Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Зейлика Романа Юрійовича

на здобуття ступеня вищої освіти Бакалавра

Вебсистема сканування хостів і портів для аналізу вразливостей мережі.

Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ. 220160.22.01.03 ПЗ

Виконав студент 3 курсу група КБс-22-1 Роман ЗЕЙЛИК
Керівник канд. техн. наук, доцент Ігор МУЛЯР
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

6 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Зейлику Роману Юрійовичу

1 Тема роботи Вебсистема сканування хостів і портів для аналізу вразливостей мережі.

Керівник роботи к.т.н. доцент Ігор Муляр

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи Проаналізувати предметну область та існуючі рішення в галузі сканування мереж. Сформулювати постановку задачі та визначити функціональні вимоги до системи. Розробити архітектуру та загальну структуру вебсистеми. Обгрунтувати вибір інструментів та технологій для реалізації. Реалізувати модулі для сканування хостів і портів. Розробити серверну частину додатку з використанням обраного стеку технологій. Реалізувати клієнтську частину вебінтерфейсу для запуску сканування та перегляду результатів. Провести тестування функціональності додатку.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. Аналіз предметної області. Огляд існуючих рішень. Постановка задачі. Функціональні вимоги до системи. Архітектура та загальна структура системи Вибір засобів та технологій для реалізації. Розробка модулів для сканування. Розробка серверної частини додатку. Розробка клієнтської частини додатку. Тестування додатку

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Архітектура додатку. Діаграма послідовності сервісу сканування. Схема бази даних

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7 Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	
Ознайомлення з предметною областю	Лютий	
Дослідження існуючих рішень	Лютий	
Постановка задачі	Березень	
Визначення загальних принципів рішення задачі	Березень	
Деталізація принципів рішення задачі	Квітень	
Розробка проєктних рішень	Квітень	
Апробація проєктних рішень	Травень	
Оформлення пояснювальної записки згідно вимог	Травень	
Оформлення графічної частини	Червень	
Захист КР	Червень	

Студент Роман ЗЕЙЛИК

Керівник кваліфікаційної роботи Ігор МУЛЯР

АНОТАЦІЯ

Тема кваліфікаційної роботи: Вебсистема сканування хостів і портів для аналізу вразливостей мережі.

Автор роботи: Зейлик Роман Юрійович.

Керівник роботи: Муляр Ігор Володимирович.

Пояснювальна записка: 67 с., 2 додатки, 17 рисунків, 41 джерело.

Графічна частина: 3 плакати, 10 презентаційних слайдів.

МЕРЕЖЕВА БЕЗПЕКА, АНАЛІЗ УРАЗЛИВОСТЕЙ, СКАНУВАННЯ ПОРТІВ, ІДЕНТИФІКАЦІЯ СЕРВІСІВ.

Кваліфікаційна робота бакалавра присвячена розробці вебсистеми для сканування хостів і портів з метою аналізу вразливостей мережі.

У роботі проведено аналіз існуючих методів виявлення вразливостей у мережевих інфраструктурах, розглянуто основні загрози кібербезпеки, а також досліджено інструменти для сканування мережевих хостів і портів. Розроблено вебсистему, що реалізує можливості перевірки доступності хостів, виявлення відкритих портів та ідентифікації сервісів. Впроваджено механізми взаємодії з мережею для ефективного аналізу безпеки. Запропоноване рішення може бути використане адміністраторами мереж для оцінки рівня безпеки інфраструктури та своєчасного виявлення потенційних загроз.

28.05.2025 _____

ABSTRACT

Subject of qualification work: Web system for scanning hosts and ports to analyze network vulnerabilities

Author: Zeilyk Roman Yuriyovych.

Head of work: Mulyar Ihor Volodymyrovych.

Explanatory note: 67 p., 2 appendices, 17 figures, 41 sources

Graphic part: 3 posters, 10 presentation slides.

NETWORK SECURITY, VULNERABILITY ANALYSIS, PORT SCANNING, HOST SCANNING, SERVICE IDENTIFICATION.

The bachelor's thesis is dedicated to the development of a web-based system for host and port scanning to analyze network vulnerabilities.

The study examines existing methods for identifying vulnerabilities in network infrastructures, explores key cybersecurity threats, and investigates tools for scanning network hosts and ports. A web system has been developed that enables host availability checks, detection of open ports, and service identification. Mechanisms for network interaction have been implemented to ensure effective security analysis. The proposed solution can be utilized by network administrators to assess infrastructure security levels and promptly detect potential threats.

28 . 05 .2025 _____

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології та комунікаційні мережі займають центральне місце в усіх сферах діяльності. Від ефективної роботи цих систем залежить не тільки бізнес-процеси, а й забезпечення надійності та безпеки даних, що передаються через мережу. З кожним роком зростає кількість загроз і атак на комп'ютерні мережі, що ставить перед фахівцями з кібербезпеки нові виклики. Одним із ключових аспектів захисту є виявлення вразливостей на етапі проектування та експлуатації мереж, що дозволяє знижувати ризики несанкціонованого доступу та втрати даних.

Сканування хостів та портів є невід'ємною частиною процесу оцінки безпеки будь-якої мережі. За допомогою цього методу можна виявити відкриті порти, визначити активні служби та з'ясувати, які з них можуть становити потенційну загрозу для системи. Завдяки скануванню можна здійснювати моніторинг і оцінку безпеки, а також своєчасно виявляти недоліки в мережевій інфраструктурі, що дозволяє запобігти можливим атакам.

Метою цієї кваліфікаційної роботи є розробка вебсистеми для сканування хостів і портів, що дозволить ефективно аналізувати вразливості мережі. Система має бути зручною та інтуїтивно зрозумілою для користувачів і забезпечувати оперативне виявлення небезпечних конфігурацій мережі. В рамках роботи будуть розглянуті різні методи сканування портів, а також особливості їх застосування для виявлення слабких місць в мережі. Окрім того, важливим аспектом є розробка веб-інтерфейсу, який спростить процес сканування та аналізу результатів для користувачів з різним рівнем технічної підготовки.

Розробка такої системи є важливим кроком у напрямку забезпечення належного рівня безпеки інформаційних систем та мереж, що дозволяє своєчасно виявляти загрози та усувати вразливості до того, як вони стануть причиною серйозних інцидентів.

1 ДОСЛІДЖЕННЯ СФЕРИ СКАНЕРІВ ХОСТІВ ТА ПОРТІВ, ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області системи сканування хостів та портів

Із стрімким розвитком цифрових технологій і зростанням обсягів даних, що передаються через комп’ютерні мережі, питання забезпечення інформаційної безпеки стало одним із ключових у сфері кіберзахисту. Ефективне управління безпекою вимагає системного підходу до виявлення, оцінки та усунення вразливостей у мережевій інфраструктурі. Одним із базових та водночас критично важливих інструментів для досягнення цієї мети є сканування хостів та портів.

Сканування хостів – це процес виявлення активних пристроїв у мережі, які відповідають на певні типи мережевих запитів. Його основна мета полягає у визначенні, які IP-адреси є активними, що дозволяє сформувати початкову карту мережі. Виявлення хостів дає змогу з'ясувати, які пристрої функціонують у певному сегменті мережі, а також оцінити їхню реакцію на різні протоколи – ICMP, TCP, UDP тощо [1].

Сканування портів, у свою чергу, дозволяє дізнатися, які саме сервіси запущені на знайдених хостах. Кожен сервіс у мережі, як правило, пов’язаний із певним портом – наприклад, вебсервери використовують порти 80 (HTTP) та 443 (HTTPS), віддалене адміністрування через SSH працює на порту 22. Якщо порт відкритий, це означає, що відповідний сервіс готовий до взаємодії з клієнтом. Таким чином, аналіз відкритих портів дозволяє отримати уявлення про структуру та функціональність мережі, а також визначити потенційно вразливі точки доступу [2].

Сканування хостів і портів не обмежується лише виявленням активності – сучасні інструменти здатні виконувати розширений збір інформації. Сюди входить визначення версій сервісів, операційних систем, аналіз банерів, а також перевірка на відомі вразливості за допомогою CVE-баз [3]. Такий підхід дозволяє виявити не лише відкриті сервіси, а й конкретні загрози, які з ними пов’язані.

Серед основних методів сканування, які використовуються в практиці кіберзахисту, можна виокремити:

– ICMP-сканування (Ping sweep) – дозволяє виявити активні хости шляхом надсилання запитів типу Echo Request;

– TCP SYN-сканування – ефективний спосіб виявлення відкритих портів без завершення повного з'єднання (так зване напіввідкрите сканування);

– UDP-сканування – дозволяє перевірити доступність служб, що використовують протокол UDP, хоча результати такого сканування є менш однозначними через специфіку цього протоколу;

– сканування версій – дає змогу дізнатися не лише про відкритість порту, а й про програмне забезпечення, що за ним стоїть;

– сканування на вразливості – інтегрується з базами відомих вразливостей і дозволяє одразу оцінити ризики [4].

У більш широкому контексті сканування є складовою етапу розвідки (reconnaissance) – як у процесі етичного тестування на проникнення, так і в реальних кібератаках. Саме тому виявлення відкритих портів і сервісів важливе не тільки для безпеки – воно є необхідною умовою для прогнозування можливих сценаріїв атак.

Сканування може виконуватись як у межах внутрішнього аудиту мережі, так і ззовні – при цьому в останньому випадку часто імітується поведінка зловмисника. У зв'язку з цим важливо враховувати юридичні та етичні аспекти застосування таких засобів. Використання сканерів без дозволу власника ресурсу може трактуватися як несанкціонований доступ і мати відповідні правові наслідки. Саме тому сканування в рамках навчальних курсів, кваліфікаційних робіт або тестових лабораторій має проводитись у контрольованому середовищі.

З розвитком хмарних технологій та активним впровадженням моделей "інфраструктура як сервіс" (IaaS) та "платформа як сервіс" (PaaS) [5], сканування мережі отримало новий виток актуальності. У віртуалізованих середовищах, де ресурси створюються та знищуються динамічно, традиційні підходи до аудиту

можуть бути недостатніми. Саме тому автоматизовані сканери хостів та портів інтегруються в CI/CD-процеси для забезпечення безпеки ще на етапі розгортання нових сервісів. У цьому контексті велике значення мають сканери, що підтримують API-інтеграцію та вміють працювати з контейнеризованими середовищами (Docker, Kubernetes) [6].

Окремо слід згадати і про специфіку сканування в середовищах IoT (Internet of Things), де багато пристроїв використовують нестандартні порти або працюють з мінімальними мережевими стеками. Такі пристрої, як інтелектуальні камери, сенсори, маршрутизатори, часто мають слабкий рівень захисту й можуть залишатися непоміченими традиційними засобами контролю. В умовах зростання кількості IoT-вузлів у промислових та побутових мережах, завдання виявлення і аналізу таких хостів виходить на перший план. Це спонукає до розробки спеціалізованих сканерів, здатних враховувати специфіку протоколів IoT та обмеженість ресурсів на самих пристроях.

Таким чином, предметна область сканування хостів і портів охоплює не лише технічні інструменти, а й методологію розвідки, аналізу вразливостей, тестування на проникнення та інформаційного аудиту загалом [7]. Її розуміння є необхідним для побудови сучасних систем забезпечення безпеки, адаптованих до викликів цифрової епохи.

1.2 Огляд існуючих рішень систем сканування хостів та портів

Сфера сканування хостів і портів представлена широким спектром програмних рішень, які можуть виконувати як базові дії – наприклад, виявлення активних IP-адрес чи відкритих портів, – так і глибший аналіз мережевих сервісів, вразливостей та конфігурацій. Існуючі інструменти можуть значно відрізнятися за функціональністю, рівнем автоматизації, швидкістю роботи, а також можливістю інтеграції в більші системи інформаційної безпеки.

