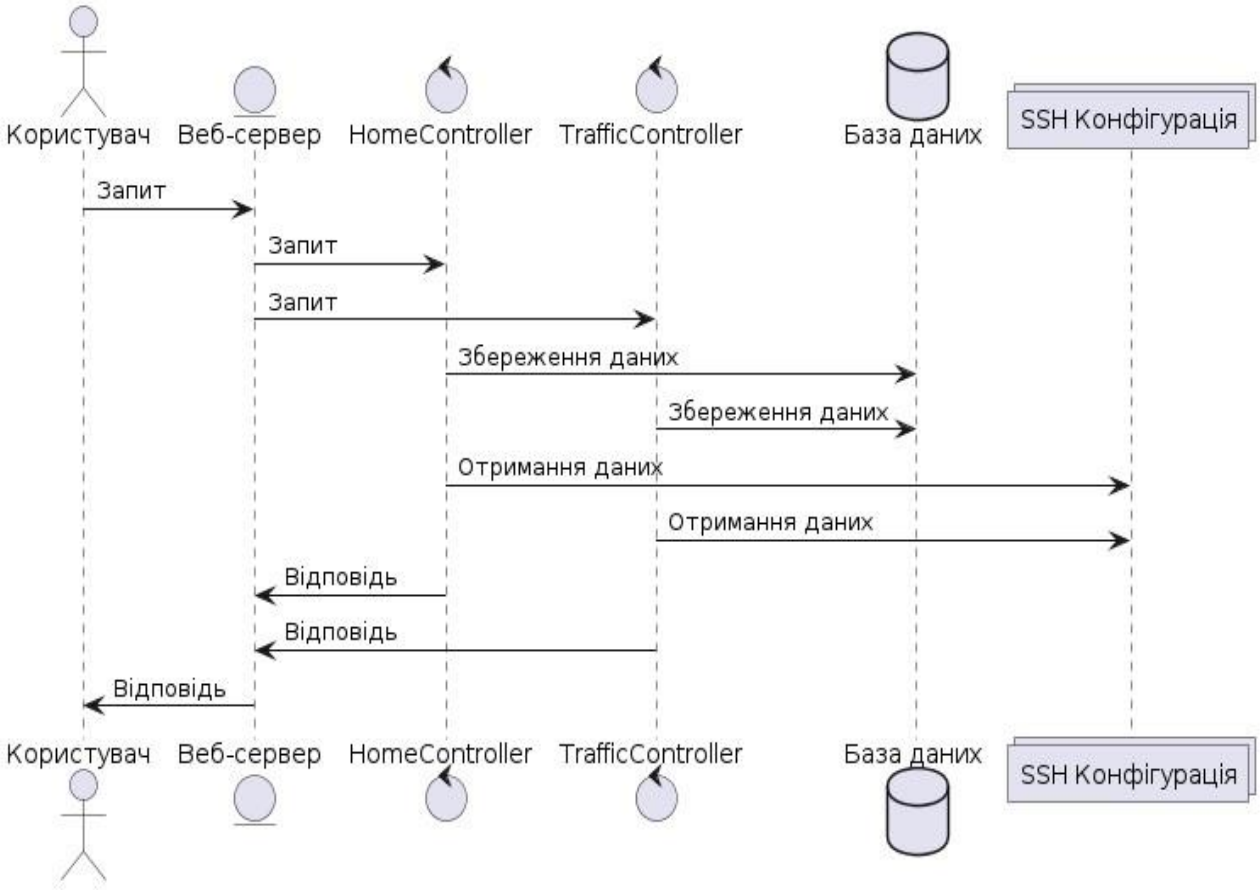


Рис. 2.6 – Діаграма ілюстрації руху даних між компонентами системи та рівнями захисту

2.6 Впровадження інших методів зробу даних з мережевого обладнання

В моїй веб-системі реалізований лише один метод для отримання даних про трафік, порти та підключені пристрої, і його може бути замало так, як не всі мережеві пристрої підтримують підключення по *SSH*. Тож далі буде описано, які ще можуть бути методи збору даних та їх майбутнє впровадження у веб-систему.

SNMP (*Simple Network Management Protocol*) є протоколом прикладного рівня, який використовується для управління та моніторингу мережевих пристроїв. Він дозволяє адміністраторам мережі отримувати інформацію про стан мережевих пристроїв та налаштовувати їх. *SNMP* працює за моделлю клієнт-сервер, де *SNMP*-агенти на мережевих пристроях збирають і зберігають інформацію, яку потім може запитувати *SNMP*-менеджер.

Основні компоненти *SNMP*:

- *SNMP*-агент – це програмне забезпечення, яке працює на мережевому пристрої та відповідає на запити від *SNMP*-менеджера;
- *SNMP*-менеджер – програмне забезпечення, яке збирає інформацію від агентів і керує мережевими пристроями;
- *MIB* (*Management Information Base*) – база даних, що містить інформацію про мережеві об'єкти, якими керує *SNMP*.

HTTP (*HyperText Transfer Protocol*) є основним протоколом для передачі даних в Інтернеті. Він використовується для обміну інформацією між веб-серверами та клієнтами (браузерами). *HTTP* може також використовуватись для отримання даних від мережевих пристроїв, якщо на них працює веб-інтерфейс або *API*.

Основні компоненти *HTTP*:

- *HTTP*-сервер – це програмне забезпечення, яке обслуговує запити від клієнтів;
- *HTTP*-клієнт – це програмне забезпечення або пристрій, який надсилає запити до сервера;

– *API* (*Application Programming Interface*) – це набір протоколів і інструментів для побудови програмного забезпечення, який дозволяє клієнтам взаємодіяти з сервером через *HTTP*.

2.7 Впровадження інших основних модулів веб-системи

2.7.1 Модуль авторизації

Впровадження модуля авторизації є ключовим для забезпечення безпеки веб-системи *NetworkMonitor*. Авторизація дозволяє контролювати доступ користувачів до функціоналу системи, забезпечуючи таким чином захист від несанкціонованого доступу до чутливих даних мережевого обладнання.

Для проектування авторизації необхідно:

1. Створити форму для введення облікових даних (логін та пароль) користувача.
2. Налаштувати серверну частину для обробки даних авторизації.
3. Зберігати облікові дані в захищеній базі даних, використовуючи шифрування паролів.
4. Впровадити сесії користувачів для забезпечення безперервного досвіду роботи з системою після входу.
5. Забезпечити багаторівневий доступ, що дозволить різним користувачам мати різні рівні доступу до функціоналу системи (наприклад, адміністратори та звичайні користувачі).

2.7.2 Вивід нових даних з мережевого обладнання

На даний момент *NetworkMonitor* вже отримує базові дані з мережевого обладнання за допомогою *SSH*, такі як прафік через інтерфейси, відкриті порти, список підключених пристроїв.

Розширення функціоналу для отримання додаткових даних допоможе забезпечити більш детальний моніторинг та управління мережею. Нові дані, які можна отримати, включають:

1. Статистика пакетів – інформація про кількість переданих та отриманих пакетів, помилки та відхилення.
2. Налаштування *VLAN* – дані про налаштовані віртуальні локальні мережі, що дозволяють краще розуміти структуру мережі.
3. *ARP*-таблиця – інформація про відповідність *IP*-адрес до *MAC*-адрес, що допоможе виявляти проблеми з адресацією в мережі.
4. Лог-файли – логи системи, що можуть містити важливі повідомлення про стан обладнання та мережі.
5. Стан *CPU* та пам'яті – моніторинг використання ресурсів мережевого обладнання для виявлення потенційних проблем з продуктивністю.
6. Статус бездротових мереж – інформація про підключені бездротові пристрої, рівень сигналу, канали тощо.

2.8 Зберігання та відображення отриманих, нових даних

Time-series дані представляють собою набір даних, організованих у хронологічному порядку. В контексті моєї веб-системи *NetworkMonitor*, це можуть бути показники мережевого трафіку, стан ресурсів обладнання (*CPU*, пам'ять), статуси інтерфейсів, тощо. Використання *Time-series* дозволяє ефективно зберігати та аналізувати великі обсяги даних, пов'язаних з часовими мітками, що важливо для моніторингу мережевих ресурсів в реальному часі, як зазначено в джерелі [13].

Для зберігання *Time-series* даних в базі даних, необхідно обрати відповідну базу даних, яка оптимізована для роботи з таким типом даних. Одним із популярних рішень є використання *InfluxDB*, яка спеціалізується на зберіганні та обробці *Time-series* даних. Іншим варіантом може бути *TimescaleDB*, розширення для *PostgreSQL*, яке додає підтримку *Time-series* даних.

При використанні *InfluxDB*, дані можуть бути організовані у вигляді вимірювань (*measurements*), кожне з яких містить поля (*fields*) та теги (*tags*). Наприклад, для зберігання даних про мережевий трафік можна створити вимірювання "*network_traffic*" з полями "*bytes_received*", "*bytes_transmitted*" та

тегами "*interface_name*", "*device_id*". При кожному зборі даних система буде додавати новий запис з часовою міткою, полями та тегами.

Для відображення *Time-series* даних в веб-інтерфейсі *NetworkMonitor* необхідно використовувати відповідні інструменти для візуалізації даних. Популярним рішенням є *Grafana*, яка дозволяє створювати дашборди для візуалізації *Time-series* даних з різних джерел, включаючи *InfluxDB* та *TimescaleDB*.

Висновок до другого розділу

У цьому розглянуто різні аспекти проектування та реалізації системи *NetworkMonitor*, зокрема методи збирання даних та забезпечення їх безпеки. Основна увага приділяється впровадженню та налаштуванню захищених з'єднань для збору даних через *SSH*, *SNMP* та *HTTP*. Також згадується важливість шифрування даних під час їх передачі за допомогою *HTTPS*, що забезпечує конфіденційність і захист від несанкціонованого доступу.

Крім того, розглянуто аспекти аутентифікації та авторизації у системі. Вказано, що поки що ці механізми не реалізовані, оскільки система не має функціоналу, що потребує розподілу доступу між різними користувачами. Планується впровадження системи ідентифікації користувачів, що дозволить використовувати різні методи аутентифікації. Таким чином, забезпечення захисту та конфіденційності даних, а також контроль доступу до системи є важливими аспектами розвитку *NetworkMonitor*.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Обґрунтування вибраних програмних засобів для реалізації проекту

3.1.1 Вибір мов програмування

Основною мовою програмування моєї веб-системи є *C#* – це сучасна об'єктно-орієнтована мова програмування, яка була розроблена компанією *Microsoft*. Вона є частиною платформи *.NET* і відома своєю простотою, надійністю та потужністю. Обрана мною вона була в більшій мірі через те, що вона використовується у зручному фреймворкі *ASP.NET Core*, який я також використовую і, який є основою мого проекту.

ASP.NET Core – це кросплатформенний фреймворк для розробки веб-додатків та веб-сервісів. Він дозволяє створювати високопродуктивні веб-додатки, які можуть працювати на різних операційних системах, таких як *Windows, macOS* та *Linux*. Детальний його опис буде наведено в підрозділі 3.2.

Причини вибору цього фреймворку:

1. Кросплатформенність, отже *ASP.NET Core* може працювати на будь-якій операційній системі, що дозволяє розробникам розгортати додатки на різних платформах.

2. Висока продуктивність, адже він є одним з найшвидших веб-фреймворків, що дозволяє створювати високопродуктивні веб-додатки.

3. Безпека та, як *ASP.NET Core* надає вбудовані засоби для забезпечення безпеки додатків, включаючи захист від атак *CSRF, XSS* та інші.

4. Модульність та розширюваність, бо цей фрейсворк побудований за принципом модульності, що дозволяє розробникам легко додавати або видаляти компоненти відповідно до потреб проекту. Це забезпечує гнучкість та простоту у впровадженні нового функціоналу.

Наступна мова прогрмування це *JavaScript* – динамічна мова програмування, яка широко використовується для створення інтерактивних веб-сторінок. Вона є однією з основних технологій веб-розробки поряд з *HTML* та CSS.

Ця мова програмування була обрана через такі причини, як інтерактивність, підтримка браузерами та велика екосистема. Детальніше описуючи, *JavaScript* дозволяє створювати динамічні та інтерактивні веб-сторінки, покращуючи користувацький досвід. *JavaScript* підтримується всіма сучасними веб-браузерами, що робить його незамінним інструментом для фронтенд розробки. І останнє заключається в тому, що ця мова має велику кількість бібліотек та фреймворків, таких як *React, Angular, Vue.js*, що спрощує розробку складних веб-додатків.

Також окрім *JavaScript* у *front-end* частині я використовую *HTML* та *CSS*.

HTML (*HyperText Markup Language*) – це стандартна мова розмітки для створення веб-сторінок. *HTML* визначає структуру веб-сторінок за допомогою різних елементів, таких як заголовки, параграфи, списки, посилання та інші.

CSS (*Cascading Style Sheets*) – це мова стилів, що використовується для опису зовнішнього вигляду веб-сторінок, створених за допомогою *HTML*. *CSS* дозволяє змінювати кольори, шрифти, розміри, розташування елементів та інші візуальні аспекти веб-сторінки.

Їх я обрав через те, що *HTML* та *CSS* є стандартами для створення та стилізації веб-сторінок і підтримуваними всіма сучасними веб-браузерами, як зазначено в роботі [14]. Вони мають мають простий та зрозумілий синтаксис, що дозволяє швидко створювати веб-сторінки та налаштовувати їх вигляд. Також можна зазначити, що *CSS* дозволяє створювати адаптивні веб-сторінки, які виглядають добре на різних пристроях та екранах різних розмірів.

Далі розберемо в чому ж переваги тих мов програмування та фреймворку, що обрав я над подібними до них.

Почну з того, що *C#* має більш лаконічний синтаксис ніж подібна йому *Java*, та кращу інтеграцію з інструментами розробки від *Microsoft*. *ASP.NET Core*

пропонує вищу продуктивність у порівнянні з деякими популярними *Java* фреймворками.

Хоча *Python* є відмінним вибором для швидкого прототипування, *C#* забезпечує кращу продуктивність та безпеку для веб-додатків. І також *ASP.NET Core* пропонує більше можливостей для налаштування та оптимізації.