Загалом такі рішення можна умовно поділити на кілька категорій:

– низькорівневі сканери портів та хостів, що працюють безпосередньо з мережею й надають "сирі" дані – прикладом є Nmap, Masscan;

– інструменти для аналізу вразливостей, які доповнюють сканування інформацією з баз даних уразливостей – наприклад, OpenVAS, Nessus;

– сканери для вебдодатків або спеціалізованих середовищ, що працюють із протоколами HTTP/HTTPS, або орієнтовані на аналіз специфічних типів інфраструктур – зокрема, Nikto, OWASP ZAP, Acunetix;

– масштабовані рішення, які дозволяють сканувати великі обсяги адресного простору, як Zmap, що може охоплювати мільйони хостів за лічені хвилини [8].

Кожне з рішень має власну філософію використання: деякі орієнтовані на досвідчених фахівців з кібербезпеки, інші – на автоматизоване застосування в рамках регулярного контролю мережі. Вибір конкретного інструменту зазвичай залежить від поставлених цілей: швидкість чи точність, простота у використанні чи гнучкість налаштувань, базовий аналіз чи поглиблена перевірка на вразливості.

Розглянемо найпопулярніші інструменти з цієї галузі з акцентом на їхні можливості, переваги, недоліки та доцільність використання у різних умовах.

Nmap - це безкоштовний і відкритий інструмент для сканування мереж, який дозволяє виявляти активні хости, відкриті порти, типи операційних систем, активні сервіси та навіть вразливості на основі виявлених сигнатур. Інструмент широко використовується фахівцями з інформаційної безпеки, системними адміністраторами, а також в освітніх цілях [9].

Основна перевага Nmap полягає у його гнучкості та багатофункціональності. Він підтримує різні методи сканування, включаючи TCP SYN-скан, UDP-сканування, сканування версій сервісів, виявлення ОС (OS fingerprinting) тощо. Окрім цього, Nmap має власну мову скриптів – NSE (Nmap Scripting Engine), яка дозволяє автоматизувати складні перевірки та розширювати функціональність інструмента [10]. За допомогою NSE можна, наприклад,

виконувати брутфорс-атаки, перевіряти наявність відомих вразливостей чи навіть витягувати інформацію з конкретних сервісів. Приклад сканування за допомогою Nmap можна переглянути на рисунку 1.1.

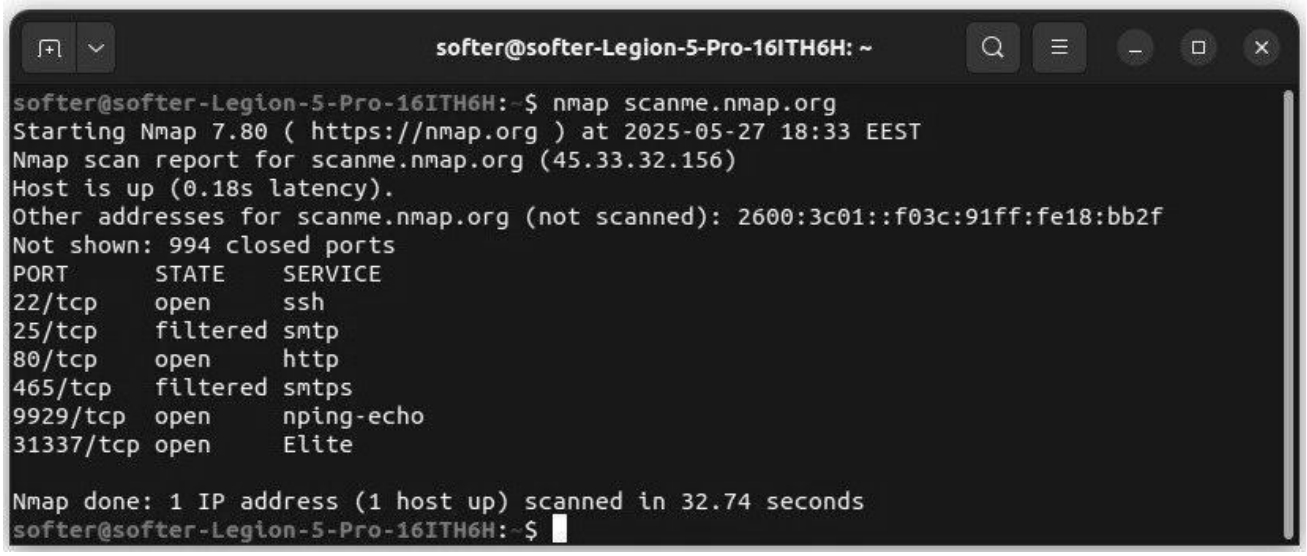


Рисунок 1.1 – Сканування за допомогою Nmap

Серед інших особливостей – можливість збереження результатів у різних форматах (звичайний текст, XML, HTML), що зручно для подальшого аналізу або звітності. Існує також графічний інтерфейс Zenmap, який спрощує використання інструмента для початківців або для візуалізації результатів сканування [11].

До переваг Nmap можна віднести:

– високу точність результатів;
– широкий набір режимів сканування;
– підтримку великої кількості параметрів;
– активну спільноту та хорошу документацію;
– можливість автоматизації завдяки скриптам NSE.

Однак інструмент має й деякі недоліки. Зокрема, при скануванні великих діапазонів IP-адрес або при використанні повільних типів сканування, час виконання може бути значним. Крім того, через активний характер сканування деякі мережеві пристрої можуть виявляти та блокувати його, розцінюючи дії як потенційно шкідливі. Важливо також пам’ятати, що ефективність виявлення

залежить від налаштувань цільових пристроїв (наприклад, наявність фаєрволів або систем виявлення вторгнень може спотворювати результати).

Загалом, Nmap є універсальним і надійним рішенням для сканування мережі, яке часто використовується як базовий інструмент у багатьох сценаріях, від звичайного “пінгу” до комплексного аналізу інфраструктури.

Masscan – це надзвичайно швидкий мережевий сканер, розроблений спеціально для сканування великих обсягів адресного простору за мінімальний час. Його основна особливість полягає у високій продуктивності: інструмент здатний сканувати до декількох мільйонів IP-адрес за хвилину, що робить його ідеальним для широкомасштабних досліджень мереж [12].

Принцип роботи Masscan багато в чому схожий на TCP SYN-скан, який також використовується в Nmap, однак реалізація тут побудована на основі власного мережевого стеку. Це дозволяє досягти максимальної швидкості, але водночас створює деякі обмеження – наприклад, Masscan не може визначати версію сервісів чи проводити складний аналіз відповідей.

Інструмент підтримує обмежений, але ефективний набір функцій: виявлення активних хостів, перевірка відкритих портів, експорт результатів у зручному форматі. Завдяки цьому він часто використовується як попередній етап перед глибшим скануванням за допомогою інших інструментів, наприклад Nmap.

Переваги Masscan:

– надзвичайно висока швидкість роботи;

– простота у використанні;

– можливість сканування великих діапазонів;

– підтримка збереження результатів для подальшої обробки.

До недоліків можна віднести:

– обмежену функціональність (відсутність аналізу версій, сервісів, ОС);

– складнощі із запуском на деяких системах через використання "сирого" доступу до мережі;

– можливість блокування з боку міжмережевих екранів через агресивний характер сканування.

Masscan є ідеальним вибором для швидкого виявлення активних хостів або відкритих портів у великій мережі, після чого ці результати можна використати для глибшого аналізу іншими інструментами. Інтерфейс вебверсії Masscan можна переглянути на рисунку 1.2 [13].

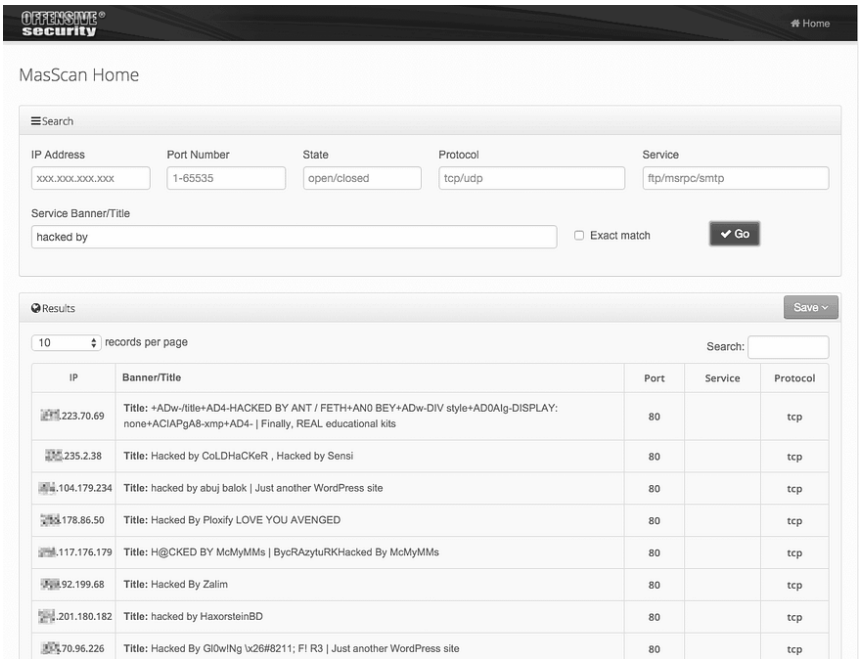


Рисунок 1.2 – Інтерфейс вебверсії Masscan [13]

OpenVAS (Open Vulnerability Assessment System) – це потужна система для сканування вразливостей, яка дозволяє виявляти широкий спектр проблем безпеки в мережевих пристроях та сервісах. Вона є частиною більшого проєкту Greenbone Vulnerability Management (GVM) і надає комплексний підхід до виявлення ризиків у мережевій інфраструктурі [14].

На відміну від базових сканерів портів, таких як Nmap або Masscan, OpenVAS не просто визначає, який порт відкритий, а глибоко аналізує знайдені сервіси на наявність відомих вразливостей. Для цього він використовує велику й постійно оновлювану базу знань про вразливості, включно з CVE-

ідентифікаторами, конфігураційними проблемами та іншими критичними параметрами [15].

Система побудована за клієнт-серверною архітектурою, де сервер відповідає за сканування, а клієнт – за керування завданнями й перегляд результатів. В OpenVAS передбачено механізми планування сканувань, фільтрації результатів, генерації звітів і навіть рекомендацій щодо усунення виявлених проблем.

Основні переваги OpenVAS:

– глибокий аналіз сервісів на наявність вразливостей; – регулярне оновлення бази даних загроз;

– підтримка генерації детальних звітів;

– можливість автоматизації процесів сканування;

– відкрите та безкоштовне рішення.

Серед недоліків:

– складність налаштування й запуску, особливо для новачків;

– вимогливість до ресурсів системи;

– порівняно тривалий час сканування;

– можливість хибнопозитивних результатів (false positives), коли вразливість вказується, хоча її може й не бути насправді.

OpenVAS підходить для використання у професійному середовищі, де необхідний глибокий аналіз мережевої безпеки та можливість регулярного моніторингу. Його переваги проявляються найкраще у великих мережах або при проведенні повного аудиту інформаційної інфраструктури. Інтерфейс програми OpenVAS можна переглянути на рисунку 1.3. [14]

OWASP ZAP (Zed Attack Proxy) – це безкоштовний і відкритий інструмент для тестування безпеки вебдодатків, розроблений і підтримуваний спільнотою OWASP (Open Worldwide Application Security Project). ZAP орієнтований як на новачків, так і на досвідчених тестувальників, і є одним із найпопулярніших засобів для проведення динамічного аналізу безпеки вебресурсів [16].