Якщо порівнювати з Node.js то знову ж таки *ASP.NET Core* часто показує кращу продуктивність та стабільність у порівнянні з *Node.js*, особливо для великих веб-додатків.

Щодо фронтенду, не двилячись на те, що *Python* може використовуватись для нього, *JavaScript* є стандартом для фронтенду завдяки своїй інтеграції з веб-браузерами.

Також *JavaScript* забезпечує кращу продуктивність та більшу кількість бібліотек для фронтенд розробки ніж подібний йому *Ruby*.

Далі вибір впав на *HTML*, тому що *XHTML* є строгішою його версією, він вимагає суворішого дотримання правил синтаксису. *HTML* пропонує більшу гнучкість та нові функціональні можливості. І виходячи з цього було обрано *CSS* бо, як описано вище, *CSS* використовується для опису зовнішнього вигляду веб-сторінок, створених за допомогою *HTML*.

3.1.2 Вибір середовища розробки та інших програмних засобів

Для розробки веб-системи було обрано *Visual Studio*, що є одним з найпотужніших і найпоширеніших середовищ розробки для платформ *.NET*. *Visual Studio* надає всі необхідні інструменти для розробки, налагодження та тестування програмного забезпечення, згідно із джерелом [15].

Причини вибору саме цього середовища розробки:

1. Інтерфейс *Visual Studio* інтуїтивно зрозумілий та легко налаштовується під потреби розробника.
2. *Visual Studio* надає потужні засоби для налагодження коду, включаючи можливість встановлення точок зупинки, перегляд змінних в режимі реального часу та інше.

3. Це середовище підтримує систему контролю версій *Git*, що дозволяє легко керувати репозиторіями та відстежувати зміни в коді.

4. І звісно підтримка безлічі плагінів та розширень, які можуть значно розширити функціональність середовища розробки.

В причинах вибору середовища було сказано, що воно підтримує *Git*, відповідно для контролю версій та керування кодом було обрано *Git* та *GitHub*. *Git* є розподіленою системою контролю версій, яка дозволяє відстежувати зміни в коді та працювати над проектом в команді. *GitHub* – це веб-сервіс для хостингу *Git*-репозиторіїв з додатковими можливостями для спільної роботи та управління проектами.

Причини вибору цієї системи контролю версій очевидні, бо вона є безкоштовною, найпопулярнішою та дуже зручною, як і веб-сервіс *GitHub*, хоча він являється не повністю безкоштовним.

У проекті використовується *Microsoft SQL Server* (*MSSQL*), який є реляційною системою управління базами даних (СУБД) від компанії Microsoft. *MSSQL* дозволяє зберігати та управляти великими обсягами даних з високою продуктивністю та безпекою, згідно з джерелом [16].

Причини вибору саме такої бази даних наступні:

1. *MSSQL* добре інтегрується з платформою *.NET*, що спрощує розробку додатків на *C#*.
2. Вона також забезпечує високу продуктивність для великих обсягів даних та складних запитів.
3. *MSSQL* надає потужні засоби для забезпечення безпеки даних, включаючи шифрування, управління доступом та аудит.
4. *MSSQL* забезпечує повну підтримку транзакцій, що гарантує цілісність даних у випадку збоїв або помилок.

Для розгортання бази даних я обрав *Docker* – це платформа для автоматизації розгортання додатків в ізольованих контейнерах, як наведено в джерелі [17]. Контейнери *Docker* включають все необхідне для запуску додатку, це може бути код, бібліотеки, залежності, конфігураційні файли, що дозволяє

запускати додаток на будь-якому сервері без необхідності додаткових налаштувань. *Docker* дозволяє швидко розгортати *MSSQL* сервер за допомогою однієї команди. Це значно зменшує час, необхідний для налаштування середовища. І також за допомогою *Docker* можна легко змінювати версії *MSSQL* сервера без необхідності складних процесів оновлення на локальній машині.

3.2 Опис технологій, використаних при проектуванні веб-системи

Для реалізації системи *NetworkMonitor* були обрані сучасні технології та інструменти, які забезпечують високу продуктивність, надійність та зручність розробки. Основними технологіями, використаними в проекті, є *ASP.NET Core* для серверної частини, *Entity Framework Core* для роботи з базою даних, та бібліотека *Renci.SshNet* для забезпечення *SSH*-з'єднань.

Архітектурна діаграма компонентів, що відображає використання технологій в системі наведена на рисунку 3.1.

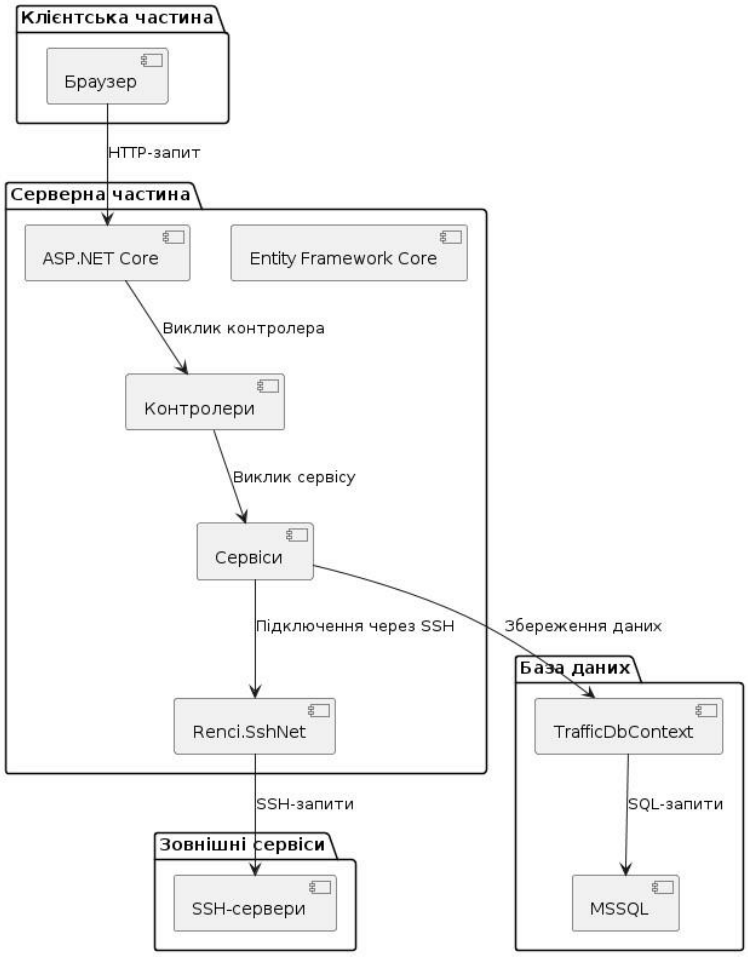


Рис. 3.1 – Архітектурна діаграма компонентів, що відображає використання технологій в системі

3.2.1 *ASP.NET Core*

Підсумовуючи прочитану мною інформацію, із роботи [18] можна зазначити, що *ASP.NET* – це відкрита і безкоштовна веб-платформа, розроблена *Microsoft* для створення динамічних веб-сайтів, веб-додатків та веб-сервісів. Вона є наступником *ASP* і базується на платформі *.NET*, що надає їй значні переваги порівняно з попередником.

Основні можливості *ASP.NET*:

1. Вибір мови програмування, тобто розробники можуть використовувати будь-яку мову *.NET* (*C#, VB.NET, F#*) для написання серверного коду. Це надає велику гнучкість і дозволяє використовувати знання та досвід, набуті в інших проектах на *.NET*.

2. Розширювана модель програмування так, як *ASP.NET* пропонує різноманітні моделі програмування, такі як *Web Forms, MVC, Web Pages* та *Web API*, що дозволяє вибрати найбільш підходящий підхід для конкретного проекту.

3. Багата інфраструктура, а саме *ASP.NET* включає велику кількість вбудованих компонентів та бібліотек, що спрощують розробку типових функцій веб-застосунків (аутентифікація, авторизація, кешування, робота з даними тощо).

4. Невід'ємною частиною є продуктивність та масштабованість, *ASP.NET* оптимізована для високої продуктивності та може бути легко масштабована для обслуговування великої кількості користувачів.

5. І також кросплатформеність так, як *ASP.NET Core*, сучасна версія *ASP.NET*, дозволяє створювати веб-застосунки, які можуть працювати на різних операційних системах (*Windows, Linux, macOS*).

В своєму проекті я використовую фреймфорк *ASP.NET MVC* (*Model-View-Controller*) – це фреймворк, розроблений *Microsoft* для створення веб-застосунків. Він реалізує шаблон проектування *MVC*, який допомагає розділити логіку застосунку на три основні компоненти:

1. Модель (*Model*) відповідає за дані та бізнес-логіку застосунку. Це можуть бути класи, що представляють сутності, а також методи для роботи з цими даними (отримання з бази даних, валідація тощо).

2. Представлення (*View*) відповідає за відображення даних користувачеві. Це *HTML*-сторінки, що містять розмітку та динамічні елементи, які формуються на основі даних з моделі. У *ASP.NET MVC* для створення представлень використовується технологія *Razor*, опис якої буде далі.

3. Контролер (*Controller*) приймає запити від користувача, обробляє їх, взаємодіє з моделлю для отримання або зміни даних, а потім вибирає відповідне представлення для відображення результату користувачеві.

Технологіz *Razor* є частиною фреймворку *ASP.NET MVC* (або *ASP.NET Core MVC*). Razor не є окремою мовою програмування, а скоріше синтаксисом, що дозволяє вбудовувати код *C#* (або *VB.NET*) безпосередньо у *HTML*-сторінки. Це дозволяє створювати динамічні веб-сторінки, де контент може змінюватися залежно від даних або логіки, що виконується на сервері.

3.2.2 *Entity Framework Core*

Згідно із джерелом [19] *Entity Framework Core* (*EF Core*) – це сучасний об'єктно-реляційний маппер (*Object-Relational Mapping, ORM*), в свою чергу *ORM* – це техніка, яка використовується для перетворення даних між несумісними типами систем (об'єктно-орієнтованими програмами та реляційними базами даних). Отже *EF Core* дозволяє *.NET* розробникам працювати з базами даних за допомогою *.NET* об'єктів. *EF Core* є частиною *.NET Core*, а також доступний для *.NET Framework*. В моїй веб-системі *EF Core* використовується для зберігання та отримання даних про мережевий трафік і підключені пристрої.

Основні можливості *Entity Framework Core*:

1. Мапінг об'єктів до реляційних баз даних: *EF Core* дозволяє розробникам мапити класи *C#* до таблиць в базі даних і навпаки, тобто мапити рядки з бази даних до об'єктів *C#*.

2. *LINQ*-запити: Розробники можуть писати запити до бази даних, використовуючи мову запитів *LINQ*, яка дозволяє здійснювати запити до колекцій об'єктів у стилі *SQL*.

3. Міграції: *EF Core* підтримує міграції, які допомагають керувати змінами в схемі бази даних, зберігаючи історію змін.

Підтримка багатьох СУБД: *EF Core* підтримує різні системи управління базами даних (СУБД), включаючи *Microsoft SQL Server*, *SQLite*, *PostgreSQL*, *MySQL* та інші.

3.2.3 Renci.SshNet

Renci.SshNet – це бібліотека для роботи з протоколом *SSH* (*Secure Shell*) у *.NET* середовищі. Вона забезпечує інтерфейс для безпечного підключення до віддалених серверів та виконання на них команд, а також для передачі файлів через *SFTP* (*SSH File Transfer Protocol*) і *SCP* (*Secure Copy Protocol*).

У своїй веб-системі *NetworkMonitor* я використовую бібліотеку *Renci.SshNet* для взаємодії з віддаленими серверами. Конкретно, в коді проекту *Renci.SshNet* дозволяє виконувати команди на віддалених серверах для збору даних про мережевий трафік, відкриті порти та підключені пристрої.

Основні можливості *Renci.SshNet*:

- підключення по *SSH* – забезпечує можливість встановлення безпечного з'єднання з віддаленими серверами через *SSH*;
- виконання команд – дозволяє виконувати команди на віддаленому сервері та отримувати їх результати;
- *SFTP* та *SCP* – підтримує передачу файлів між локальним та віддаленим серверами за допомогою *SFTP* та *SCP*;
- тунелювання – підтримує перенаправлення портів через *SSH*;
- інтерактивні сесії – дозволяє створювати інтерактивні сесії з віддаленим сервером.

3.3 Реалізація основних компонентів системи

3.3.1 Модуль збору даних

Модуль збору даних відповідає за підключення до віддалених серверів через *SSH* для збору мережевої статистики, даних про порти та підключені пристрої. Цей модуль реалізує основну функціональність збирання даних у реальному часі. Далі буде описано які класи та методи відносяться до цього модулю.

Клас *SshService* забезпечує підключення до віддалених серверів через *SSH* та виконує необхідні команди для збору даних.

Він має декілька методів, перший із них це *GetOverallTrafficData* він підключається до віддаленого сервера, виконує команду *netstat -e* та повертає загальну мережеву статистику (лістинг 3.1).

Лістинг 3.1. Метод *GetOverallTrafficData*

```
public NetworkStatistics GetOverallTrafficData()
{
    var networkStatistics = new NetworkStatistics();

    using var client = new SshClient(_config.Host, _config.Username,
_config.Password);
    client.Connect();

    var command = client.CreateCommand("netstat -e");
    var result = command.Execute();

    client.Disconnect();

    ParseNetworkStatistics(result, networkStatistics);
    return networkStatistics;
}
```

Наступний метод це *GetPortData* він підключається до віддаленого сервера, виконує команду *netstat -anob* та повертає дані про порти. Реалізація цього методу дуже схожа на реалізацію минулого, відмінність лише вотримуваних даних. Відмінність реалізації цього методу показана в лістингу 3.2.

Лістинг 3.2. Відмінність реалізації методу *GetPortData*

```
    portDataList = ParsePortData(result);

    return portDataList;
}
```

Останній метод цього класу це *GetConnectedDevices* підключається до віддаленого сервера, виконує команду *arp -a* та повертає дані про підключені пристрої. Як і у випадку із минулим методом відмінність реалізації лише у отримуваних даних, показаних у лістингу 3.3.