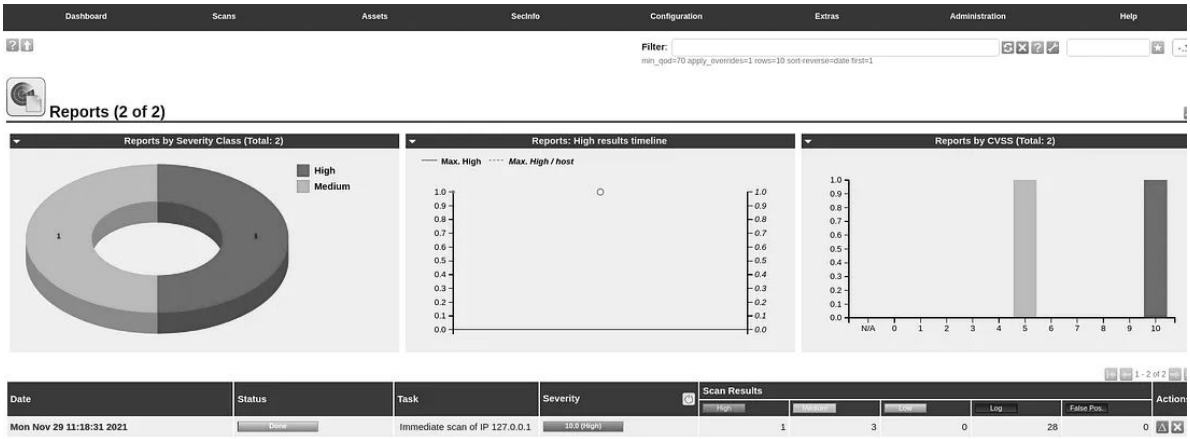


Рисунок 1.3 – Інтерфейс програми OpenVAS [14]

Основна ідея ZAP полягає в тому, щоб діяти як проксі-сервер між користувачем і вебдодатком. Таким чином, інструмент може аналізувати HTTP/HTTPS-трафік, перехоплювати запити, змінювати їх, здійснювати автоматичне сканування та виявляти вразливості, пов’язані з вебпротоколами та логікою додатку.

OWASP ZAP дозволяє проводити як пасивне, так і активне сканування вебсторінок, а також автоматично виявляти поширені вебвразливості, такі як SQL-ін’єкції, XSS, небезпечні cookie, відкриті редиректи та багато інших. Інструмент підтримує ручне тестування, має вбудовані засоби для фуззингу, редагування запитів, перегляду сесій, і може бути інтегрований у процеси безперервної інтеграції та розгортання для автоматизації перевірок безпеки [16].

Серед основних переваг ZAP варто відзначити його зручний графічний інтерфейс, активну підтримку з боку спільноти, регулярні оновлення, а також високу гнучкість у налаштуваннях і можливість розширення через плагіни. Інструмент підходить як для ручного, так і для повністю автоматизованого тестування, що робить його універсальним вибором для фахівців з безпеки [17].

Водночас, ZAP не позбавлений недоліків. Наприклад, для ефективного використання інструмента користувач повинен мати базове розуміння HTTP-протоколів і принципів роботи вебзастосунків. Автоматичне сканування іноді може давати хибнопозитивні результати, а сучасні односторінкові застосунки

(SPA) іноді потребують додаткової конфігурації для коректної перевірки [18]. Крім того, ZAP не орієнтований на аналіз інфраструктури за межами вебрівня, тобто його застосування обмежується саме вебдодатками. Інтерфейс OWASP ZAP можна переглянути на рисунку 1.4. [16]

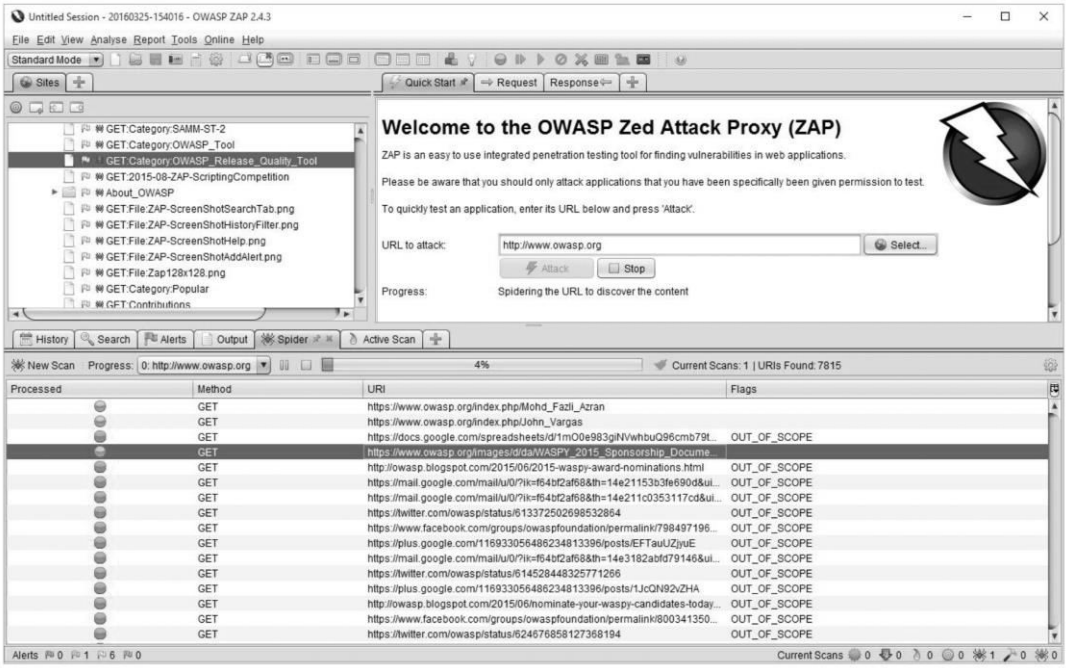


Рисунок 1.4 – Інтерфейс програми OWASP ZAP [16]

OWASP ZAP є чудовим вибором для розробників, тестувальників і аналітиків безпеки, які працюють із вебдодатками. Завдяки простоті у використанні та широким можливостям, він часто використовується як точка входу до світу веббезпеки.

ZMap – це високопродуктивний мережевий сканер, створений для швидкого аналізу великих обсягів IP-простору. Розроблений дослідниками з Мічиганського університету, інструмент був спеціально оптимізований для використання в академічних та аналітичних цілях, зокрема для досліджень глобального стану Інтернету, моніторингу протоколів, аналізу тенденцій безпеки тощо [19].

На відміну від універсальних інструментів на зразок Nmap, ZMap спеціалізується на швидкісному однотипному скануванні. Він не призначений для багатофункціонального аналізу окремих хостів, а натомість дозволяє проводити

сканування одного порту по всьому IP-простору IPv4 у межах кількох хвилин. Це досягається завдяки використанню спеціально оптимізованого мережевого стеку і суворої спеціалізації інструмента під один конкретний тип завдань. Приклад роботи Zmap можна переглянути на рисунку 1.5.

Рисунок 1.5 – Приклад використання Zmap

Серед переваг ZMap можна виділити його виняткову швидкість – за умови правильної конфігурації він здатний просканувати весь доступний діапазон IPv4 менш ніж за годину. Інструмент підтримує модулі для роботи з різними протоколами, такими як TCP SYN, UDP, ICMP, а також дозволяє легко зберігати результати у форматах, зручних для подальшої обробки.

Однак ZMap має й свої обмеження. Зокрема, він не підтримує глибокого аналізу відповідей і не визначає версії сервісів чи операційних систем. Також через агресивний характер роботи ZMap може спричиняти спрацьовування захисних систем або навіть блокування з боку провайдерів, якщо не дотримуватися етичних та технічних норм.

Таким чином, ZMap є незамінним інструментом у випадках, коли потрібне швидке й масштабне сканування мережі, але потребує обережного використання та грамотного планування сканувальних кампаній.

Проведений огляд інструментів для сканування хостів, портів, виявлення вразливостей та аналізу вебдодатків демонструє широкий спектр доступних рішень, кожне з яких орієнтоване на свій клас завдань. Інструменти на кшталт

Nmap залишаються основою для початкового збору інформації про мережу, в той час як рішення типу OpenVAS дозволяють здійснювати глибокий аналіз безпеки з урахуванням відомих уразливостей. Своєю чергою, OWASP ZAP орієнтований саме на специфіку вебдодатків і дозволяє ефективно знаходити типові вразливості, пов'язані з вебінтерфейсами та логікою взаємодії з користувачем. А ZMap демонструє можливості високошвидкісного масового сканування, незамінного у випадках досліджень глобального масштабу.

Таким чином, жоден із представлених інструментів не є універсальним, однак у поєднанні вони формують комплексний інструментарій для проведення повного аналізу мережевої безпеки. При розробці власної системи доцільно орієнтуватися на гнучкість, масштабованість та поєднання підходів – починаючи від поверхневого сканування до глибокого аналізу знайдених сервісів і вразливостей.

1.3 Постановка задачі

У сучасному цифровому середовищі, де значна частина бізнес-процесів, державного управління та особистих ініціатив реалізується з використанням мережевих технологій, питання безпеки комп'ютерних систем та інфраструктури стає пріоритетним [20]. Однією з базових процедур, яка дозволяє оцінити ступінь захищеності інформаційного середовища, є сканування мережі, що включає в себе виявлення активних хостів, відкритих портів, мережевих сервісів та супутньої інформації про пристрої.

Сучасні інструменти для мережевого сканування, такі як Nmap, Masscan, OWASP ZAP та інші, володіють широким функціоналом і високою ефективністю. Проте значна частина з них має консольний інтерфейс, потребує знання команд, мережевих протоколів, принципів роботи TCP/IP-стека, що ускладнює їх використання для звичайних користувачів або адміністраторів, які не мають

глибоких технічних навичок. Навіть ті, хто має досвід роботи з такими інструментами, нерідко потребують простішого способу інтеграції результатів сканування з внутрішніми обліковими або аналітичними системами, а також зручного зберігання і перегляду історії перевірок.

У зв'язку з цим виникає потреба у створенні веборієнтованої системи, яка б поєднувала простоту використання з можливостями гнучкого сканування та збереження результатів. Така система має працювати локально, тобто без необхідності використовувати сторонні хмарні сервіси, забезпечуючи, водночас, достатній рівень функціональності для виконання основних діагностичних задач. Важливою є також можливість подальшого розширення системи - як у напрямку глибшого аналізу мережі, так і в частині інтеграції з відкритими API для збору додаткової інформації про домени або IP-адреси.

Метою даної кваліфікаційної роботи є створення локальної веборієнтованої системи для сканування хостів і портів, яка дозволяє користувачам швидко та зручно виконувати базовий аналіз доступності мережевих ресурсів. Система має на меті спростити процес первинної діагностики мережі, зробити його доступним для ширшого кола користувачів – від адміністраторів невеликих інфраструктур до звичайних власників вебресурсів, яким важливо переконатися у відсутності базових вразливостей або помилок у конфігурації [21].

На відміну від багатьох професійних інструментів, запропонована система повинна мати простий інтерфейс та зрозумілий процес використання. Користувач має можливість ввести IP-адресу або доменне ім'я, після чого система виконує перевірку активності хосту, визначає відкриті порти та, за можливості, встановлює тип сервісу, який відповідає на кожному з них. Окремо передбачається реалізація спроби ідентифікації операційної системи, а також використання відкритих API для збору додаткової інформації про хост – наприклад, щодо належності IP-адреси, доменного імені чи сертифікатів.

Для гнучкості в реалізації передбачається поєднання двох підходів: власних алгоритмів для виконання простого сканування, а також залучення перевірених

зовнішніх інструментів для виконання складніших завдань. Це дозволяє з одного боку – уникнути повної залежності від сторонніх засобів, а з іншого – забезпечити якісні результати без потреби розробляти складні компоненти з нуля.