Лістинг 3.3. Відмінність реалізації методу *GetConnectedDevices*

```
    connectedDevicesList = ParseConnectedDevices(result);

    return connectedDevicesList;
}
```

Після кожно із цих методів також є ще по одному методу, вони відповідають за аналіз отриманих даних за допомогою потужного інструменту для роботи з текстом *Regex*. Приклад реалізації одного такого методу *ParseConnectedDevices*, який аналізує дані отримані методом *GetConnectedDevices* приведений у лістингу 3.4.

Лістинг 3.4. Приклад реалізації методу аналізу отриманих даних

```
private List<ConnectedDevice> ParseConnectedDevices(string data)
{
    var connectedDevicesList = new List<ConnectedDevice>();

    var regex = new
Regex(@"(?<ip>(?:\d{1,3}\.){3}\d{1,3})\s+(?<mac>(?:[0-9a-fA-F]{2}-){5}[0-
9a-fA-F]{2})\s+(?<type>\S+)");

    var matches = regex.Matches(data);

    foreach (Match match in matches)
    {
        if (match.Success)
        {
            connectedDevicesList.Add(new ConnectedDevice
            {
                IpAddress = match.Groups["ip"].Value,
                MacAddress = match.Groups["mac"].Value,
                Type = match.Groups["type"].Value
```

```
            });
        }
    }

    return connectedDevicesList;
}
```

Клас *NetworkStatistics* використовується для зберігання зібраних даних про мережеву статистику, включаючи байти отримані та відправлені, кількість пакетів тощо. В лістингу 3.5 показано, як реалізований цей клас та, які саме дані зберігає.

Лістинг 3.5. Приклад реалізації класу *NetworkStatistics*

```
public class NetworkStatistics
{

    public int Id { get; set; }
    public long BytesSent { get; set; }
    public long BytesReceived { get; set; }
    public int UnicastPacketsReceived { get; set; }
    public int UnicastPacketsSent { get; set; }
    public int NonUnicastPacketsReceived { get; set; }
    public int NonUnicastPacketsSent { get; set; }
    public int DiscardsReceived { get; set; }
    public int DiscardsSent { get; set; }
    public int ErrorsReceived { get; set; }
    public int ErrorsSent { get; set; }
    public int UnknownProtocolsReceived { get; set; }

    public DateTime Timestamp { get; set; }
}
```

Клас *PortData* використовується для зберігання інформації про відкриті порти, включаючи протокол, локальну та зовнішню адресу, стан порту тощо. В лістингу 3.6 приведено приклад реалізації цього класу.

Лістинг 3.6. риклад реалізації класу *PortData*

```
public class PortData
{
    public string Protocol { get; set; }
    public string LocalAddress { get; set; }
    public int LocalPort { get; set; }
    public string ForeignAddress { get; set; }
    public int ForeignPort { get; set; }
    public string State { get; set; }
    public int PID { get; set; }
```

```
    public string ProcessName { get; set; }
}
```

Клас *ConnectedDevice* використовується для зберігання інформації про підключені пристрої, включаючи *IP*-адресу, *MAC*-адресу та тип пристрою. В лістингу 3.7 приведено приклад реалізації цього класу.

Лістинг 3.7. Приклад реалізації класу *ConnectedDevice*

```
public class ConnectedDevice
{

    public string IpAddress { get; set; }
    public string MacAddress { get; set; }
    public string Type { get; set; }
}
```

3.3.2 Модуль аналізу даних

Модуль аналізу даних відповідає за обробку, фільтрацію та зберігання зібраних даних у базі даних. Цей модуль виконує основні операції з даними, такі як збереження нових записів, оновлення існуючих, а також виконання запитів для отримання необхідної інформації.

Клас *TrafficDbContext* є контекстом бази даних, що використовується для взаємодії з *MSSQL* сервером за допомогою *Entity Framework Core*. Він містить налаштування для створення таблиць та збереження даних. Приклад реалізації цього класу приведений в лістингу 3.8.

Лістинг 3.8. Приклад реалізації класу *TrafficDbContext*

```
using Microsoft.EntityFrameworkCore;
using NetworkMonitor.Models;

namespace NetworkMonitor.Persistence;

public sealed class TrafficDbContext : DbContext
{
    public DbSet<NetworkStatistics> NetworkStatistics { get; set; }

    public TrafficDbContext(DbContextOptions<TrafficDbContext> options)
        : base(options)
    {
```

```
            Database.EnsureCreated();
        }
    }
```

3.3.3 Модуль візуалізації даних

Модуль візуалізації даних відповідає за надання користувачам зручного інтерфейсу для перегляду зібраної та обробленої інформації. Цей модуль реалізує функціональність відображення даних у вигляді таблиць, графіків та інших візуальних елементів.

Клас *HomeController* забезпечує завантаження основних сторінок веб-додатку.

Метод *Index* завантажує головну сторінку з інформацією про мережеву статистику.

Метод *Privacy* завантажує сторінку з політикою конфіденційності.

Метод *Error* відображає інформацію про помилки.

Реалізація цього класу та його методів приведена в лістингу 3.9.

Лістинг 3.9. Приклад реалізації класу *HomeController* та його методів

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

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

    public IActionResult Index()
    {
        return View();
    }

    public IActionResult Privacy()
    {
        return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]
    public IActionResult Error()
    {
```

```
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id
?? HttpContext.TraceIdentifier });
    }
}
```

Клас *TrafficController* відповідає за отримання та відправку даних про мережевий трафік, порти та підключені пристрої до клієнта. Далі буде опис трьох основних його методів.

В лістингу 3.10 буде наведено реалізацію цих методів.

Метод *GetOverallTrafficData* відповідає за отримання загальних даних про мережевий трафік та їх відправку до клієнта.

Метод *GetPortData* відповідає за отримання даних про порти та їх відправку до клієнта.

Метод *GetConnectedDevices* відповідає за отримання даних про підключені пристрої та їх відправку до клієнта.

Лістинг 3.10. Реалізація основних методів класу *TrafficController*

```
[HttpGet]
public IActionResult GetOverallTrafficData()
{
    var data = _context.NetworkStatistics
        .OrderByDescending(x => x.Timestamp)
        .Take(1000)
        .ToList();

    return PartialView("OverallTrafficDataPartial", data);
}

public IActionResult GetPortData()
{
    var portData = _sshService.GetPortData();
    return PartialView("PortDataPartial", portData);
}

public IActionResult GetConnectedDevices()
{
    var connectedDevices = _sshService.GetConnectedDevices();
    return PartialView("ConnectedDevicesPartial", connectedDevices);
}
```

3.3.4 Види (*Views*)

ConnectedDevicesPartial.cshtml відповідає за відображення інформації про підключені пристрої. В лістингу 3.11 представлено реалізацію цього виду.

Лістинг 3.11. Реалізація виду *ConnectedDevicesPartial.cshtml*

```
@model List<ConnectedDevice>

 <table>
    <thead>
    <tr>
        <th>IP Address</th>
        <th>MAC Address</th>
        <th>Type</th>
    </tr>
    </thead>
    <tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td>@item.IpAddress</td>
            <td>@item.MacAddress</td>
            <td>@item.Type</td>
        </tr>
    }
    </tbody>
</table>
```

OverallTrafficDataPartial.cshtml відповідає за відображення загальної інформації про мережевий трафік. Його реалізація досить велика, тому її можна буде переглянути в додатку Б.

PortDataPartial.cshtml відповідає за відображення даних про порти. В лістингу 3.12 представлено реалізацію цього виду.

Лістинг 3.12. Реалізація виду *PortDataPartial.cshtml*

```
@model IEnumerable<PortData>

<table>
    <thead>
    <tr>
        <th>Protocol</th>
        <th>Local Address</th>
        <th>Local Port</th>
        <th>Foreign Address</th>
        <th>Foreign Port</th>
```

```
            <th>State</th>
            <th>PID</th>
            <th>Process Name</th>
        </tr>
        </thead>
        <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.Protocol</td>
                <td>@item.LocalAddress</td>
                <td>@item.LocalPort</td>
                <td>@item.ForeignAddress</td>
                <td>@item.ForeignPort</td>
                <td>@item.State</td>
                <td>@item.PID</td>
                <td>@item.ProcessName</td>
            </tr>
        }
        </tbody>
</table>
```

3.3.5 Взаємодія між компонентами

Компоненти веб-системи тісно взаємодіють між собою для забезпечення збору, обробки та візуалізації даних, наступним чином:

1. Збір даних – *SshService* підключається до віддалених серверів і збирає необхідні дані за допомогою методів *GetOverallTrafficData*, *GetPortData* та *GetConnectedDevices*.

2. Обробка та збереження даних – зібрані дані обробляються і зберігаються в базі даних за допомогою *TrafficDbContext*.

3. Візуалізація даних – *TrafficController* отримує дані з бази даних і відправляє їх до клієнта у вигляді *JSON*. Види (*Views*) *ConnectedDevicesPartial.cshtml*, *TrafficData.cshtml* та *PortData.cshtml* відображають дані у зручному для користувача вигляді. Ці види забезпечують інтерактивні та динамічні графіки та таблиці, що полегшує аналіз і моніторинг мережевого трафіку та підключених пристроїв в реальному часі.

Діаграма взаємоїді цих компонентів представлена на рисунку 3.2.

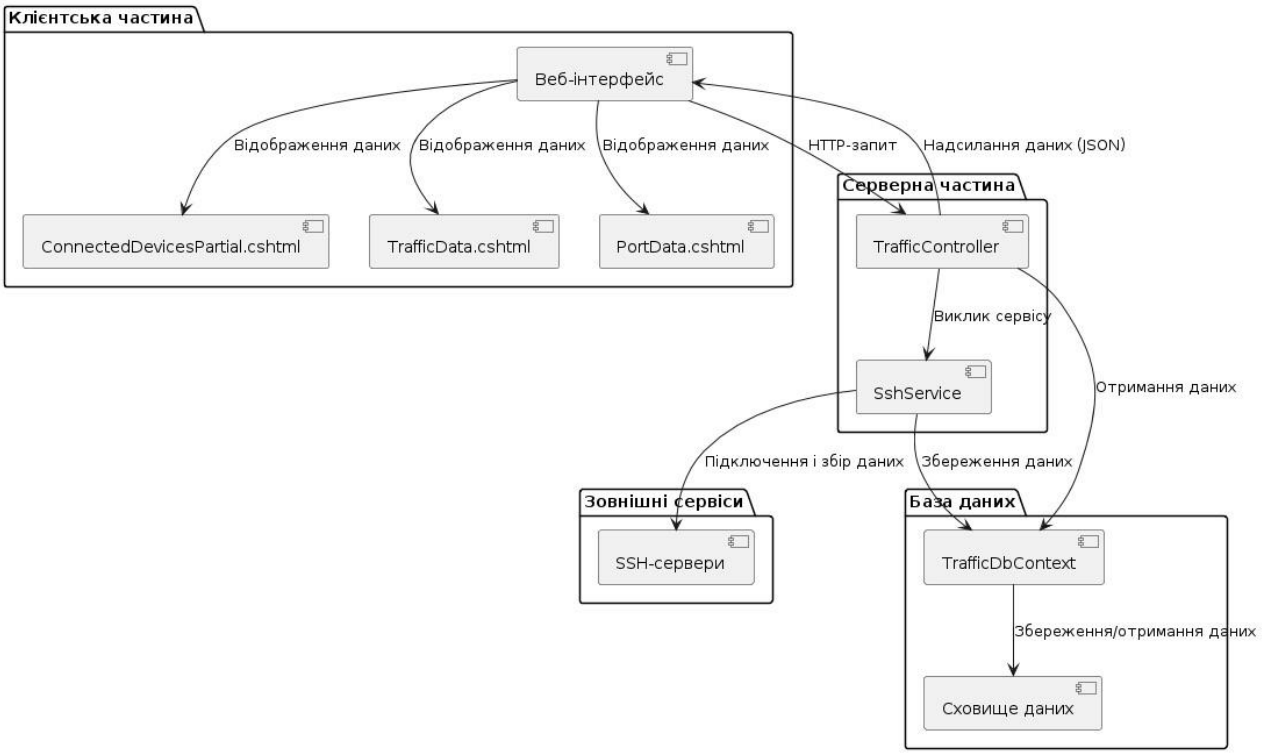


Рис. 3.2 – Діаграма взаємодії компонентів

3.4 Опис основного файлу веб-системи

Файл *Program.cs* є основним входом в моїй веб-системі. Він налаштовує та запускає *ASP.NET Core* додаток, відповідає за конфігурацію сервісів, налаштування залежностей та побудову *HTTP*-конвеєра. Реалізація цього файлу наведена в лістингу 3.13.

Лістинг 3.13. Реалізація основного файлу *Program.cs*

```
using Microsoft.EntityFrameworkCore;
using NetworkMonitor.Configurations;
using NetworkMonitor.Extensions;
using NetworkMonitor.Persistence;
using NetworkMonitor.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<TrafficDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultCo
nnection")));
```

```
builder.Services.AddTransient<SshService>();
builder.Services.AddHostedService<UpdateTrafficDataService>();

builder.Services.AddConfiguration<SshConfiguration>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

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

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

Перше, що знаходиться в цьому файлі це звісно підключення потрібних просторів імен, окрім того там реалізовано наступне:

1. Створення об'єкта *WebApplicationBuilder*, який використовується для налаштування веб-додатка.
2. Додавання сервісів до контейнера:
   a) додавання підтримки контролерів з відображеннями (*MVC*);
   b) налаштування *TrafficDbContext* для використання *SQL Server* з рядком підключення, що зберігається в конфігурації;
   c) реєстрація *SshService* з короткотривалим життєвим циклом (*transient*) та *UpdateTrafficDataService* як фонового сервісу;
   d) додавання конфігурації для *SshConfiguration*;
3. Побудова додатка.
4. Налаштування *HTTP*-конвеєра:
   a) Налаштування обробки помилок та *HSTS* (*HTTP Strict Transport Security*) для захисту додатка в середовищі продакшн.

b) Налаштування проміжного програмного забезпечення (*middleware*):

c) Налаштування маршрутів для контролерів за замовчуванням.

5. Запуск додатка.

Як було описано вище в цьому переліку в другому пункті відбувається налаштування класу *TrafficDbContext* для використання бази даних, і в наступному файлі *appsettings.json* відбувається створення нової бази даних, у разі, якщо не підключено вже створену. Реалізація даного файлу представлено в лістингу 3.14.

Лістинг 3.14. Реалізація файлу *appsettings.json*

```
"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
},
"AllowedHosts": "*",
"ConnectionStrings": {
  "DefaultConnection": "server=localhost;database=NetworkMonitor3;user
id=sa;password=MyPass@word;TrustServerCertificate=True;"
},
```

3.5 Розгортання та налаштування веб-системи

Першим кроком потрібно завантажити та відкрити *Docker*, для розгортання *MSSQL* сервера, тому що так досить зручно, адже *MSSQL* сервер, налаштований в контейнері, буде працювати однаково на локальному комп'ютері розробника, тестовому сервері та в продуктивному середовищі. Щоб підключитися до *MSSQL* потрібно відкрити командний рядок в операційній системі та ввести туди наступну одну команду “docker run -e "ACCEPT_EULA=Y" –e "MSSQL_SA_PASSWORD=MyPass@word" -p 1433:1433 --name mssql --hostname mssql -d mcr.microsoft.com/mssql/server:2022-latest”. Вводити без двох крайніх лапок. Далі в *Docker* з’явиться mssql сервер і його потрібно запустити, як це показано на рисунку 3.15.

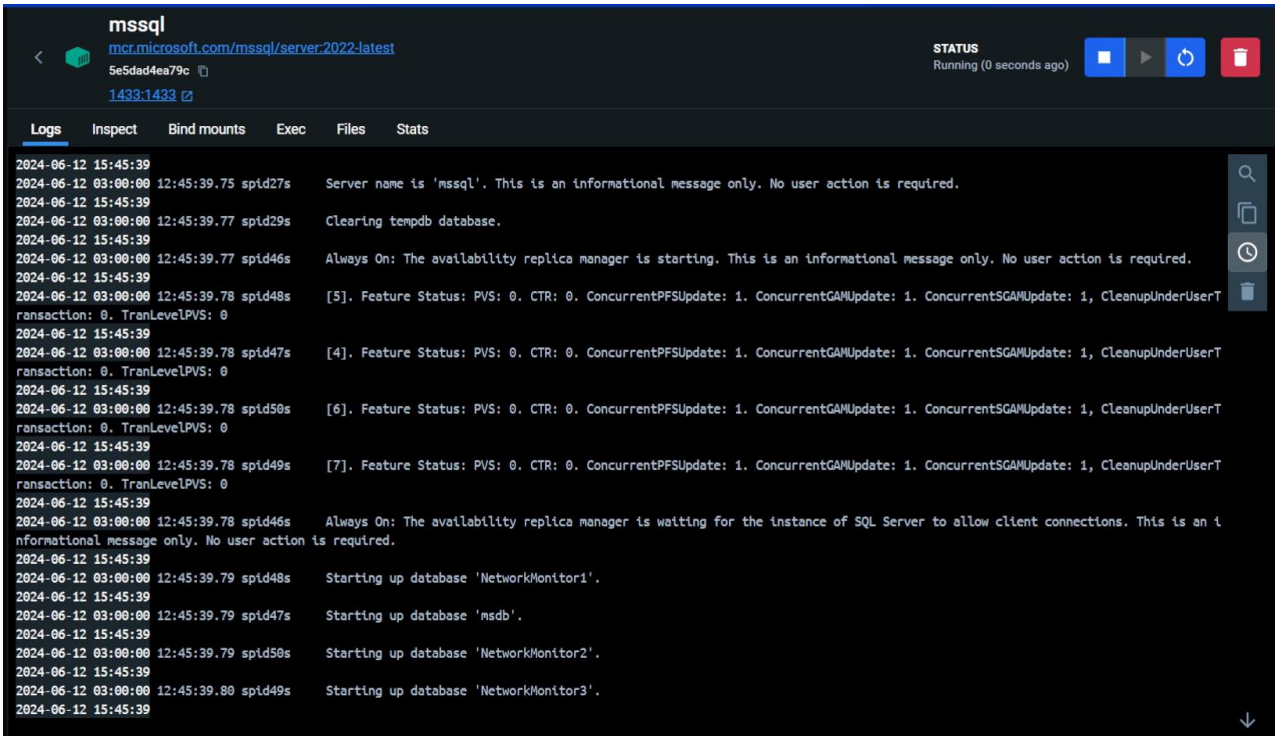


Рис. 3.15 – Запуск mssql серверу в *Docker*

Після того переходимо у *Visual Studio* вибираємо файл *Program.cs* та запускаємо його, як показано на рисунку 3.16.

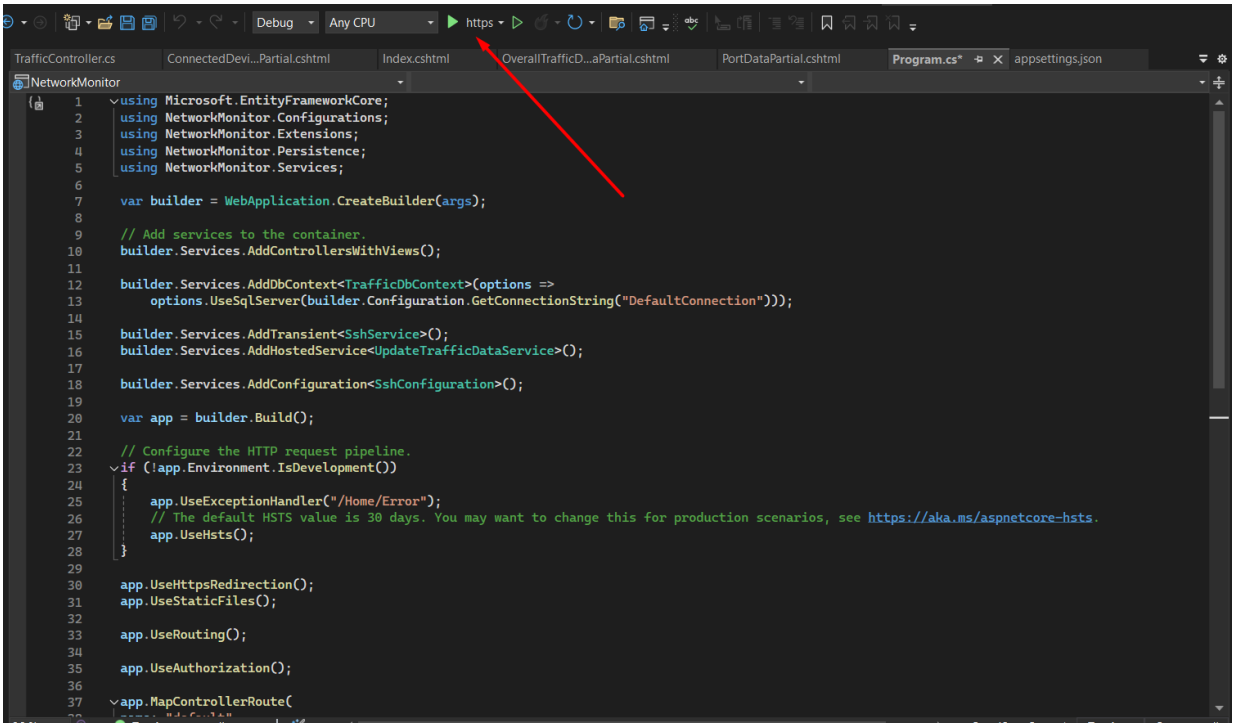


Рис. 3.16 – Запуск файлу *Program.cs* у *Visual Studio*

Після чого відкривається початкова сторінка веб-системи і для перегляду трафіку та інших даних переходимо у розділ *Traffic*, як це зображено на рисунку 3.17.

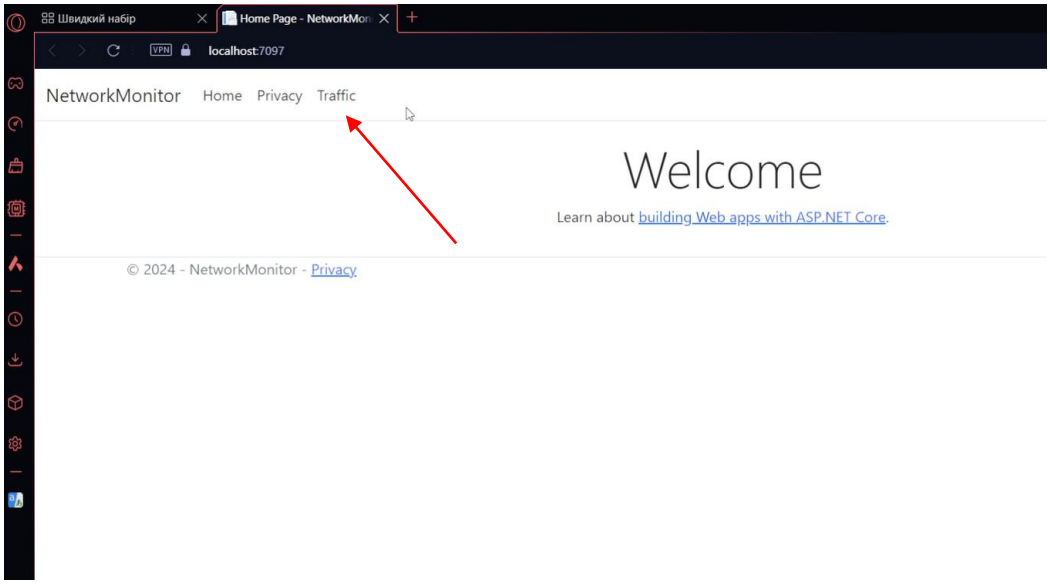


Рис. 3.17 – Початкова сторінка веб-системи

Перейшовши у розділ *Traffic*, перше, що ми бачимо це графік трафіку, який проходить через порти маршрутизатора, як це зображено на рисунку 3.18. За його відображення, як раз і відповідає, описаний вище, файл *OverallTrafficDataPartial.cshtml*.

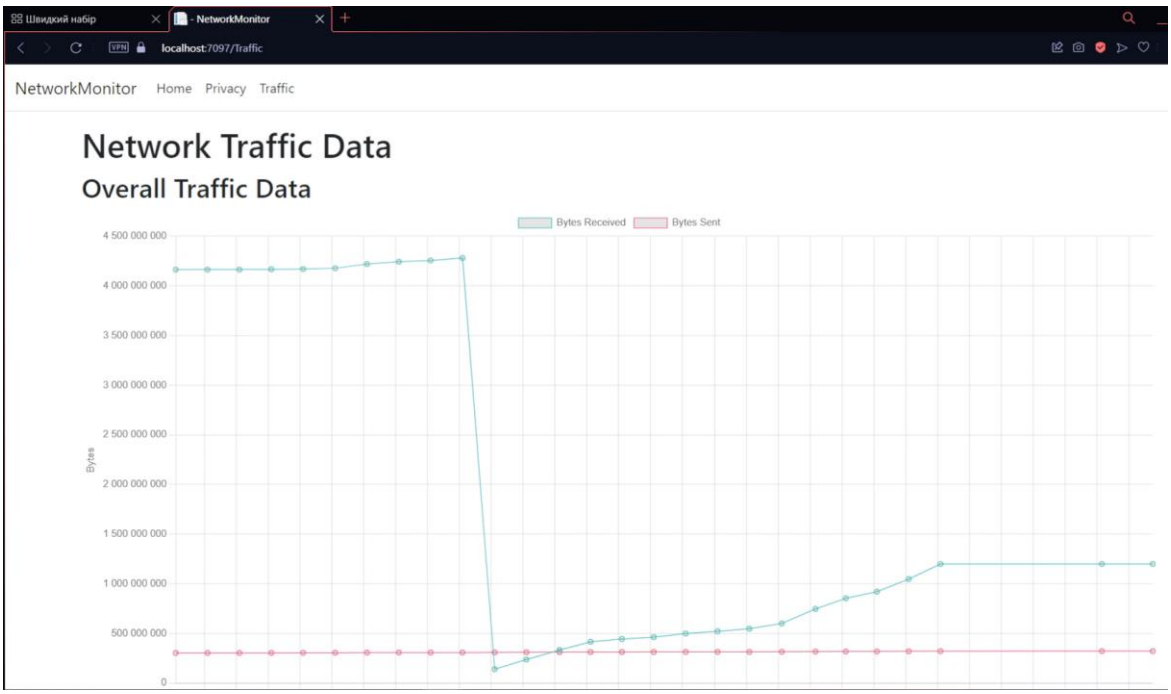
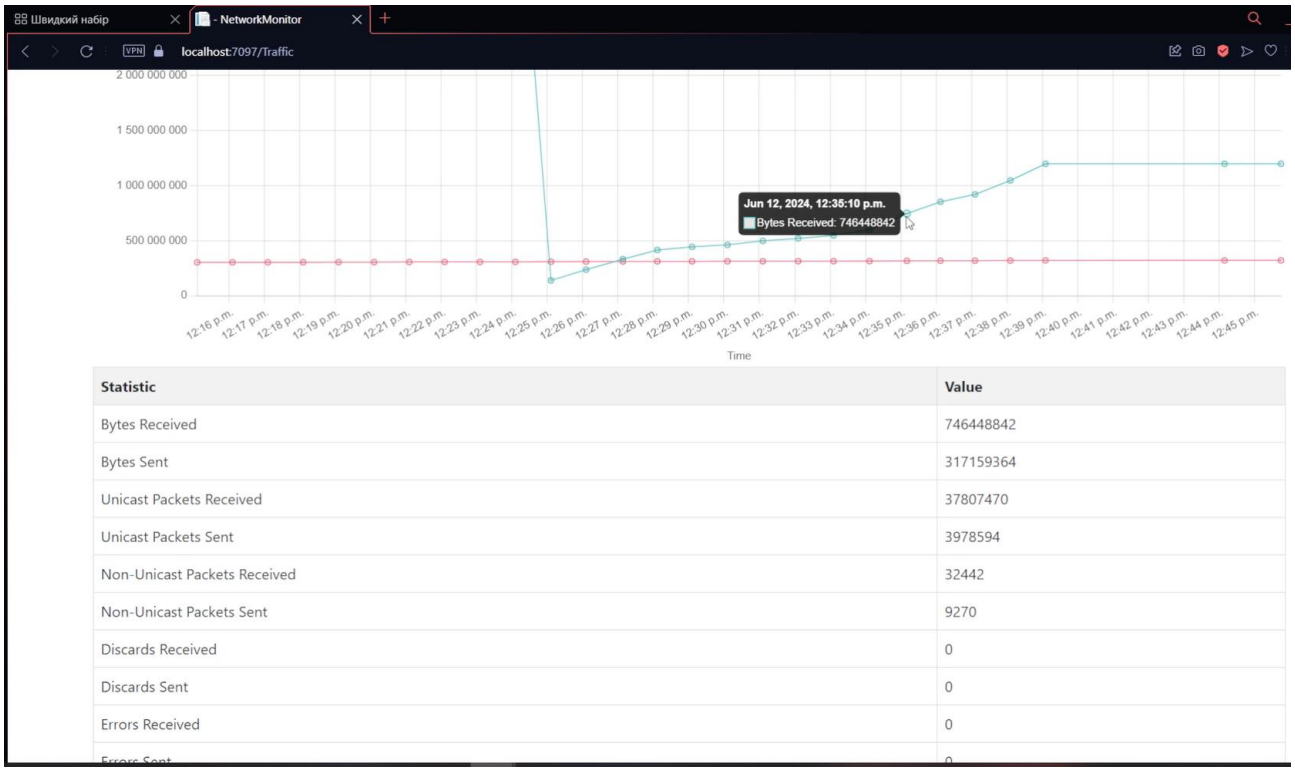