Результати кожного сканування мають зберігатися, щоб користувач мав змогу переглядати історію перевірок та аналізувати динаміку змін. Це важливо для виявлення нових сервісів, які з'явилися на хості з часом, або для фіксації змін у відкритих портах після оновлень чи змін конфігурації на стороні серверів.

Система передбачається як локальний вебдодаток, який може бути розгорнутий безпосередньо на робочій машині користувача або на внутрішньому сервері організації. Такий підхід дозволяє забезпечити контроль над усім процесом сканування, уникнути витоку даних у зовнішні сервіси, а також забезпечити гнучкість у налаштуванні. У разі необхідності не виключається можливість розгортання системи на віддаленому сервері з публічним доступом, проте основна ідея полягає саме у локальності й безпосередньому контролі користувача над сканувальним процесом.

Функціональність системи має бути адаптованою як до зовнішніх мереж (інтернет), так і до внутрішніх корпоративних або домашніх мереж. Водночас варто враховувати, що при роботі у внутрішній мережі деякі можливості можуть бути обмежені – зокрема, доступ до зовнішніх API, що використовуються для збору додаткової інформації про домени чи IP-адреси. Незважаючи на це, ключова функціональність сканування та збереження результатів залишатиметься доступною.

Загалом, запропонована система має вирішити проблему доступності базового інструменту мережевого аналізу для широкого кола користувачів. Вона поєднуватиме простоту у використанні, достатній рівень технічної гнучкості, можливість розширення, а також підтримку збереження і перегляду результатів. Це дозволить не лише виконувати разові перевірки, а й накопичувати історію сканувань для подальшого аналізу змін у конфігурації мережевих ресурсів.

2 ПРОЄКТУВАННЯ СИСТЕМИ СКАНУВАННЯ ХОСТІВ ТА ПОРТІВ

2.1 Функціональні вимоги до системи

Дана система сканування хостів і портів покликана забезпечити користувача зручним інструментом для базового аналізу доступності мережевих ресурсів. Основне завдання полягає у виявленні активних хостів у мережі, визначенні відкритих портів та зборі базової інформації про сервіси, що працюють на цих портах. При цьому важливо, щоб система залишалась простою у використанні, надаючи інтуїтивно зрозумілий вебінтерфейс навіть для користувачів, які не мають глибоких знань у сфері мережевої безпеки.

У якості основного функціоналу передбачається можливість введення користувачем IP-адреси, діапазону адрес або доменного імені для подальшого сканування. Система повинна самостійно ініціювати перевірку на наявність активності хосту, тобто чи відповідає вказана адреса на мережеві запити. У разі підтвердження активності хосту, система продовжує виконання перевірки портів – на предмет їх відкритості, а також, за можливості, ідентифікації сервісів, що працюють на відповідних портах. Окремо передбачається функція визначення операційної системи або програмного стеку, який використовується на віддаленому вузлі – ця можливість буде реалізована шляхом інтеграції з уже існуючими інструментами, що підтримують подібну функціональність [22].

Крім базового сканування, система повинна мати механізм збереження результатів кожної перевірки у базу даних. Це дозволить користувачам повертатися до попередніх результатів, переглядати хронологію змін, а також отримувати загальну аналітику щодо стану мережевих ресурсів. Передбачається реалізація розділу з історією сканувань, де можна буде переглядати як короткі звіти, так і детальні результати попередніх перевірок.

Для забезпечення більшої зручності користувачів система повинна підтримувати можливість планування сканувань. Це дозволить автоматизувати процес перевірки мережі на регулярній основі. Наприклад, користувач може

налаштувати щоденне або щотижневе сканування певних хостів або мережевих сегментів. Планування сканувань має бути інтегроване з механізмом звітності, що дозволить автоматично генерувати і зберігати результати сканувань на заплановані періоди. Це особливо корисно для організацій, де потрібно регулярно перевіряти стан мережі без необхідності вручну ініціювати кожне сканування.

Окрім цього, система повинна мати можливість взаємодії з відкритими API для збору додаткової інформації про домени, IP-адреси та сертифікати хостів. Наприклад, може бути корисно отримувати географічну інформацію про розташування хоста або дані про власника IP-адреси. Така функціональність дозволяє значно розширити можливості аналізу мережевих ресурсів без потреби в створенні окремих баз даних або інструментів для цих цілей.

Система також має бути гнучкою щодо режиму використання. Вона повинна працювати як у локальній мережі, так і у зовнішньому інтернет-просторі. При роботі в локальній мережі система може мати певні обмеження, наприклад, у відсутності доступу до зовнішніх API або до публічних даних, однак функціональність сканування портів і хостів залишатиметься повною. У разі роботи з публічно доступними хостами з інтернету система може використовувати повний набір функцій, зокрема доступ до зовнішніх API для збору додаткової інформації.

Окрему увагу слід приділити інтерфейсу користувача, який повинен бути інтуїтивно зрозумілим та зручним навіть для користувачів, які не мають технічної підготовки в сфері мережевих технологій. Вебінтерфейс має забезпечувати швидкий доступ до основних функцій системи – запуску сканування, перегляду результатів, історії перевірок та налаштування параметрів. Форми для запуску сканування повинні бути простими: достатньо одного або кількох полів для введення IP-адреси, діапазону або доменного імені, з можливістю вибору режиму чи типу перевірки.

Результати сканування повинні відображатися у наочному вигляді – бажано у вигляді таблиці, де кожен запис містить інформацію про IP, відкриті порти,

сервіси та додаткові відомості, якщо вони були отримані. Колірне виділення активних/неактивних хостів, підсвічування знайдених сервісів або потенційних вразливостей може значно покращити сприйняття результатів. Важливими є також механізми сортування, фільтрації та пошуку по історії сканувань – вони дозволять користувачам швидко орієнтуватися у великій кількості збережених даних.

Також інтерфейс повинен надавати зворотний зв’язок про хід виконання запитів – наприклад, індикатор виконання сканування, повідомлення про успішне завершення або помилки під час перевірки. Загальний підхід до проєктування інтерфейсу повинен відповідати принципам мінімалізму: користувач має отримувати лише ту інформацію, яка дійсно є корисною в конкретному контексті.

Отже, підсумовуючи всі аспекти, даний додаток повинен відповідати наступним вимогам та містити наступні функції:

– можливість проведення сканування хостів і портів на для перевірки чи працюють вони чи ні;

– можливість виявлення сервісів та продуктів, що знаходяться за заданими портами;

– можливість виявлення операційної системи на хості;

– перегляд активних мереж, в якому перебуває пристрій;

– зручний та простий інтерфейс користувача;

– зберігання та перегляд даних про сканування;

– можливість планування сканувань.

Описавши функціональні вимоги, можна приступати до наступних етапів проєктування додатку і посилатись на них у процесі розробки. Вони є основою для визначення архітектури системи, розробки бази даних, вибору технологій та побудови взаємодії між компонентами. Чітко сформульовані вимоги також дозволяють забезпечити відповідність кінцевого продукту очікуванням користувача та спростити тестування функціональності системи.

2.2 Архітектура за загальна структура системи

Розробка системи сканування хостів та портів передбачає створення вебзастосунку з чітким розмежуванням між клієнтською та серверною частинами, що реалізується у вигляді класичної клієнт-серверної архітектури [23]. Такий підхід забезпечує масштабованість, гнучкість у розробці окремих компонентів та зручність подальшої підтримки системи. Простий приклад даної архітектури можна переглянути на рисунку 2.1

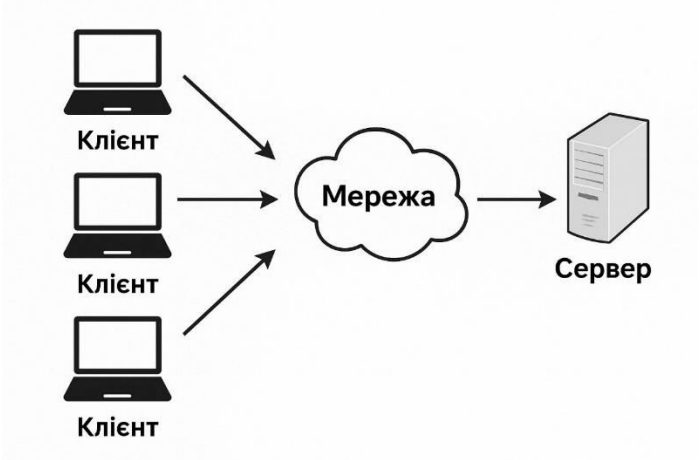


Рисунок 2.1 – Принцип роботи клієнт серверної архітектури

У цій архітектурі серверна частина відповідає за обробку логіки сканування, збереження даних, взаємодію з зовнішніми сервісами (наприклад, API для отримання інформації про IP або домени), а також за обробку запитів з клієнта. У той час клієнтська частина реалізує інтерфейс користувача та забезпечує взаємодію з сервером через API – переважно у форматі JSON [24].

З огляду на потребу в динамічному оновленні вмісту сторінки (наприклад, при відображенні прогресу сканування або результатів у режимі реального часу), планується використання підходу Single Page Application (SPA) для клієнтської частини. Це дозволить зменшити навантаження на сервер при зміні стану інтерфейсу, а також забезпечить зручнішу й швидшу взаємодію користувача з системою [25].

Кожен запит на сканування, сформований через клієнтський інтерфейс, надсилається на сервер, де виконується відповідна обробка. Після завершення сканування результати зберігаються у базі даних, звідки їх можна переглядати пізніше через історію або інтерфейс звітності. Такий розподіл відповідальностей між клієнтом і сервером дозволяє чітко організувати логіку системи, розділити її на незалежні модулі й у майбутньому легко розширювати функціональність без зміни основної структури.

Використання архітектури Single Page Application у клієнтській частині обумовлено прагненням забезпечити максимально швидку і зручну взаємодію користувача з системою без постійного перезавантаження сторінок. У SPA-застосунках усі основні ресурси (HTML-шаблони, стилі, JavaScript-код) завантажуються один раз при відкритті сторінки, а подальша навігація та обробка даних відбувається через запити до бекенду, здебільшого у форматі JSON. Принцип роботи SPA показано на рисунку 2.2 [25]

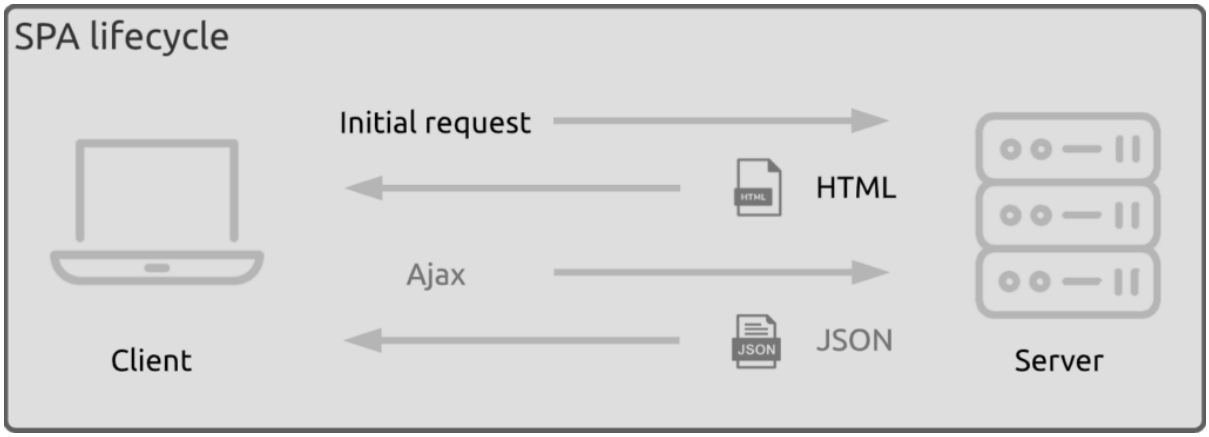


Рисунок 2.2 – Принцип роботи SPA [25]

Такий підхід є особливо доречним для системи сканування, оскільки дозволяє організувати динамічне оновлення інтерфейсу в реальному часі – наприклад, під час виконання тривалого сканування користувач може бачити його статус без необхідності оновлювати сторінку. Це також забезпечує більш плавний досвід використання: перехід між розділами (історія, налаштування, запуск

сканування) відбувається миттєво, що сприяє зменшенню навантаження на сервер і покращенню продуктивності.

Крім того, SPA дає змогу розділити обов'язки між фронтендом і бекендом: клієнтська частина відповідає лише за візуалізацію та взаємодію з користувачем, а всі критичні операції, включно зі скануванням, зберіганням даних і логікою безпеки, реалізуються на сервері. Такий підхід сприяє кращому контролю над логікою додатку, дозволяє ізолювати потенційно вразливі процеси на серверній стороні та мінімізувати ризики несанкціонованого доступу до функціоналу сканування.

З технічної точки зору, це також відкриває шлях до можливого використання WebSocket-з'єднання [26] для доставки результатів сканування в реальному часі, хоча базова реалізація може обмежитися періодичними AJAX-запитами (polling). У майбутньому такий підхід дозволить легко впровадити оновлення без потреби суттєвого перероблення архітектури.

У межах реалізації динамічного оновлення даних на клієнтській стороні одним з ключових завдань є забезпечення передачі результатів сканування або статусів виконання в режимі реального часу. Оскільки процеси сканування можуть тривати досить довго, користувачеві важливо бачити проміжні результати або хоча б знати, що система працює.

Для цього доцільно використовувати механізм двостороннього зв'язку між клієнтом і сервером, реалізований за допомогою WebSocket. У цій системі планується використання Pusher – хмарного сервісу, який надає зручний API для роботи з каналами в реальному часі та добре інтегрується з вебтехнологіями [27].

Pusher дозволяє серверу «пушити» оновлення до клієнта відразу після появи нових даних, минаючи необхідність ручного оновлення сторінки чи періодичного опитування сервера (polling). Це особливо зручно для таких випадків, як:

– оновлення статусу сканування (очікується, виконується, завершено, помилка та інші);

– надсилання часткових результатів по мірі обробки кожного хоста чи порту, щоб інформувати вже отримані дані;

– інформування про помилки або завершення сканування.

На стороні сервера реалізується логіка, яка по завершенню певного етапу або отриманні нового результату ініціює виклик до Pusher API, що в свою чергу транслює повідомлення у відповідний канал, пов’язаний із поточним користувачем або конкретним завданням сканування. Клієнтська частина, підписана на цей канал, отримує повідомлення миттєво й оновлює інтерфейс.

Цей підхід не лише зменшує навантаження на сервер, але й підвищує зручність користування системою, створюючи відчуття «живого» інтерфейсу, який оперативно реагує на дії користувача. У майбутньому можливе розширення цього функціоналу – наприклад, додавання групових повідомлень або централізованої панелі моніторингу кількох паралельних сканувань.

Серверна частина системи виконує ключову роль у логіці управління скануванням, обробкою результатів та взаємодією з базою даних. Враховуючи потенційно тривалі процеси, пов’язані з мережею – такі як сканування портів, визначення сервісів чи взаємодія з зовнішніми API – їх виконання у головному потоці вебзастосунку є недоцільним. Це може призвести до затримок у відповіді системи та загального зниження її стабільності.

Для вирішення цієї проблеми планується використання системи фонових задач, зокрема Sidekiq, яка добре інтегрується з архітектурою клієнт-серверного вебзастосунку та дозволяє виконувати ресурсоємні завдання поза межами основного потоку запитів.

Кожен запит на сканування, ініційований користувачем через вебінтерфейс, не запускає процес безпосередньо. Натомість, на сервері створюється новий запис про сканування в базі даних та додається задача до черги виконання, яка обробляється окремим “воркером”. Це дозволяє:

– не блокувати вебсервер;

– ефективно масштабувати навантаження;

– контролювати кількість одночасних сканувань;

– повторно запускати невдалі задачі.

Sidekiq підтримує механізм повторів, пріоритизації задач і моніторингу виконання, що робить його особливо зручним для побудови надійної фонової обробки в системах, де важлива стійкість до збоїв [28].

Після завершення фонової задачі з результатами сканування, інформація записується в базу даних, після чого генерується повідомлення через Pusher (описано вище) для оновлення інтерфейсу користувача. Таким чином, користувач бачить актуальний статус сканування та результати одразу після завершення.

Централізація всієї логіки сканування у фонових задачах також дозволяє у майбутньому винести її в окремий мікросервіс або навіть розгорнути сканери на ізольованих хостах (для підвищення безпеки або розподілу навантаження).

Отже, розроблювана вебсистема сканування хостів і портів базується на клієнт-серверній архітектурі, у якій логіка чітко розділена між клієнтською частиною (інтерфейс користувача) та серверною частиною (керування задачами, обробка результатів, збереження даних). Такий підхід забезпечує структурованість, масштабованість і зручність у подальшій підтримці та розширенні системи.

Клієнтська частина реалізована як односторінковий застосунок (SPA), що дозволяє забезпечити швидку взаємодію користувача з системою без потреби постійного перезавантаження сторінок. Через API клієнт надсилає запити на сервер для запуску сканування, а також отримує оновлення результатів у реальному часі завдяки інтеграції з сервісом Pusher. Це дозволяє миттєво відображати зміну статусу сканування, надходження результатів або повідомлень про помилки.

На серверній стороні опрацьовуються запити, створюються відповідні записи у базі даних та ініціюється виконання фонових задач за допомогою системи Sidekiq. Такий підхід дозволяє не блокувати основний потік програми, ефективно обробляти складні або тривалі операції, а також масштабувати

виконання задач при зростанні кількості користувачів. Після завершення сканування результати фіксуються в базі, і сервер надсилає сигнал через Pusher, який в свою чергу миттєво доставляє повідомлення клієнту для оновлення інтерфейсу. Кінцеву схему архітектури проєкту можна переглянути в додатку Б.

Уся зібрана інформація – включно з історією сканувань, деталями про хости, результати перевірок – зберігається у базі даних. Це дозволяє не лише переглядати попередні сканування, а й повторно використовувати вже отриману інформацію для порівняння змін у мережевій інфраструктурі.

Таким чином, загальна взаємодія між компонентами забезпечує безперервну роботу: користувач ініціює сканування через зручний інтерфейс, сервер обробляє запит і делегує його виконання у фоновий режим, а результати надходять у реальному часі без потреби вручну оновлювати сторінку. Це створює цілісний, стабільний і сучасний досвід користування системою.

2.3 Вибір засобів та технологій для реалізації додатку

Реалізація серверної частини вебсистеми вимагає вибору технологій, які здатні забезпечити надійну обробку HTTP-запитів, інтеграцію з базою даних, керування фоновими задачами, підтримку API, а також можливість масштабування у разі необхідності. На сучасному етапі існує велика кількість фреймворків та середовищ, які можна використовувати для створення подібних систем. Для аналізу розглянемо кілька найбільш популярних та широко використовуваних рішень: Node.js, Django, Spring Boot, Laravel та Ruby on Rails.

Node.js є платформою на основі JavaScript, яка дозволяє створювати високопродуктивні серверні додатки з неблокуючим введенням/виведенням. Його ключова перевага – асинхронна модель обробки запитів, яка добре підходить для сценаріїв із великою кількістю одночасних підключень. Завдяки численним бібліотекам (наприклад, Express.js) Node.js може швидко

розгортатися, однак для побудови масштабованої та структурованої системи часто доводиться самостійно інтегрувати велику кількість модулів та рішень. Це ускладнює підтримку проєкту у довготривалій перспективі. Крім того, хоч JavaScript є популярною мовою, вона не завжди підходить для реалізації складної серверної логіки через свою динамічну типізацію та низький рівень формалізації структури коду [29].

Іншим потужним інструментом є Django – фреймворк для Python, який слідує принципу "з коробки" (batteries included). Він забезпечує швидку розробку завдяки великій кількості вбудованих рішень: ORM, адмін-панель, засоби безпеки, підтримка авторизації, маршрутизації тощо. Django також підтримує розширення через численні плагіни та добре підходить для розробки API через Django REST Framework. Проте, у випадках, коли проєкт має бути дуже чітко модульованим або потребує великої кількості паралельної обробки задач (наприклад, у вигляді воркерів), Python та Django не завжди забезпечують найкращу продуктивність, особливо у порівнянні з іншими, більш "низькорівневими" рішеннями [30].

Spring Boot – це один з найпопулярніших фреймворків для розробки серверних застосунків на Java. Він дає змогу швидко створювати самодостатні програми, що включають в себе всі необхідні компоненти для веброзробки, включаючи API, безпеку, доступ до баз даних та інші функції. Основною перевагою Spring Boot є його здатність до масштабування та висока продуктивність завдяки використанню Java. Spring також добре підтримує паралельну обробку запитів, що може бути корисним для обробки фонових задач. Однак розробка на Java може бути менш швидкою порівняно з іншими мовами через більш складний синтаксис та велике навантаження на пам'ять у великих проєктах. Крім того, для роботи з Spring Boot необхідно мати глибші знання Java, що може вплинути на час навчання та розробки [31].

Laravel – це фреймворк для PHP, який забезпечує зручну інтерфейсну розробку, простоту використання та великі можливості для розширення. Laravel

широко застосовується в малих і середніх проєктах завдяки швидкості розробки, інтуїтивно зрозумілому синтаксису і численним вбудованим функціям, таким як Eloquent ORM, системи маршрутизації та шаблонізації, підтримка авторизації та аутентифікації. Однак продуктивність PHP у порівнянні з такими мовами, як Java або Go, може бути обмеженою при великих навантаженнях або при роботі з великими обсягами даних. Тому Laravel підходить для створення невеликих вебдодатків, але для масштабних проєктів його використання може бути не таким ефективним, особливо якщо потрібно працювати з великою кількістю паралельних запитів [32].

У порівнянні з іншими технологіями, Ruby on Rails поєднує простоту розробки, потужні вбудовані інструменти для створення API, чудову підтримку модульності та швидку інтеграцію з базами даних. З самого початку фреймворк орієнтований на забезпечення максимальної швидкості розробки завдяки принципу "Convention over Configuration" – тобто надає готові шаблони та стандарти, що мінімізує необхідність налаштовувати кожен аспект системи вручну. Ruby on Rails містить потужний ActiveRecord (ORM для роботи з базами даних), підтримує мікросервісну архітектуру та добре інтегрується з фоновими системами обробки задач, такими як Sidekiq. Rails також забезпечує чудову підтримку RESTful API, що робить його ідеальним для реалізації вебсистеми сканування хостів і портів, де зручність інтеграції з різними сервісами (наприклад, для отримання даних про хости через відкриті API) є важливою вимогою [33].

Враховуючи вищеописані переваги та недоліки кожної технології, для цього проєкту найбільш підходящим вибором є Ruby on Rails. Його потужний набір інструментів для швидкої розробки, чудова інтеграція з базами даних та можливість використання фонових задач роблять його ідеальним для реалізації вебсистеми з постійною взаємодією з базою даних і зовнішніми API. Rails також добре підходить для проєктів, де важлива швидка розробка, зручний для користувача інтерфейс та стабільна робота з великою кількістю одночасних запитів.