Рис. 3.18 – Частина веб-системи, що відображає графік трафіку із портів

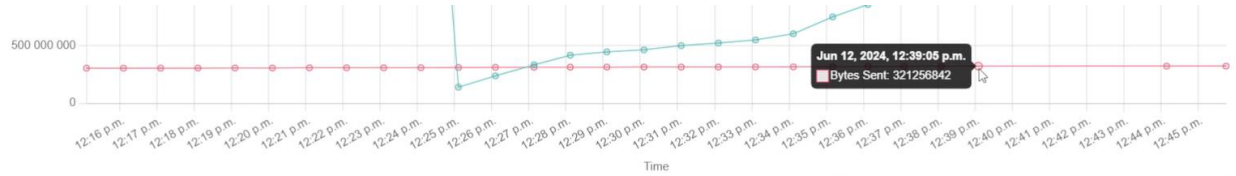
Продовження цієї частини показує дані про трафік у вибраний момент на графіку, що можна побачити на рисунку 3.19.



Statistic	Value
Bytes Received	746448842
Bytes Sent	317159364
Unicast Packets Received	37807470
Unicast Packets Sent	3978594
Non-Unicast Packets Received	32442
Non-Unicast Packets Sent	9270
Discards Received	0
Discards Sent	0
Errors Received	0
Errors Sent	0

Рис. 3.19 – Продовження першої частини веб-системи, що показує дані трафіку у вибраний момент

Наступний рисунок 3.20 відображає вибрані в потрібний момент дані, що були надіслані.

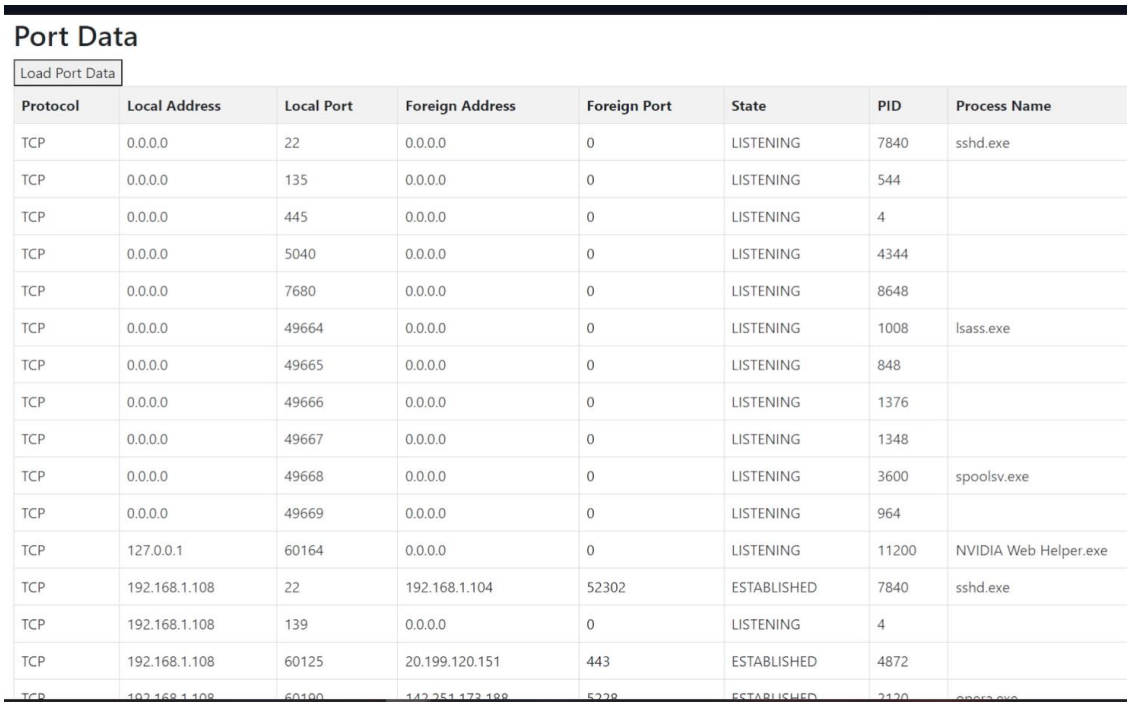


Statistic	Value
Bytes Received	1198737776
Bytes Sent	321256842
Unicast Packets Received	38159598
Unicast Packets Sent	4021194
Non-Unicast Packets Received	32832
Non-Unicast Packets Sent	9324
Discards Received	0

Рис. 3.20 – Відображення надісланих даних

Наступна частина показує дані про порти маршрутизатора, які завантажуються при натисканні на кнопку, як це зображено на рисунку 3.21. За

відображення цієї частини відповідає, описаний вище файл, *PortDataPartial.cshtml*.



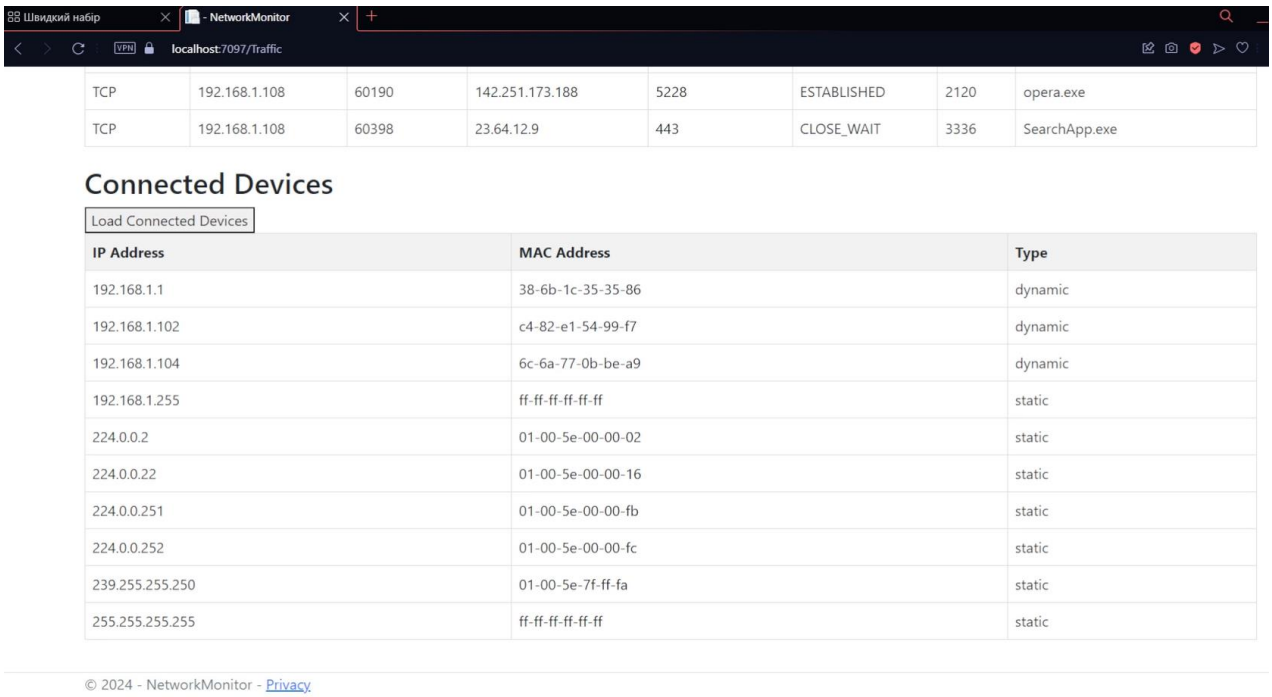
Port Data

Load Port Data

Protocol	Local Address	Local Port	Foreign Address	Foreign Port	State	PID	Process Name
TCP	0.0.0.0	22	0.0.0.0	0	LISTENING	7840	sshd.exe
TCP	0.0.0.0	135	0.0.0.0	0	LISTENING	544	
TCP	0.0.0.0	445	0.0.0.0	0	LISTENING	4	
TCP	0.0.0.0	5040	0.0.0.0	0	LISTENING	4344	
TCP	0.0.0.0	7680	0.0.0.0	0	LISTENING	8648	
TCP	0.0.0.0	49664	0.0.0.0	0	LISTENING	1008	lsass.exe
TCP	0.0.0.0	49665	0.0.0.0	0	LISTENING	848	
TCP	0.0.0.0	49666	0.0.0.0	0	LISTENING	1376	
TCP	0.0.0.0	49667	0.0.0.0	0	LISTENING	1348	
TCP	0.0.0.0	49668	0.0.0.0	0	LISTENING	3600	spoolsv.exe
TCP	0.0.0.0	49669	0.0.0.0	0	LISTENING	964	
TCP	127.0.0.1	60164	0.0.0.0	0	LISTENING	11200	NVIDIA Web Helper.exe
TCP	192.168.1.108	22	192.168.1.104	52302	ESTABLISHED	7840	sshd.exe
TCP	192.168.1.108	139	0.0.0.0	0	LISTENING	4	
TCP	192.168.1.108	60125	20.199.120.151	443	ESTABLISHED	4872	

Рис. 3.21 – Частина веб-системи, що відображає дані про самі порти

І остання частина веб-системи показує дані про підключені пристрої до маршрутизатора, які також завантажуються при натисканні на кнопку, як це зображено на рисунке 3.22. За відображення цієї частини відповідає, описаний вище, файл *ConnectedDevicesPartial.cshtml*.



Швидкий набір | NetworkMonitor | localhost:7097/Traffic

TCP	192.168.1.108	60190	142.251.173.188	5228	ESTABLISHED	2120	opera.exe
TCP	192.168.1.108	60398	23.64.12.9	443	CLOSE_WAIT	3336	SearchApp.exe

Connected Devices

Load Connected Devices

IP Address	MAC Address	Type
192.168.1.1	38-6b-1c-35-35-86	dynamic
192.168.1.102	c4-82-e1-54-99-f7	dynamic
192.168.1.104	6c-6a-77-0b-be-a9	dynamic
192.168.1.255	ff-ff-ff-ff-ff-ff	static
224.0.0.2	01-00-5e-00-00-02	static
224.0.0.22	01-00-5e-00-00-16	static
224.0.0.251	01-00-5e-00-00-fb	static
224.0.0.252	01-00-5e-00-00-fc	static
239.255.255.250	01-00-5e-7f-ff-fa	static
255.255.255.255	ff-ff-ff-ff-ff-ff	static

© 2024 - NetworkMonitor - Privacy

Рис. 3.22 – Частина веб-системи, що відображає дані про підключені пристрої

3.6 Реалізація інших методів збору даних з мережевого обладнання

Щоб реалізувати отримання даних за допомогою *SNMP* потрбіно впровадити *SNMP*-менеджер в мою веб-систему *NetworkMonitor* для запитів до *SNMP*-агентів на роутерах. Це передбачає налаштування бібліотек для роботи зі *SNMP*, таких як *Net-SNMP*. Далі потрібно завантаження відповідних *MIB*-файлів для вашого роутера. Це дозволить менеджеру розуміти дані, які надаються агентом. Наступним кроком буде написання логіки для періодичного опитування *SNMP*-агентів, використовуючи *SNMP*-запити типу *GET* для отримання інформації про трафік, порти та підключені пристрої. І останнє це обробка отриманих даних і їх інтеграція в існуючі структури даних *NetworkMonitor*.

Щоб реалізувати отримання даних за допомогою *HTTP* потрібно дослідити документації роутера для визначення доступних *HTTP API*, які дозволяють отримувати необхідні дані. Далі йде інтеграція *HTTP*-клієнта в *NetworkMonitor* для відправки запитів до роутера. Наступний крок це впровадження механізмів аутентифікації, якщо *API* роутера вимагає авторизації (наприклад, через *OAuth*). Написання логіки для періодичного відправлення *HTTP*-запитів до роутера для отримання інформації про трафік, порти та підключені пристрої. І останнє це також обробка отриманих *JSON* або *XML* даних та їх інтеграція в існуючі структури даних *NetworkMonitor*.

Для впровадження *SNMP* та *HTTP* методів в *NetworkMonitor* необхідно:

1. Додати нові модулі та класи, які будуть відповідати за збір даних через *SNMP* та *HTTP*.
2. Оновити конфігураційний файл для підтримки параметрів *SNMP* (наприклад, *IP*-адреси агентів, ком'юніті строки) та *HTTP* (наприклад, *URL API*, ключі аутентифікації).
3. Додати можливість вибору способу збору даних (*SSH*, *SNMP*, *HTTP*) в інтерфейсі користувача.
4. Провести всебічне тестування для впевненості у коректній роботі нових методів збору даних.

3.7 Реалізація інших основних модулів веб-системи

3.7.1 Модуль авторизації

Для реалізації модуля авторизації в моїй веб-системі необхідно виконати наступні кроки:

1. Створення форми для введення облікових даних. А саме реалізувати форму для введення логіну та пароля користувача на клієнтській стороні. Це можна зробити за допомогою *Razor Pages* або *MVC Views* у *.NET*, використовуючи *HTML* та *CSS* для створення інтуїтивно зрозумілого інтерфейсу.

2. Налаштування серверної частини для обробки даних авторизації. А саме використати популярні бібліотеки для роботи з аутентифікацією, такі як *Microsoft.AspNetCore.Identity*. Ця бібліотека забезпечує готові рішення для реєстрації, входу, виходу та управління обліковими записами користувачів. Також налаштувати сервіси аутентифікації у *Startup.cs* файлі.

3. Зберігання облікових даних у захищеній базі даних. Тобто використовувати *Entity Framework Core* для взаємодії з базою даних. Дані користувачів зберігаються у таблицях, створених *Identity*. Забезпечити шифрування паролів за допомогою вбудованих функцій *Identity*, які використовують *PasswordHasher*.

4. Впровадження сесій користувачів, для цього потрібно налаштувати механізм сесій для забезпечення безперервного досвіду роботи користувачів після входу. Це можна зробити за допомогою *Microsoft.AspNetCore.Session*.

5. Забезпечення багаторівневого доступу. А саме реалізувати багаторівневу авторизацію, що дозволяє різним користувачам мати різні рівні доступу до функціоналу системи. Для цього можна використовувати *RoleManager* та *UserManager* з *Microsoft.AspNetCore.Identity* для управління ролями та їх призначення користувачам.

3.7.2 Реалізація виводу нових даних з мережевого обладнання

В підрозділі 2.7.2 вже було описано, які нові дані можна отримувати з мережевого обладнання, в цьому ж підрозділі перейдемо до опису їх реалізації.

Для впровадження отримання нових даних, необхідно розширити існуючий інтерфейс збору даних для підтримки нових типів інформації. Додати відповідні запити до мережевого обладнання через *SSH, SNMP* або *HTTP API*. Оновити базу даних для зберігання нових типів даних. І також модифікувати інтерфейс користувача для відображення нових даних, забезпечуючи зручний доступ до детальної інформації.

3.7.3 Реалізація зберігання отриманих, нових даних

В підрозділі 2.8 було описано про майбутнє впровадження Time-series даних для моєї веб-системи, в цьому підрозділі перейдем до їх майбутньої реалізації.

Для інтеграції *Time-series* даних в існуючу веб-систему *NetworkMonitor*, необхідно реалізувати кілька ключових компонентів:

1. Збір даних, тобто модуль збору даних повинен періодично запитувати дані з мережевого обладнання (через *SSH, SNMP* або *HTTP*) і зберігати їх у *Time-series* базі даних. Для цього можна використовувати крон-завдання або відповідні тригери в системі.

2. Збереження даних в базі даних з відповідними часовими мітками, після збору даних. Наприклад, дані про використання *CPU* можна зберігати в вимірюванні "*cpu_usage*" з полями "*usage_percentage*" та тегами "*device_id*", "*core*".

Для аналізу *Time-series* даних можна використовувати запити до бази даних, які дозволяють виконувати агрегацію, обчислення середніх значень, пошук аномалій та інші аналітичні операції. Це дозволить швидко отримувати необхідну інформацію про стан мережі та виявляти проблеми.

3.7.4 Реалізація відображення *Time-series* даних

Інтеграція *Grafana* в веб-систему *NetworkMonitor* дозволить створювати графіки та діаграми для відображення динаміки мережевого трафіку, використання ресурсів, статусів інтерфейсів та інших показників. Користувачі зможуть налаштовувати власні дашборди, вибирати потрібні інтервали часу, додавати алерти для відстеження критичних показників.

Щоб реалізувати відображення даних, необхідно налаштувати *Grafana* для підключення до *Time-series* бази даних та створити відповідні дашборди. Веб-інтерфейс *NetworkMonitor* може бути інтегрований з *Grafana* через вбудовані фрейми або *API*-виклики, що дозволить користувачам легко переглядати та аналізувати дані безпосередньо в системі.

Таким чином, впровадження *Time-series* даних у веб-систему *NetworkMonitor* дозволить ефективно зберігати, обробляти та візуалізувати великі обсяги мережевих даних, забезпечуючи більш детальний і гнучкий моніторинг мережевих ресурсів.

Висновок до третього розділу

У третьому розділі дипломного проекту розглянуто практичну реалізацію та результати дослідження. Основною мовою програмування для розробки веб-системи обрано *C#* у фреймворку *ASP.NET Core*, який забезпечує високу продуктивність, безпеку та кросплатформеність. Також використовувалися *JavaScript* для інтерактивності, *HTML* та *CSS* для структури та стилізації веб-сторінок. Для середовища розробки обрано *Visual Studio*, а для контролю версій – *Git* та *GitHub*. В якості СУБД використовувався *Microsoft SQL Server*.

Окрім цього, у проекті застосовані сучасні технології, такі як *Entity Framework Core* для роботи з базою даних та бібліотека *Renci.SshNet* для забезпечення *SSH*-з’єднань. Основні компоненти системи включають модуль збору даних, що відповідає за підключення до віддалених серверів і збір мережевої статистики в реальному часі.

РОЗДІЛ 4
ТЕХНІКО-ЕКОНОМІЧНА ЧАСТИНА

4.1 Організаційно-економічне обґрунтування проекту

У даній роботі проведено аналіз технологій для розробки веб-системи відображення стану мережевого обладнання та розроблено продукт, який дозволяє відстежувати обсяг трафіку, що проходить через порти *Wi-Fi* роутера. Головною метою проекту було створення зручного інструменту для користувачів, що надає такі можливості:

- збір статистики про трафік, що проходить через порти роутера;
- зберігання зібраної статистики в базі даних;
- відображення статистики у вигляді інтерактивного графіка на веб-сторінці.

Для реалізації проекту було використано сучасні технології та інструменти. Зокрема, для написання серверної частини (*back-end*) було обрано мову програмування *C#* та фреймворк *ASP.NET MVC*, які забезпечують високу продуктивність та зручність у розробці. Для взаємодії з базою даних *MSSQL* було використано *Docker Desktop* на, якому її було запущено, що дозволяє ефективно працювати з даними. Інтегроване середовище розробки *Visual Studio* було використано для написання самого коду проекту. На стороні клієнта (*front-end*) було використано *HTML, CSS* та *JavaScript* для створення веб-інтерфейсу.

Перед початком розробки було проведено аналіз існуючих рішень, визначено основні вимоги та обмеження проекту. Було досліджено різні підходи до збору мережевої статистики та обрано оптимальний варіант з урахуванням особливостей цільової платформи, а саме *Wi-Fi* маршрутизатор.

У результаті було створено веб-систему, яка дозволяє користувачам легко та зручно відстежувати стан мережевого обладнання. Система має потенціал для подальшого розвитку та вдосконалення, оскільки може бути розширена додатковими функціями, такими як:

- моніторинг інших параметрів мережевого обладнання (наприклад, завантаження процесора, використання пам'яті);
- налаштування сповіщень про критичні події;
- збереження детальної історії стану мережі;
- інтеграція з іншими системами моніторингу.

Підсумовуючи, розроблена веб-система є важливим кроком у напрямку створення комплексного інструменту для моніторингу та управління мережевою інфраструктурою. Вона може бути корисною як для домашніх користувачів, так і для невеликих організацій, які прагнуть забезпечити стабільну та ефективну роботу своєї мережі.

В таблиці 4.1 наведено переваги та недоліки існуючих аналогів та розробленої в рамках цього дипломного проекту моніторингової веб-системи «*NetworkMonitor*».

Таблиця 4.1

Порівняння переваг та недоліків існуючих аналогів та моніторингової веб-системи «*NetworkMonitor*»

Моніторингова веб-системи	**Переваги**	**Недоліки**
SolarWinds NPM	Широкий спектр функцій, гнучкість, масштабованість, візуалізація даних, мобільний додаток	Висока вартість, складність налаштування, відсутність інтеграції з деякими системами
PRTG Network Monitor	Безкоштовна версія, простота налаштування, моніторинг *IoT*-пристроїв, вдосконалений моніторинг веб-додатків, розширена безпека, підтримка мобільних пристроїв	Обмежені можливості безкоштовної версії, відсутність деяких функцій в безкоштовній версії

Продовження таблиці 4.1

Моніторингова веб-системи	**Переваги**	**Недоліки**
Zabbix	Безкоштовна та відкрита платформа, гнучкість, масштабованість, інтеграція з іншими системами	Складність налаштування, відсутність мобільного додатку
Nagios	Безкоштовна та відкрита платформа, простота налаштування, широкий спектр плагінів	Складний інтерфейс командного рядка, відсутність візуалізації даних
Cisco Prime Infrastructure	Глибокий моніторинг обладнання *Cisco*, інтеграція з іншими продуктами *Cisco*	Негнучкість для роботи з обладнанням інших виробників, висока вартість
ManageEngine OpManager	Широкий спектр функцій, гнучкість, масштабованість, візуалізація даних, мобільний додаток	Висока вартість деяких функцій
NetworkMonitor	Простий та зрозумілий інтерфейс, масштабованість, безкоштовна та відкрита платформа, візуалізація даних	Немає мобільного додатку, немає досить глибокого моніторингу, без широкого спектру функцій

4.1.1 Організаційне обґрунтування

Проект з розробки веб-системи моніторингу вимагає комплексного підходу до планування та управління всіма етапами створення, починаючи від визначення вимог та функціональності системи до її розгортання та тестування. Організаційне обґрунтування включає аналіз вимог, вибір технологічного стеку

та визначення архітектури системи, що дозволяє забезпечити ефективну реалізацію проекту та досягнення поставлених цілей.

Класифікаційна оцінка проекту:

1. Клас проекту – монопроект. Проект з розробки веб-системи відображення стану мережевого обладнання є монопроектом, оскільки він орієнтований на створення одного конкретного продукту.

2. Тип проекту – змішаний. Проект включає технічні, організаційні та комерційні аспекти. Технічний аспект полягає у розробці програмного забезпечення, організаційний – у плануванні та управлінні проектом, комерційний – у потенційній монетизації та просуванні продукту.

3. Вид проекту – учбово-освітній. Проект розроблявся з метою практичного навчання та створення простої моніторингової веб-системи, яка буде легкою в користуванні та налаштуванні.

4. Тривалість проекту – короткостроковий.

5. Складність проекту – легка. Проект має легкий рівень складності, оскільки не має інновацій чи складних технологій, хіба що використання фреймворку *ASP.NET MVC*.

6. Розмір проекту – невеликий. Оскільки він орієнтований лише на моніторинг мережевого обладнання.

7. Рівень проекту – корпоративний. Проект спрямований на створення продукту, який може бути використаний невеликою організацією із обмеженим бюджетом.

Далі в таблиці 4.2 буде наведено класифікаційну оцінку проекту.

Таблиця 4.2

Класифікаційна оцінка проекту

Критерій	**Оцінка**
Клас проекту	Монопроект
Тип проекту	Змішаний
Вид проекту	Учбово-освітній

Продовження таблиці 4.2

Критерій	**Оцінка**
Тривалість проекту	Короткостроковий
Складність проекту	Легка
Розмір проекту	Невеликий
Рівень проекту	Корпоративний

Розробка веб-системи відображення стану мережевого обладнання включає наступні етапи виконання, кожен з яких має визначені терміни:

1. Підготовчий етап:
   - визначення цілей і завдань проекту, аналіз існуючих рішень і визначення основних вимог до системи;
   - тривалість: 2 тижні.
2. Проектування:
   - розробка технічного завдання, визначення архітектури системи;
   - тривалість: 1 тиждень.
3. Розробка:
   - написання коду з використанням *ASP.NET MVC*, тестування та налагодження;
   - тривалість: близько 1-го місяця.
4. Впровадження:
   - підключення за допомогою *SSH* до *WI-FI* маршрутизатора, налаштування та забезпечення безперебійної роботи;
   - тривалість: 2 дні.
5. Підтримка та розвиток:
   - регулярні оновлення, вдосконалення функціональності, та реалізація нових можливостей;
   - тривалість: постійно.

4.2 Маркетингове обґрунтування проекту

У сучасному світі інформаційні технології відіграють ключову роль в ефективному управлінні бізнес-процесами. Мережеве обладнання є невід'ємною частиною будь-якої організації, незалежно від її розмірів. Відповідно, наявність ефективних засобів моніторингу цього обладнання стає критично важливою для забезпечення безперебійної роботи. Веб-система відображення стану мережевого обладнання розроблена в цьому проекті може бути застосована для задоволення потреб невеликих організацій з обмеженим бюджетом, пропонуючи доступне, зрозуміле та масштабоване рішення.

4.2.1 Оцінка ринку збуту й конкуренція

Розробка може бути використана в невеликих організаціях, таких як малі бізнеси, освітні установи, некомерційні організації та інші підприємства, які потребують ефективного моніторингу мережевого обладнання, але мають обмежений бюджет на інформаційні технології (IT).

Основний ринок збуту — це локальні та регіональні ринки, де домінують невеликі підприємства, що прагнуть оптимізувати свої витрати на IT. Особливу увагу буде приділено ринкам країн з економікою, що розвивається, де потреба в доступних IT-рішеннях особливо висока.

Потенційними споживачами є малі та середні підприємства, державні установи, школи, університети та некомерційні організації. Це ті організації, які мають власну IT-інфраструктуру, але не можуть дозволити собі дорогі рішення для її моніторингу.

На ринку існує кілька конкурентів, які пропонують рішення для моніторингу мережевого обладнання. Деякі з них це компанії, як SolarWinds, PRTG Network Monitor та Zabbix. Вартість їхніх продуктів може коливатися від безкоштовних версій з обмеженими функціями до дорогих корпоративних рішень. Основною перевагою мого проекту є його низька вартість, простота у використанні та можливість масштабування без значних додаткових витрат.