Ruby on Rails є потужним фреймворком, що дозволяє швидко розробляти вебдодатки, що є особливо важливим для проекту вебсистеми сканування хостів та портів. Його основна перевага полягає в принципі "Convention over Configuration", що дозволяє значно скоротити час налаштувань на початковому етапі і зосередитися безпосередньо на реалізації функціональності. Завдяки цьому, розробка системи буде здійснюватися швидко, а кожен аспект системи буде відповідати стандартам фреймворку, що полегшує подальшу підтримку.

Однією з ключових складових є ActiveRecord, який дозволяє зручно працювати з базою даних, не вимагаючи написання складних SQL-запитів. Це дозволяє зберігати всі результати сканувань, історії користувачів і налаштування в базі даних, спрощуючи їх обробку та взаємодію з іншими частинами системи. Використання migrations дозволяє легко управляти змінами в базі даних і забезпечує зручний контроль за версіями її структури.

Rails також має потужну підтримку для створення RESTful API, що необхідно для забезпечення взаємодії між серверною частиною та клієнтським інтерфейсом. Це дозволяє ефективно передавати дані про хости, порти і результати сканувань, а також реалізувати інтерфейс для запуску нових сканувань або перевірки статусу поточних.

Для обробки складних і тривалих операцій, таких як сканування хостів чи інтеграція з зовнішніми API, важливо використовувати систему фонових задач. Sidekiq є одним з найпопулярніших інструментів для цього в Ruby on Rails. Він дозволяє асинхронно обробляти довготривалі задачі, не блокуючи основний потік програми. Це забезпечує швидкий і зручний процес взаємодії з користувачем, оскільки користувач може отримати оновлення про статус сканування, не чекаючи на завершення операцій у реальному часі.

Ще однією важливою перевагою Rails є вбудовані засоби безпеки, які захищають від поширених вразливостей, таких як SQL-ін'єкції, Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF). Для системи сканування хостів та портів це критично важливо, оскільки додаток працюватиме з

відкритими інтерфейсами, що можуть бути вразливими для атак. Використання стандартних інструментів безпеки забезпечить захист даних користувачів та системи в цілому.

Усі ці характеристики роблять Ruby on Rails ідеальним вибором для розробки вебсистеми сканування хостів і портів. Завдяки зручності в роботі з базами даних, швидкому створенню API, інтеграції з фоновими задачами та високому рівню безпеки, Rails дозволить реалізувати всі необхідні функції системи з мінімальними витратами часу і зусиль, забезпечуючи при цьому надійність і масштабованість.

Перейдемо до вибору інструментів для клієнтської частини проєкту. Почнемемо з традиційних підходів, таких як використання серверного рендерингу в Rails. У цьому випадку сервер генерує HTML-код на сервері, а потім відправляє його на клієнт. Цей підхід є досить простим для розробки, але він не надає такої гнучкості та швидкості, яку можна отримати з більш сучасними технологіями, такими як клієнтські фреймворки. За допомогою серверного рендерингу важко реалізувати складні взаємодії в реальному часі або швидко оновлювати частини інтерфейсу без перезавантаження сторінки. Це може стати обмеженням, якщо у системі потрібно працювати з великою кількістю даних або здійснювати часті оновлення стану [34].

Ще одним варіантом є використання бібліотеки для побудови інтерфейсу, наприклад, jQuery. Цей інструмент колись був дуже популярним завдяки своїй простоті та можливості маніпулювати DOM. Однак він має кілька суттєвих недоліків. По-перше, jQuery не забезпечує такої організації коду, як сучасні фреймворки. Це може призвести до ускладнень у масштабуванні проекту. По-друге, при використанні jQuery важче реалізувати компонентний підхід до розробки, що ускладнює тестування та підтримку коду. Оскільки наш проект передбачає інтерактивну роботу з даними сканування, ці обмеження можуть стати проблемою [35].

Іншим популярним фреймворком для клієнтської частини є Angular. Angular є потужним фреймворком, який пропонує багато вбудованих функцій, таких як двостороннє зв’язування даних та ін’єкція залежностей. Однак, попри свою потужність, Angular має значну криву навчання і часто є надмірно складним для проектів, де не потрібна вся ця функціональність. Також він може бути дещо важким для налаштування, що може уповільнити процес розробки, особливо якщо команда має обмежений досвід роботи з ним [36].

Нарешті, є Vue.js, який є легким і гнучким фреймворком, що дозволяє швидко створювати динамічні веб-додатки. Vue.js має простіший синтаксис і меншу криву навчання, порівняно з Angular, але все ж не має такої широкої екосистеми і підтримки, як React. Хоча Vue.js може бути хорошим вибором для невеликих проектів, у більш масштабних системах він може мати обмеження щодо кількості доступних інструментів і компонентів, які можуть спростити розробку.

React – це бібліотека для створення інтерфейсів користувача, яка здобула популярність завдяки своїй простоті, гнучкості та здатності працювати з динамічними даними в реальному часі. Вона була розроблена компанією Facebook і дозволяє створювати інтерфейси на основі компонентів, що значно покращує організацію коду і його підтримку в майбутньому [37].

Однією з основних переваг React є використання віртуального DOM. Це дозволяє значно покращити продуктивність порівняно з традиційними підходами, оскільки React може швидко оновлювати лише ті частини інтерфейсу, які змінилися, без необхідності переробляти весь DOM. Для нашого проекту, де потрібно ефективно відображати результати сканування хостів і портів з великими обсягами даних, це є важливим фактором, оскільки забезпечує швидкість і чуйність інтерфейсу навіть при обробці складних запитів.

Ще однією перевагою є підтримка одностороннього потоку даних, що дозволяє зручніше керувати станом програми. У нашому випадку це важливо для правильного відображення результатів сканування, адже кожен елемент

інтерфейсу може залежати від різних даних, які змінюються в реальному часі. React дозволяє чітко визначити, як і коли ці дані мають змінювати стан інтерфейсу, що значно полегшує розробку інтерактивних елементів.

До того ж, React підтримує компонентний підхід до розробки. Це дозволяє розділяти інтерфейс на окремі блоки, кожен з яких має свою відповідальність. Такий підхід значно полегшує тестування та супроводження коду, а також дозволяє швидко масштабувати проект. Для нашого випадку, де інтерфейс повинен бути гнучким і підтримувати велику кількість компонентів (наприклад, таблиці, графіки, сповіщення), це є великою перевагою.

Таким чином, вибір React для клієнтської частини є обґрунтованим завдяки його високій продуктивності, гнучкості, багатій екосистемі та здатності легко обробляти динамічні дані. Всі ці фактори важливі для реалізації вебсистеми сканування хостів і портів для аналізу вразливостей мережі, де необхідно забезпечити швидкий, зручний і інтуїтивно зрозумілий інтерфейс для користувачів.

Підсумовуючи розгляд вибору технологій для створення клієнтської та серверної частини вебсистеми, можна зробити висновок, що поєднання Rails та React є оптимальним рішенням для розробки даного додатку.

Rails на серверній стороні забезпечує швидку розробку, надаючи потужні інструменти для побудови веб-додатків, зокрема завдяки зручній роботі з базами даних, маршрутизацією та моделями. Використання цього фреймворку дозволяє ефективно обробляти запити та взаємодіяти з базою даних, що є важливим для збереження і аналізу результатів сканування мережі.

З іншого боку, React на клієнтській стороні дозволяє створити інтерактивний і динамічний інтерфейс, що здатний ефективно працювати з великими обсягами даних, відображаючи результати сканування хостів і портів у реальному часі. Завдяки своїй здатності до швидкого оновлення лише змінених частин інтерфейсу та використанню компонентного підходу, React забезпечує високу продуктивність і зручність для розробників.

Таким чином, поєднання цих технологій дозволяє створити високопродуктивну та масштабовану вебсистему для сканування хостів і портів, яка буде зручною у використанні для кінцевих користувачів та легко підтримуваною для розробників. Rails і React є ідеальними інструментами для вирішення задач цього проекту, оскільки вони відповідають вимогам до швидкості, інтерактивності та зручності розробки.

2.4 Висновки

У межах даного розділу було здійснено проєктування вебсистеми сканування хостів і портів для аналізу вразливостей мережі. На основі поставлених цілей та задач сформульовано функціональні вимоги, які охоплюють запуск сканувань із заданими параметрами, отримання та перегляд результатів, збереження історії, а також реалізацію зручного користувацького інтерфейсу.

В якості архітектурної моделі обрано клієнт-серверну архітектуру з реалізацією інтерфейсу у вигляді односторінкового додатка (SPA). Такий підхід забезпечує інтерактивність та зручність користування системою. Для асинхронного виконання сканувань інтегровано Sidekiq як фреймворк для обробки фонових задач, а для забезпечення миттєвого зворотного зв’язку з користувачем використовується Pusher, що дозволяє оновлювати інтерфейс у реальному часі.

У процесі вибору технологій для реалізації було обґрунтовано застосування Ruby on Rails як бекенд-фреймворку, який забезпечує зручну роботу з API, авторизацією, чергами та базою даних. Для фронтенд-частини використовується React, що дозволяє створювати динамічний та адаптивний SPA-інтерфейс.

Загалом, результати цього етапу створюють надійну основу для подальшої реалізації системи, забезпечуючи її масштабованість, модульність та зручність в експлуатації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ СКАНУВАННЯ ХОСТІВ ТА ПОРТІВ

3.1 Розробка та опис модулів для сканування

У рамках створення вебсистеми для аналізу вразливостей мережі одним із важливих етапів є розробка модулів для сканування хостів і портів. Ці модулі виконують ключову роль у виявленні потенційних точок входу для зловмисників, виявленні відкритих портів та перевірці доступності мережевих служб.

Основною задачею сканування є швидке і точне виявлення активних хостів, визначення відкритих портів на кожному з них та збирання інформації про можливі вразливості мережі. Для досягнення цієї мети необхідно створити ефективні, масштабовані та надійні алгоритми, які б дозволяли працювати з великою кількістю хостів, забезпечуючи при цьому точність та мінімальні затримки.

Розробка модулів для сканування здійснюється за допомогою мови програмування Ruby, що забезпечує високу гнучкість при написанні мережевих інструментів.

Для початку опишемо модуль для виявлення активних мереж (CurrentNetworks). Даний модуль відповідає за визначення активних мереж, до яких підключений комп'ютер. Для цього він використовує метод Socket.getifaddrs, який дозволяє отримати список мережевих інтерфейсів та їхні властивості. Після отримання цієї інформації, модуль перевіряє наявність IPv4-адрес і маски підмережі у кожного інтерфейсу.

Для кожного активного інтерфейсу модуль обчислює мережеву адресу, застосовуючи побітову операцію AND між IP-адресою та маскою підмережі. Це дозволяє визначити адресу мережі, до якої належить цей інтерфейс. Крім того, обчислюється префікс CIDR для маски підмережі, що дозволяє коректно представляти її у форматі, наприклад, /24.

Модуль збирає всю отриману інформацію про мережі в об'єкти OpenStruct, де зберігаються ім'я інтерфейсу, IP-адреса, маска підмережі, мережеве ім'я та префікс CIDR. Така структура даних є зручним джерелом для подальшого використання в аналізі вразливостей або скануванні мереж.

Оскільки нам потрібно зробити додаток зручним для користувачів, нам потрібен модуль для отримання IP-адрес з різних представлень, наприклад з доменного імені чи маски мережі. Для цього розроблено модуль IPRetrieve, що призначений для обробки вхідних даних у вигляді списку хостів або CIDR-діапазонів та отримання відповідних IP-адрес. Його основна мета – зібрати та повернути IP-адреси для заданих хостів або діапазонів.

Метод retrieve_ips є основним і приймає рядок, що містить список хостів або CIDR-діапазонів, розділених комами. Після цього рядок розбивається на окремі елементи, кожен з яких обробляється через метод process_host.