4.2.3 Стратегія маркетингу

Поширення продукту можливо здійснювати через Інтернет, використовуючи рекламу на веб-сайтах та партнерські канали. Додатково можна створити кілька демонстраційних версій для безкоштовного тестування, що дозволить потенційним клієнтам ознайомитися з продуктом перед його придбанням.

Основним принципом ціноутворення буде доступність. Продукт буде мати кілька варіантів ліцензій: безкоштовну базову версію з обмеженими функціями та платні версії з розширеним функціоналом. Це дозволить залучити широку аудиторію та забезпечити гнучкість у виборі необхідного функціоналу.

Рекламна кампанія буде зосереджена на онлайн-каналах, таких як соціальні мережі, спеціалізовані форуми та блоги, присвячені IT та управлінню мережами. Планується активне використання контент-маркетингу, включаючи статті, відеоогляди та вебінари, щоб залучити потенційних клієнтів. Для стимулювання продажів будуть використовуватися знижки на першу покупку, програми лояльності та реферальні програми.

Для забезпечення високої якості обслуговування клієнтів буде організована служба підтримки, яка надаватиме консультації та технічну допомогу. Це включає електронну пошту, онлайн-чат та базу знань на веб-сайті продукту.

Реалізація проекту вимагатиме співпраці з кількома підприємствами, включаючи хостинг-провайдерів та маркетингові агентства. Це дозволить забезпечити надійне функціонування системи та ефективне просування продукту на ринку.

Співпраця з іншими компаніями в галузі дозволить розширити функціональні можливості продукту та підвищити його конкурентоспроможність. Можливе укладання партнерських угод з розробниками сумісного програмного забезпечення та постачальниками IT-послуг.

Нижче в таблиці 4.3 буде наведено порівняння цін на подібні моніторингові веб-системи в інших країнах.

Таблиця 4.3

Порівняння цін подібних моніторингових веб-систем в інших країнах

Продукт	**Країна**	**Початкова ціна (USD/рік)**	**Особливості**
SolarWinds NPM	США	від 2,995	Висока функціональність, потужні інструменти аналізу
PRTG Network Monitor	Німеччина	від 1,750	Легкість налаштування, хороша підтримка
Zabbix	Латвія	Безкоштовно (*Open Source*)	Велика спільнота, гнучкість налаштувань
Nagios XI	США	від 1,995	Широкі можливості інтеграції, багатий функціонал
ManageEngine OpManager	Індія	від 715	Добре підходить для малого та середнього бізнесу, зручний інтерфейс

В наступній таблиці 4.4 порівняння вигідності реалізації моніторингових веб-систем в різних країнах.

Таблиця 4.4

Порівняння вигідності реалізації моніторингових веб-систем в різних країнах.

Країна	**Вигідність реалізації**	**Причини**
Україна	Висока	Низькі витрати на розробку, зростаючий ринок IT
Індія	Висока	Високий рівень технічних кадрів, низькі витрати

Продовження таблиці 4.4

Країна	**Вигідність реалізації**	**Причини**
Польща	Середня	Зростаючий ринок, помірні витрати
США	Низька	Високі витрати на розробку, насичений ринок
Німеччина	Середня	Високий рівень конкуренції, але стабільний ринок
Бразилія	Середня	Ринок, що розвивається, але низька купівельна спроможність

4.3 Економічні розрахунки проекту

4.3.1 Визначення трудомісткості розробки мережевим плануванням

В таблиці 4.5 наведено склад робіт проекту і їхня тривалість.

Таблиця 4.5

Склад робіт проекту і їхня тривалість

№ коду роботи	**Найменування роботи**	**Тривалість (дні)**
0-1	Визначення цілей і завдань проекту, аналіз існуючих рішень і визначення основних вимог до системи	14
1-2	Розробка технічного завдання, визначення архітектури системи	7
2-3	Написання коду з використанням ASP.NET MVC та підключення бази даних	26
3-4	Тестування та налагодження веб-системи	8
4-5	Підключення за допомогою *SSH* до *WI-FI* маршрутизатора, забезпечення безперебійної роботи	2
5-6	Написання необхідної документації	3

Загальна тривалість створення проекту - 60 днів

На основі визначених етапів та їх тривалості побудовано мережевий графік (рис. 4.1), який відображає взаємозв'язки між роботами та критичний шлях проекту. Критичний шлях визначає мінімальний час, необхідний для завершення проекту, і включає роботи, затримка яких призведе до затримки всього проекту.

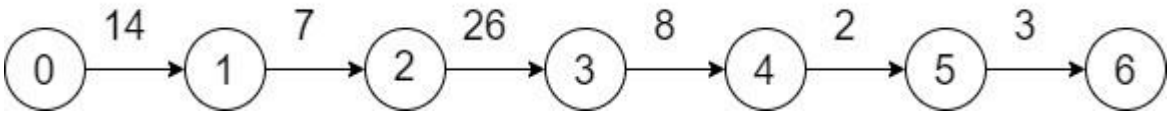


Рис. 4.1 – Мережевий графік проекту

4.3.2 Визначення трудомісткості розробки програмного продукту (ПП)

Тривалість розробки ПП залежить від обсягу ПП, трудомісткості його розробки, кваліфікації виконавців, а також планових строків, які диктуються умовами ринку.

У якості вихідних даних для визначення трудомісткості розробки ПП буде використано типовий склад етапів і укрупнені норми часу на розробку програмних засобів (ПЗ) [20]. Методом структурної аналогії по відповідних каталогах аналогів ПЗ на підставі таблиці 4.6 визначається обсяг програмних засобів у тисячах умовних машинних команд програми-аналога.

Таблиця 4.6

Каталог аналогів ПЗ

Найменування типу ПЗ	**Обсяг функцій ПЗ-*Vo*, ум. машинних команд**
1. ПЗ СУБД	2500-9800
2. ПЗ система ведення лінійних файлів	860-6600
3. Комплексні системи ведення БД	950-7430
4. ПЗ уведення інформації	1060-5750
5. ПЗ автоматизації засобів по каталозі	680-7000
6. ПЗ автоматизованих розрахунків	1300-8600

Продовження таблиці 4.6

Найменування типу ПЗ	**Обсяг функцій ПЗ-*Vo*, ум. машинних команд**
7. ПЗ загальної математики й ПЗ імітаційного моделювання	7800-8800
8. ПЗ організації обчислювального процесу	1300-10200
9. ПЗ оптимізаційних розрахунків	1300-4200

Виходячи з функціоналу та складності розробленої моніторингової веб-системи, можна провести аналогію з ПЗ уведення інформації (позиція 4 у таблиці 4.6). Припустимо, що обсяг такого ПЗ становить 2000 умовних машинних команд.

Для визначення трудомісткості розробки скористаємося даними з таблиці 4.7 (також узята з методичних вказівок), де наведено укрупнені норми часу на розробку ПЗ залежно від його обсягу.

Таблиця 4.7

Каталог аналогів ПЗ

Обсяг ПЗ, тис. умовних машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262
4.00	283
5.00	306

Згідно з таблицею 4.7, трудомісткість розробки 2000 умовних машинних команд відповідає 244 люд/год. Враховуючи, що розробка простої моніторингової веб-системи є відносно нескладним проектом, застосуємо поправочний коефіцієнт $К_к$ = 0,8, який враховує умови розробки (рівень кваліфікації розробників, доступність інструментів тощо).

Таким чином, трудомісткість розробки моніторингової веб-системи розраховується за формулою 4.1.

$$T_р = К_к * T_р \quad (4.1)$$

де $T_р$ – трудомісткість розробки;

$К_к$ – поправочний коефіцієнт.

$T_р = 0,8 * 244 = 195,2$ люд/год.

Трудомісткість розробки ПП повинна включати розробку наступних етапів:

– технічного завдання – ТЗ;

– технічного проекту – ТП;

– робочого проекту – РП;

– впровадження – ВН.

Як зазначено в роботі [21] трудомісткість розроблювального ПП визначається для кожного етапу окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни й ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{ТЗ} = T_р * L_i * К_н \quad (4.2)$$

$$T_{ТП} = T_р * L_2 * К_н \quad (4.3)$$

$$T_{РП} = T_р * L_3 * К_н * К_т \quad (4.4)$$

$$T_{ВН} = T_р * L_4 * К_н \quad (4.5)$$

де:

$T_р$ – укрупнена норма часу на розробку аналога ПЗ, чол/год (195,2), що коректується поправочним коефіцієнтом, що враховує умови розробки ПЗ, тобто в умовах комп'ютера, $К_к$ = (0.7/0.8);

L_i – питома вага і-го етапу розробки (табл. 4,8);

$К_н$ – поправочний коефіцієнт, що враховує ступінь новизни;

$К_т$ – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм.

Таблиця 4.8

Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПЗ

Код стадії	**Ступінь новизни**		
	А	**Б**	**В**
ТЗ	0.15	0.12	0.12
ТП	0.16	0.15	0.11
РП	0.55	0.58	0.61
ВП	0.14	0.15	0.16

Ступінь новизни мого проекту найбільш співпадає із варіантом Б. Тому що його можна вважати розвитком існуючих рішень. Отже $L_i = 0.12$, $L_2 = 0.15$, $L_3 = 0.58$, $L_4 = 0.15$.

Розрахуємо трудомісткість розробки ПП для кожного етапу окремо згідно з формулами (4.2) - (4.5).

$Т_{тз} = 195,2 * 0,7 * 0,12 * 0.4 = 6,56$ люд/год;

$Т_{тп} = 195,2 * 0,7 * 0,15 * 0.4 = 8,2$ люд/год;

$Т_{рп} = 195,2 * 0,7 * 0,58 * 0.4 * 0,7 = 22,19$ люд/год;

$Т_{вн} = 195,2 * 0,7 * 0,15 * 0.4 = 8,2$ люд/год;

Загальна трудомісткість розробки ПП складає:

$\sum T_i = 6,56 + 8,2 + 22,19 + 8,2 = 45,15$ люд/год.

Тривалість розробки ПП у роках вираховується за формулою 4.6.

$$Т_{ПП} = \sum T_{ij} / (8,0 \bullet 0,73) \quad (4.6)$$

де:

$\sum T_{ij}$- сумарна тривалість розробки, л;

8,0 - тривалість робочого дня (коефіцієнт перекладу в робочі дні), л; 0,73 - коефіцієнт перекладу в календарні дні;

T_{ij} - трудомісткість j-го виду робіт з і-м етапом.

$T_{ПП} = 45,15 / (8,0 * 0,73) = 7,73$ (дн.)

4.3.3 Розрахунок ціни розробки ПП

У підрозділі "Розрахунок ціни" для визначення ціни необхідно розрахувати основну заробітну плату розроблювачів, матеріальні витрати, вартість машино-години та інші витрати на розробку.

ПП розглядається й створюється як продукція виробничо-технічного призначення, що допускає багаторазове тиражування й відчуження від безпосередніх розроблювачів, отже для визначення ціни використовується формула 4.7.

$$Ц = К * С + П_р \quad (4.7)$$

де:

Ц – ціна розробки програмного продукту;

К – коефіцієнт обліку витрат на виготовлення дослідного зразка ПП (К = 1,1);

С – витрати на розробку програмної продукції (кошторисна собівартість);

$П_р$ – нормативний прибуток.

Нормативний прибуток розраховується за формулою 4.8.

$$П_р = (С - С_м) * Р_н / 100 \quad (4.8)$$

де:

$Р_н$ – норматив рентабельності (приймаємо 25%);

$С_м$ – матеріальні витрати.

Для розробки моніторингової веб-системи матеріальні витрати будуть мінімальними, оскільки основні ресурси – це час розробників та обчислювальні потужності. Враховуючи це, приймемо $С_м = 1000$ грн.

Витрати, пов'язані з використанням обчислювальної техніки, визначаються за формулою 4.9.

$$С_{еом} = t^{еом} * К_и^{еом} * Ц^{еом} * К_{бд}^{еом} * К_е^{еом} \quad (4.9)$$

де:

$t^{еом}$ – час використання ЕОМ для розробки моніторингової веб-системи (приймаємо 73 години);

$К_и^{еом}$ – поправочний коефіцієнт обліку часу використання ЕОМ (1,08);

$Ц^{еом}$ – ціна однієї години роботи ЕОМ (приймаємо 45 грн/год);

$К_{бд}^{еом}$ – коефіцієнт обліку ступеня використання СУБД (1,1, оскільки використовується *MSSQL*);

$К_е^{еом}$ – коефіцієнт обліку швидкодії ЕОМ (1,0, оскільки швидкодія ЕОМ більше $20*10^{30}$ оп/с).

Підставивши значення у формулу, отримаємо:

$С_{еом} = 73 * 1,08 * 45 * 1,1 * 1 = 3902,58$ грн.

Враховуючи, що над проектом працює один розробник із середньомісячним окладом 20 000 грн, а трудомісткість розробки становить 45,15 людино-годин (згідно із попередніми розрахунками), основна заробітна плата розраховується за формулою 4.10.

$$С_{зо} = (З_і * К_о * \tau_і) / Д_р \quad (4.10)$$

де:

$З_і$ – середньомісячний оклад і-того виконавця, грн;

$К_о$ – коефіцієнт обліку окладу керівників і консультантів проекту (рекомендується 0,1);

$\tau_і$ – трудомісткість робіт, виконуваних і-тим виконавцем, люд-днів;

$Д_р$ – середня кількість робочих днів у місяці (рекомендовано 21-22).

Підставивши значення у формулу, отримаємо:

$С_{зо} = (20000 * 0,1 * (45,15 / 8)) / 22 = 513,06$ грн.

Додаткова заробітна плата розраховується як 10% від основної заробітної плати:

$С_{зд} = С_{зо} * 0,1 = 513,06 * 0,1 = 51,3$ грн.

Відрахування на соціальне страхування складають 22% від суми основної та додаткової заробітної плати:

$С_{сс} = 0,22 * (С_{зо} + С_{зд}) = 0,22 * (513,06 + 51,3) = 124,16$ грн.

Накладні витрати приймаємо у розмірі 50% від основної заробітної плати:

$С_н = 0,5 * С_{зо} = 0,5 * 513,06 = 256,53$ грн.