Метод process_host визначає, чи є переданий хост у форматі CIDR (наприклад, 192.168.0.0/24). Якщо так, викликається метод get_ips_from_cird, який обчислює всі IP-адреси з діапазону. Якщо ж хост не є CIDR-діапазоном, програма намагається отримати його IP-адресу через DNS-запит за допомогою бібліотеки Resolv. Для цього переданий хост спершу перетворюється на URL, якщо це необхідно, а потім для нього виконується DNS-запит.

Якщо хост не вдалося розпізнати або виникла помилка при спробі отримання IP-адреси, метод повертає nil.

Всі отримані IP-адреси для кожного хоста або CIDR-діапазону зберігаються в хеш-таблиці, де ключем є сам хост, а значенням – список IP-адрес, що відповідають цьому хосту або діапазону.

Для підтримки коректності введених даних, метод valid_cidr? перевіряє, чи відповідає вхідна стрічка формату CIDR, що складається з IPv4-адреси та маски підмережі (наприклад, 192.168.0.0/24).

Цей модуль є корисним для тих випадків, коли необхідно швидко отримати IP-адреси з різних джерел, таких як конкретні хости або діапазони IP-адрес у

форматі CIDR, що робить його зручним інструментом для збору даних в процесі аналізу мережі чи вразливостей.

Модуль UpScanner реалізує механізм для перевірки доступності списку IP-адрес через пінг. Основна мета цього модуля – визначити, які хости з наданого списку активні (відповідають на пінг), а які ні.

Конструктор класу приймає три параметри:

– ips: список IP-адрес, які потрібно перевірити;

– timeout: час очікування відповіді на пінг (за замовчуванням — 1 секунда);

– concurrency: кількість одночасно виконуваних перевірок (за замовчуванням — 1024), що дозволяє значно прискорити сканування за рахунок паралельної обробки.

Метод scan ініціює процес сканування. Він розподіляє перевірки IP-адрес серед потоків, використовуючи пул потоків Concurrent::FixedThreadPool. Кожен потік виконує пінг для певного IP-адресу і, в залежності від результату, додає адресу до списку активних (up) або неактивних (down). Після того як всі перевірки завершено, метод повертає хеш з відсортованими списками активних і неактивних IP-адрес.

Для перевірки доступності використовується клас Net::Ping::External, який намагається надіслати пінг і повертає результат. Якщо пінг не вдається, або відбувається інша помилка, метод обробляє виключення і повертає false.

Цей модуль є корисним для швидкої перевірки доступності великої кількості хостів одночасно, що може бути використано для моніторингу мережі або збору даних про стан хостів під час сканування на вразливості.

Модуль PortScanner призначений для перевірки відкритих та закритих портів на заданому хості. Він використовує асинхронне програмування для швидкої обробки великої кількості портів, що значно підвищує ефективність сканування в порівнянні з синхронними підходами.

Конструктор класу приймає три параметри:

– host: IP-адреса або домен хоста, на якому буде виконуватись сканування (за замовчуванням – "127.0.0.1");

– ports: діапазон портів для сканування (за замовчуванням – з 1 по 1024);

– timeout: максимальний час очікування для кожного підключення (за замовчуванням – 0.5 сек).

Модуль використовує асинхронні можливості через бібліотеку Async, що дозволяє виконувати сканування портів одночасно для декількох портів, скорочуючи час на перевірку кожного з них. Для керування кількістю одночасних підключень використовується семафор Async::Semaphore, що обмежує кількість активних потоків.

Метод scan_port здійснює перевірку конкретного порту на наявність відкритого з'єднання за допомогою класу Async::IO::Endpoint.tcp, який створює TCP-з'єднання з портом. Якщо з'єднання успішно встановлено, порт додається до списку відкритих портів. Якщо з'єднання відхилено або тайм-аут, порт додається до списку закритих.

Основна функція, scan, викликає асинхронне сканування через метод async_scan, який запускає всі перевірки для портів у фоновому режимі. Результати сортуються за номерами портів у два списки: відкриті і закриті порти.

Цей модуль дуже корисний для швидкого виявлення відкритих портів на хості, що є основою для подальшого аналізу вразливостей або для виконання аудиту безпеки мережі.

Модуль ServiceScanner здійснює виявлення мережевих сервісів, що працюють на зазначених портах хоста, за допомогою інструмента Nmap. Він дозволяє виявляти, які сервіси доступні на конкретних портах і надавати додаткову інформацію про них, зокрема їхні версії та продукти.

Метод detect_services є основним у цьому класі і виконує сканування портів хоста з використанням Nmap. Метод приймає список портів, які потрібно перевірити, і використовує тимчасовий файл для збереження результатів сканування у форматі XML. Після завершення сканування, цей файл обробляється

за допомогою бібліотеки Nmap::XML, щоб отримати деталі про кожен порт та відповідний сервіс.

Для кожного порту модуль збирає наступні дані:

– порт: номер порту;

– статус: стан порту (відкритий, закритий);

– сервіс: ім'я сервісу, якщо воно виявлено за допомогою Nmap, або визначене за допомогою таблиці COMMON_SERVICES для найбільш поширених портів;

– версія: версія сервісу, якщо вона доступна;

– продукт: продукт, що відповідає за сервіс, якщо ця інформація надана.

Якщо сервіс на порту не визначено, модуль використовує метод detect_service, який співвідносить порти з відомими сервісами за допомогою таблиці COMMON_SERVICES. Якщо порт не знайдений у таблиці, він позначається як "unknown".

Метод detect_services повертає масив хешів, де кожен хеш містить інформацію про один порт і його сервіс.

Цей модуль є корисним для автоматичного сканування доступних сервісів на хості, що може бути використано для аналізу безпеки, виявлення вразливостей або для проведення аудиту мережі.

Наступним опишемо OsScanner, що відповідає за визначення операційної системи на віддалених хостах шляхом сканування з використанням утиліти nmap. Після ініціалізації об'єкта класу OsScanner із вказаним списком IP-адрес, сервіс проводить сканування з увімкненою опцією виявлення операційної системи (-O). Сканування виконується одночасно для всіх зазначених хостів, а результати зберігаються у тимчасовому XML-файлі.

Після завершення сканування, сервіс відкриває цей файл і обробляє результати за допомогою бібліотеки nmap/xml, яка дозволяє зручно зчитувати структуру відповіді. Для кожного знайденого хоста зчитується його IP-адреса, статус (наприклад, "up" чи "down") та найвірогідніша відповідність операційної

системи, яку вдалося ідентифікувати. Якщо визначити ОС не вдалося, результатом буде "Unknown". На завершення, сервіс повертає масив об’єктів з інформацією про кожен просканований хост.

Усі проміжні файли, що використовуються для зберігання результатів сканування, автоматично видаляються після завершення обробки, що гарантує чистоту середовища та безпеку даних.

Модуль IpInfo надає можливість отримати детальну інформацію про IP-адреси, звертаючись до публічного API – ip-api.com, що дозволяє отримувати такі дані, як географічне розташування, провайдер, тип підключення та інші відомості, пов'язані з IP.

Метод get_info є основним і приймає один параметр в виді переліку IP-адрес. Метод виконує HTTP-запити до сервісу ip-api.com для кожної IP-адреси у списку. Для кожної адреси будується URL запиту у вигляді http://ip-api.com/json/#{ip}, і отримана відповідь обробляється як JSON. Якщо запит успішний (отримано код відповіді 200 OK), дані парситься і додаються до результату для відповідної IP-адреси. Якщо ж запит не вдається або виникає інша помилка, замість результату для цієї адреси записується повідомлення про помилку.

Результат повертається у вигляді хешу, де ключами є IP-адреси, а значеннями – інформація про кожну адресу або повідомлення про помилку, якщо дані не вдалося отримати.

Сервіс ip-apі повертає перелік даних про IP-адресу, а саме: країна, в якій знаходиться IP-адреса, місто і штат/область, постачальник інтернет послуг, координати, а також іншу корисну інформацію.

Отже, цей модуль корисний для швидкого збору інформації про IP-адреси, що може бути використано в процесах моніторингу, аудиту безпеки або для аналізу мережевого трафіку.

Всі ці модулі буде об’єднувати один сервіс, який і буде відповідати за наші сканування – ScanJob. Сервіс ScanJob реалізує повний процес сканування мережі

для заданих IP-адрес. Він автоматично обробляє отримані адреси, перевіряє їх активність, виконує сканування портів, виявляє доступні сервіси та збирає додаткову інформацію про хости.

Спочатку сервіс отримує список IP-адрес, які необхідно сканувати, використовуючи метод IpRetrieve.retrieve_ips. Цей метод обробляє введення користувача, яке може містити як окремі IP-адреси, так і CIDR-діапазони. Після цього сервіс перевіряє активність кожної з цих адрес за допомогою UpScanner, визначаючи, які хости є доступними для пінгу та які — ні. Результати перевірки діляться на два списки: активні (up) і неактивні (down) хости.

Для кожного активного хоста сервіс звертається до IpInfo для отримання додаткової інформації, такої як географічне розташування, провайдер Інтернет-послуг, організація, що надає доступ, і інші параметри. Ця інформація записується в базу даних разом зі статусом хоста (активний чи неактивний).

Далі, для кожного активного хоста сервіс виконує сканування портів у межах заданого діапазону за допомогою PortScanner. Після цього визначаються відкриті та закриті порти. Якщо для хоста є відкриті порти, то ServiceScanner використовується для виявлення сервісів, які працюють на цих портах, а також для збору додаткової інформації, такої як версія сервісу та продукт, що працює на порту.

У результаті, для кожного активного хоста будуть створюються записи в базі даних, які містять IP-адресу хоста, статус його доступності, інформацію про сервіси та порти. Якщо порти є закритими, для них створюються записи без інформації про сервіси.

Для неактивних хостів записуються лише дані, що вказують на їхню недоступність. Після того як всі дані оброблено, статус сканування оновлюється на "завершено". Якщо в процесі виникає помилка, статус змінюється на "неуспішно", і вказується причину збою. Отже, саме цей сервіс буде зв’язувати модулі сканування з нашим вебдодатком.

Оскільки ми збираємося реалізовувати заплановані завдання, нам також

потрібно сервіс, відповідальний за дану логіку – ScheduledScanJob. Після перевірки, чи завдання активне і підлягає виконанню, він створює новий запис скануванняі передає його у виконання сервісу ScanJob. Після цього оновлюється інформація про останній і наступний запуск. У разі повторюваного розкладу робота повторно планується на майбутню дату, а одноразові сканування деактивуються після виконання.

Таким чином, ScheduledScanJob автоматизує цикл запуску сканувань і забезпечує безперервну роботу сервісу згідно з заданим розкладом.

Отже, в результаті було розроблено перелік модулів: CurrentNetworks, IPRetreive, UpScanner, PortScanner, ServiceScanner, OsScanner та IPInfo, що будуть використовуватись додатком, а також сервіс ScanJob та ScheduledScanJob, що будуть їх всіх об’єднувати. Кінцевий код кожного з даних сервісів а також діаграму послідовності головного сервісу можна переглянути в додатку А та додатку Б.

3.2 Розробка серверної частини додатку

Ruby on Rails (RoR) — це потужний фреймворк для розробки веб-додатків, який використовує архітектуру MVC (Model-View-Controller). Ця архітектура дозволяє ефективно організувати код та розподілити відповідальність між різними компонентами додатку. У RoR модель (Model) відповідає за взаємодію з базою даних, контролер (Controller) обробляє запити від користувачів та координує роботу моделі і представлення, а представлення (View) відповідає за відображення даних користувачеві. У нашому випадку представлення буде реалізовано за допомогою React, що дозволить створити динамічний інтерфейс, однак це буде детальніше описано в наступному розділі [38].