Загальна кошторисна собівартість розробки моніторингової веб-системи розраховується як сума всіх витрат, за формулою 4.11.

$$С = С_м + С_е + С_{зо} + С_{зд} + С_{сс} + С_н \quad (4.11)$$

$С = 1000 + 3902,58 + 513,06 + 51,3 + 124,16 + 256,53 = 5847,63$ грн.

Нормативний прибуток ($П_р$) визначається за формулою 4,12.

$$П_р = (С - С_м) * Р_н / 100 \quad (4.12)$$

Рн – норматив рентабельності, (рекомендується 25 %);

См – матеріальні витрати, грн.

$П_р = (5847,63 – 1000) * 25 / 100 = 1211,9$ грн.

Ціна розробки вираховується за наведеною вище формулою 4.7.

$Ц = 1,1 * 5847,63 + 1211,9 = 7644,29$ грн.

Отже, розрахункова ціна розробки веб-системи відображення стану мережевого обладнання становить 7644,29 грн.

Висновки до четвертого розділу

На основі організаційно-економічного обґрунтування проекту можна зробити висновок, що розробка веб-системи відображення стану мережевого обладнання є доцільною з точки зору інвестицій та ресурсів. Проект передбачає використання сучасних технологій, таких як *ASP.NET MVC* для серверної частини та *Docker* для роботи з базами даних, що забезпечує високу продуктивність та зручність у розробці. Аналіз існуючих рішень і визначення основних вимог дозволили обрати оптимальні методи для збору і відображення мережевої статистики, що відповідають потребам цільової аудиторії проекту.

Маркетингове обґрунтування проекту підкреслює важливість наявності ефективних інструментів моніторингу мережевого обладнання для невеликих організацій. Розроблена веб-система надає доступне та зрозуміле рішення для малих бізнесів, освітніх установ та некомерційних організацій, які мають

обмежений бюджет на інформаційні технології. Оцінка ринку збуту та аналіз конкуренції показали, що існує значний попит на подібні системи, особливо серед підприємств, які шукають прості у використанні і недорогі рішення для моніторингу мережевого обладнання.

Економічні розрахунки проекту демонструють реалістичність і фінансову обґрунтованість проекту. Загальна тривалість розробки складає 60 днів, з чітко визначеними етапами робіт та їхньою тривалістю. Витрати на розробку веб-системи були ретельно прораховані, включаючи матеріальні витрати, трудомісткість, та нормативний прибуток, що становить 1211,9 грн. В результаті, розрахункова ціна розробки системи складає 7644,29 грн, що є конкурентоспроможною в порівнянні з аналогічними рішеннями на ринку.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ

5.1 Основні положення охорони праці

Охорона праці – це комплекс правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів, спрямованих на збереження життя, здоров’я і працездатності працівників під час виконання ними своїх трудових обов’язків (Закон України "Про охорону праці"). Це невід'ємна складова будь-якої трудової діяльності, включаючи розробку програмного забезпечення. Виконання вимог законодавства та правил безпеки праці дозволяє уникати нещасних випадків та професійних захворювань, забезпечуючи здоров’я і працездатність працівників.

Згідно зі статтею 13 Закону України "Про охорону праці" [22], роботодавець зобов’язаний створити на робочому місці умови праці відповідно до нормативно-правових актів і забезпечити дотримання законодавчих вимог щодо прав працівників у сфері охорони праці.

Працівники повинні знати і виконувати вимоги нормативно-правових актів з охорони праці, проходити медичні огляди у встановленому порядку та дотримуватись правил безпеки на робочому місці.

Держава здійснює управління охороною праці, встановлює єдині вимоги, контролює дотримання законодавства та проводить інші заходи для забезпечення безпечних і здорових умов праці.

Важливим аспектом охорони праці є навчання та інструктаж працівників з питань безпеки праці. Роботодавець повинен забезпечити проведення інструктажів, навчання та перевірку знань працівників щодо охорони праці.

Охорона праці також включає моніторинг і оцінку ризиків на робочому місці, щоб визначити потенційні небезпеки та вжити заходів для їх усунення або мінімізації. Роботодавці повинні запроваджувати системи управління охороною праці, які дозволяють виявляти небезпеки, аналізувати ризики та реалізовувати

ефективні заходи для їх контролю. Це включає регулярний перегляд і оновлення процедур безпеки, а також забезпечення працівників засобами індивідуального захисту.

5.2 Недоліки та умови роботи за комп'ютером

Робота за комп'ютером має специфічні особливості та ризики для здоров'я працівників. Основні недоліки включають:

1. Навантаження на зір через те, що тривала робота за монітором може призвести до втоми очей, погіршення зору, сухості та подразнення.

2. Навантаження на опорно-руховий апарат, адже неправильна постава, тривале сидіння в одній позі та повторювані рухи можуть викликати біль у спині, шиї, руках та кистях.

3. Психологічне навантаження, тому що інформаційне перевантаження, стрес та недостатність спілкування можуть негативно впливати на психічне здоров'я.

Для мінімізації цих ризиків слід дотримуватись таких умов праці:

1. Ергономіка робочого місця, наприклад, правильно підібрані стіл, стілець, монітор та клавіатура забезпечують комфортну роботу.

2. Рівномірне і достатнє освітлення допомагає зменшити навантаження на зір.

3. Регулярні перерви, фізичні вправи та розминка допомагають уникнути втоми.

4. Створення сприятливої атмосфери на роботі, підтримка колег і можливість спілкування допомагають зберегти психічне здоров'я.

Також важливо звертати увагу на такі аспекти:

1. Температура та вологість в офісі повинні бути оптимальними для комфортної роботи. Надто висока або низька температура може впливати на самопочуття і продуктивність працівників.

2. Шум на робочому місці може бути джерелом стресу і зниження концентрації. Використання шумозахисних панелей або навушників допомагає зменшити вплив шуму.

Важливо забезпечити можливості для зняття стресу, наприклад, через психологічну підтримку або організацію заходів для відпочинку.

5.3 Електробезпека

Під час роботи з комп'ютерною технікою важливо пам'ятати про потенційну небезпеку ураження електричним струмом через несправне обладнання або пошкоджені дроти.

Основні правила електробезпеки включають:

- всі електроприлади повинні бути заземлені;
- використання джерел безперебійного живлення (ДБЖ) і мережевих фільтрів;
- після завершення роботи або під час тривалих перерв електрообладнання потрібно вимикати;
- ремонт електрообладнання повинні виконувати кваліфіковані фахівці;

працівники мають проходити інструктаж і знати правила безпеки.

5.4 Пожежна безпека при роботі з комп'ютером

Комп'ютерна техніка може бути джерелом пожежної небезпеки. Для запобігання пожежі слід дотримуватись правил пожежної безпеки в Україні, затверджених наказом МВС України від 30 грудня 2014 року № 1417 [23]. Зокрема:

- розташовувати комп'ютери на відстані не менше 1 метра від легкозаймистих матеріалів;
- електропроводка повинна бути справною, з наявністю заземлення;
- вентиляційні отвори не повинні бути заблоковані;
- стан електрообладнання слід перевіряти та обслуговувати;
- наявність вогнегасників і вміння працівників ними користуватись;

негайне відключення обладнання, повідомлення пожежної служби та гасіння пожежі.

5.5 Вентиляція

Відповідно до державних санітарних норм та правил роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [24], приміщення з комп'ютерами мають бути обладнані системами вентиляції для підтримання оптимальних параметрів мікроклімату. Недостатня вентиляція може призвести до підвищення температури та вологості, накопичення шкідливих речовин і зниження рівня кисню в повітрі.

Для забезпечення належної вентиляції слід:

- регулярно провітрювати приміщення, відкривати вікна кожні 1-2 години;
- використовувати системи вентиляції, у разі великої кількості техніки або відсутності можливості регулярного провітрювання;
- Контролювати параметри мікроклімату, температура, вологість і рівень кисню повинні відповідати вимогам;
- Очищати повітря, використовувати повітряні фільтри.

Забезпечення належної вентиляції сприяє створенню безпечних і комфортних умов праці, збереженню здоров'я працівників та підвищенню їхньої продуктивності.

Висновок до п'ятого розділу

Охорона праці є обов'язковою системою заходів для забезпечення безпеки та здоров'я працівників, включаючи тих, хто працює з комп'ютерами. Роботодавці та працівники зобов'язані дотримуватись законодавства і правил безпеки, а держава здійснює контроль та управління у цій сфері.

Робота за комп'ютером може призвести до проблем із зором, опорно-руховим апаратом та психологічним станом. Для мінімізації цих ризиків необхідно дотримуватись ергономіки робочого місця, забезпечувати належне

освітлення, дотримуватись режиму праці та відпочинку, а також створювати сприятливий психологічний клімат.

Робота з комп’ютерною технікою пов’язана з ризиком ураження електричним струмом. Для забезпечення електробезпеки необхідно заземлювати обладнання, використовувати захисні пристрої, відключати техніку від мережі після роботи, не займатись самостійним ремонтом та проходити інструктаж з електробезпеки.

ЗАГАЛЬНІ ВИСНОВКИ

У дипломній роботі було досягнуто низку визначених завдань та описано у відповідних розділах, що розкривають теоретичні та практичні аспекти розробки веб-системи *NetworkMonitor* для моніторингу мережевого обладнання.

У першому розділі проведено аналіз сучасних методів і технологій моніторингу мереж. Розглянуто основні концепції та порівняно різні інструменти, що підтвердило необхідність розробки гнучкої та безпечної веб-системи.

Другий розділ присвячено проектуванню системи *NetworkMonitor*. Розглянуто архітектуру, методи збирання даних і їх захисту, використання шифрування *HTTPS*, а також заплановано впровадження аутентифікації та авторизації користувачів.

У третьому розділі описано практичну реалізацію системи. Використано *C#* у фреймворку *ASP.NET Core*, *JavaScript*, *HTML*, *CSS*, *Microsoft SQL Server* та *Entity Framework Core* для створення компонентів системи, які підключаються до віддалених серверів і збирають мережеву статистику в реальному часі.

Четвертий розділ містить техніко-економічне обґрунтування проекту, включаючи аналіз витрат, кошторисну собівартість та розрахунок прибутку. Доведено доцільність розробки веб-системи з точки зору інвестицій та ресурсів.

П'ятий розділ розглядає питання безпеки та вентиляції приміщень з комп'ютерною технікою. Зазначено необхідність систем вентиляції для підтримання оптимальних параметрів мікроклімату, а також заходи з пожежної безпеки та організаційні аспекти роботи з комп'ютерною технікою.

Таким чином, у дипломній роботі виконано всі визначені завдання: проведено аналіз методів моніторингу мереж, спроектовано та розроблено веб-систему NetworkMonitor, здійснено техніко-економічне обґрунтування проекту та розглянуто питання безпеки й експлуатації приміщень з комп'ютерною технікою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Типи мережевих протоколів і їх призначення. URL: https://deltahost.ua/ua/tipi-merezhevix-protokoliv-i-ih-priznachennya-http-ip-ssh-ftp-pop3-mac.html (дата звернення: 14.03.2024).

2. А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник. Комп’ютерні мережі : навч. посіб. Львів, «Магнолія 2006». 253 с.

3. О. П. Петренко, В. О. Олійник, О. В. Петренко. Комп'ютерні мережі: концепції, протоколи та технології : навч. посіб. Львівська політехніка, 2018. 600 с.

4. Tanenbaum, A. S., & Wetherall, D. J. Computer Networks. USA, Pearson, 2011. 912 с.

5. Mauro, D., & Schmidt, K. Essential SNMP. USA, O'Reilly Media, 2005. 290 с.

6. Solarwinds Network Performance Monitor. URL: https://www.solarwinds.com/network-performance-monitor (дата звернення 22.03.2024).

7. PRTG Network Monitor URL: https://www.paessler.com/prtg/prtg-network-monitor (дата звернення 23.03.2024).

8. Zabbix network monitor. URL: https://www.zabbix.com/network_monitoring (дата звернення 23.03.2024).

9. Nagios Network Analyzer. URL: https://www.nagios.com/products/nagios-network-analyzer/ (дата звернення 24.03.2024).

10. Cisco Prime Infrastructure. URL: https://www.cisco.com/c/en_ca/products/cloud-systems-management/prime-infrastructure/index.html (дата звернення 24.03.2024).

11. ManageEngine OpManager. URL: https://www.manageengine.com/network-monitoring/network-performance-monitoring.html?indexnew (дата звернення 24.03.2024).

12. AJAX з допомогою jQuery. URL: https://catalogueofarticles.com/tehnologiyi/ajax-z-dopomogoju-jquery/ (дата звернення 02.04.2024).

13. Difference between time-series database and relational database. URL: https://stackoverflow.com/questions/35428606/difference-between-time-series-database-and-relational-database (дата звернення 05.04.2024).

14. Розробка з боку Front end – що це таке і чим відрізняється від Back end? URL: https://dan-it.com.ua/uk/blog/rozrobka-z-boku-front-end-shho-ce-take-i-chim-vidriznjaietsja-vid-back-end/ (дата звернення 28.03.2024).

15. What is Visual Studio? URL: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022 (дата звернення 19.03.2024).

16. What is SQL Server? URL: https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16 (дата звернення 08.04.2024).

17. Docker overview. URL: https://docs.docker.com/guides/docker-overview/ (дата звернення 08.04.2024).

18. ASP.NET overview. URL: https://learn.microsoft.com/en-us/aspnet/overview (дата звернення 05.04.2024).

19. Entity framework core. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення 06.04.2024).

20. С.С. Тумилина. Інвестиційне проектування : практ. посіб. з економічного обґрунтування інвестиційних проектів. Фінстат – Інформ, 1995. 205 с.

21. Вігуржинська С.Ю, Колесник В.І. Дипломне проектування економічної частини проекту : метод. посіб. Одеса, ОНТУ, 2016. 23 с.

22. Закон Про охорону праці в Україні: Відомості Верховної Ради України (ВВР), 1992, № 49. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 07.05.2024).

23. Наказ Про затвердження Правил пожежної безпеки в Україні: Наказ М-ва внутр. справ України від 30.12.2014 р. № 1417: станом на 7 квіт. 2023 р. URL: https://zakon.rada.gov.ua/laws/show/z0252-15#Text (дата звернення: 07.05.2024).

24. ДСанПІН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 08.05.2024).