Однією з основних особливостей Ruby on Rails є вбудована підтримка ActiveRecord, об’єктно-реляційного маппера (ORM), який значно спрощує роботу

з базами даних. ActiveRecord дозволяє працювати з даними через об'єкти, а не писати SQL-запити вручну, що підвищує продуктивність і знижує ймовірність помилок. Кожна модель у RoR є відображенням таблиці в базі даних, а кожен об'єкт цієї моделі відповідає одній запису в таблиці. За допомогою ActiveRecord можна легко виконувати операції з даними, такі як отримання записів, їх оновлення або видалення, не вдаючись до безпосереднього використання SQL [39].

Контролери в RoR обробляють запити, що надходять від користувачів, здійснюють взаємодію з моделями і передають отримані дані до представлень. Вони визначають логіку обробки запитів і формують відповідь для користувача. Використовуючи принципи MVC, RoR дає можливість чітко розподіляти функціональність між різними компонентами, що дозволяє підвищити зручність та ефективність розробки [40].

Однією з важливих особливостей RoR є підтримка міграцій, які дозволяють змінювати структуру бази даних без необхідності вручну писати SQL для кожної зміни. Це дуже зручно на етапі розробки, коли структура даних може змінюватися кілька разів [41].

В цілому, використання Ruby on Rails для розробки серверної частини дозволяє швидко створювати масштабовані та зручні веб-додатки, що забезпечують ефективну взаємодію з базами даних завдяки ActiveRecord, а архітектура MVC допомагає організувати код таким чином, щоб він залишався зрозумілим і зручним для подальшої підтримки.

Першим етапом для розробки серверної частини буде створення бази даних та відповідних моделей. Для нашого проекту використовуватиметься реляційна база даних PostgreSQL, а взаємодія з нею здійснюватиметься через ActiveRecord, що дозволяє ефективно працювати з даними через об'єкти Ruby. Оскільки основна мета вебсистеми – сканування хостів та портів для аналізу вразливостей мережі, структура бази даних відображатиме основні об'єкти системи, такі як сканування, хости та порти.

Схема бази даних складається з чотирьох основних таблиць: scans, scheduled_scans, hosts та ports. Кожна з цих таблиць відповідає за збереження певних даних, необхідних для виконання сканування та подальшого аналізу.

Таблиця scans містить основну інформацію про сканування, таку як вхідні параметри (діапазон портів для сканування), статус виконання, дата завершення сканування та можливі помилки. Ця таблиця також зберігає час створення та оновлення запису.

Таблиця hosts містить інформацію про хости, які були знайдені під час сканування. Для кожного хоста зберігається його IP-адреса, статус (чи доступний хост), доменне ім’я, а також додаткові дані, отримані через API, що дозволяє отримати більше інформації про кожен хост. Кожен запис в цій таблиці пов'язаний з конкретним скануванням, що забезпечується через зовнішній ключ scan_id.

Таблиця ports містить дані про порти, відкриті на хостах. Для кожного порту зберігається його номер, статус (відкритий чи закритий), а також додаткова інформація про сервіс, версію і продукт, який працює на цьому порту. Кожен порт також асоційований з конкретним хостом через зовнішній ключ host_id. Структуру таблиць hosts та ports та їх зв’язок можна переглянути на рисунку 3.1.

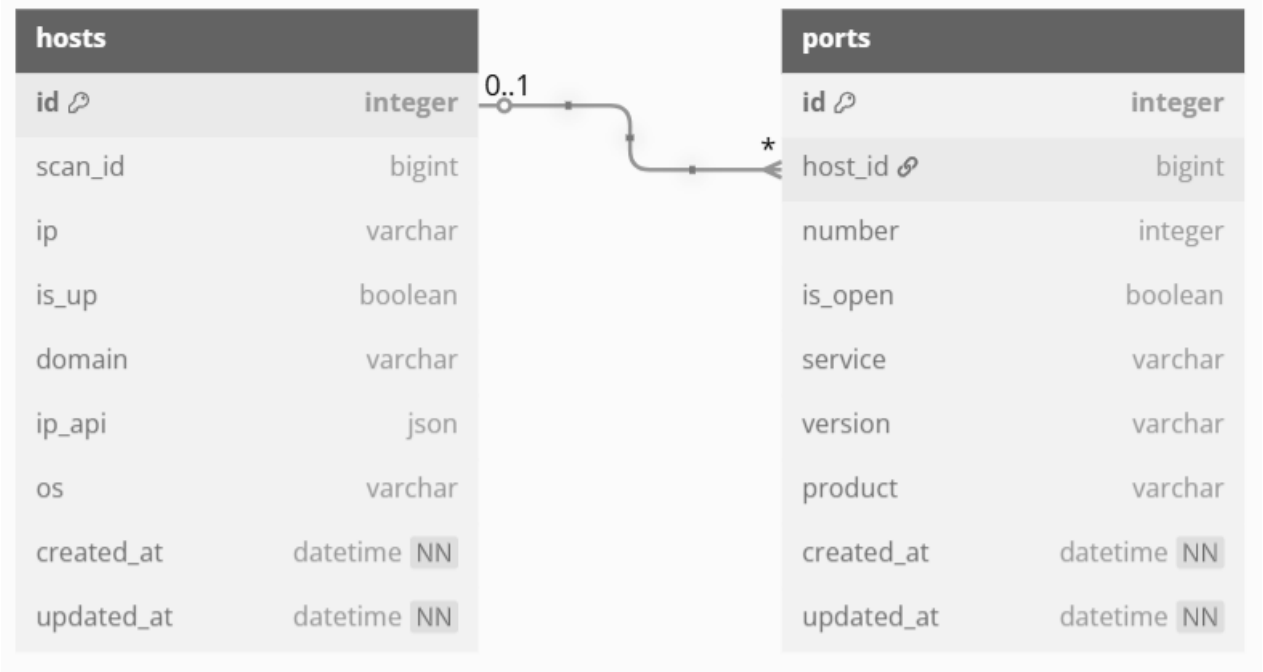


Рисунок 3.1 – Таблиці hosts та ports

Між таблицями встановлені зовнішні ключі: таблиця hosts має зовнішній ключ до таблиці scans, що дозволяє зв'язувати хости з конкретним скануванням, а таблиця ports має зовнішній ключ до таблиці hosts, що дозволяє зв'язати порти з хостами.

Таблиця scheduled_scans відповідає за збереження інформації про заплановані сканування. У ній зберігаються такі параметри, як назва завдання (name), опис, вхідні дані (input, які можуть містити IP-адреси або доменні імена), діапазон портів (start_port, end_port), а також тип розкладу (schedule_type), що визначає, чи буде сканування одноразовим чи повторюваним.

У випадку повторюваного сканування таблиця також зберігає частоту (frequency), значення періодичності (frequency_value), дату початку (start_date) і, за потреби, дату завершення (end_date). Поле active вказує, чи активне завдання, а next_run_at і last_run_at дозволяють відслідковувати, коли відбулося останнє сканування та коли заплановано наступне.

Ця таблиця пов’язана з таблицею scans через зовнішній ключ (неявно реалізований у моделі Scan через поле scheduled_scan_id). Це дає змогу відслідковувати історію запусків для кожного запланованого завдання та забезпечує гнучке керування повторюваними скануваннями. Таким чином, таблиця scheduled_scans відіграє ключову роль у реалізації механізму автоматизації та планування в системі. Таблиці scans та scheduled scans та їх зв’язки можна переглянути на рисунку 3.2.

Для ефективного доступу до даних в базі даних, на кожному з зовнішніх ключів створені індекси. Це дозволяє прискорити пошук та забезпечити швидку роботу з великими обсягами даних, які можуть виникати під час сканування великих мереж.

Таким чином, створення цієї бази даних забезпечить надійне збереження і ефективну обробку даних, що збираються під час сканування хостів і портів. Використання ActiveRecord дозволить зручно взаємодіяти з цією базою через

об'єкти Ruby, що зменшить кількість необхідного коду і спростить обробку запитів до бази даних.

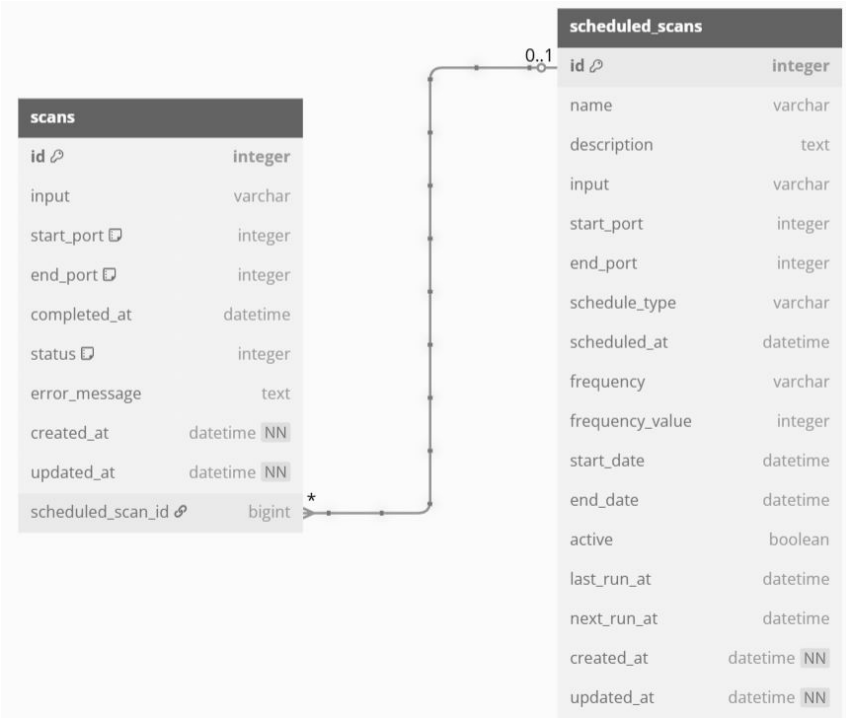


Рисунок 3.2 – Таблиці scans та scheduled_scans

Ця структура бази даних є гнучкою і масштабованою, що дозволяє легко додавати нові функціональні можливості у майбутньому, наприклад, додаткові поля для зберігання інформації про вразливості, або можливість сканування додаткових протоколів.

Для кожної таблиці в базі даних була створена відповідна модель в Ruby on Rails. Модель Scan відповідає за таблицю scans, де зберігаються дані про сканування, такі як параметри діапазону портів, статус виконання і помилки. Модель дозволяє зручно працювати з інформацією про процес сканування та його результати через методи ActiveRecord, не пишучи SQL-запити вручну.

Моделі Host та Port відповідають за таблиці hosts та ports відповідно. Модель Host зберігає дані про хости, знайдені під час сканування, включаючи IP-адресу, статус доступності та додаткову інформацію з API. Модель Port обробляє дані про порти на хостах, такі як номер порту, його статус і інформацію про сервіс.

Зв'язки між моделями (через зовнішні ключі) дозволяють легко маніпулювати даними і забезпечують інтеграцію між таблицями бази даних.

Модель ScheduledScan відповідає за таблицю scheduled_scans і реалізує логіку планування сканувань. Вона дозволяє створювати як одноразові, так і періодичні завдання з використанням типу розкладу (schedule_type) та параметрів повторення (frequency, frequency_value, start_date, end_date).

Модель пов’язана з Scan через зв'язок has_many, що дає змогу зберігати й опрацьовувати історію всіх запусків для конкретного запланованого завдання. Завдяки валідаціям, модель забезпечує коректність вхідних даних, зокрема правильність діапазону портів, наявність дати початку і завершення розкладу, а також перевірку, що одноразове сканування заплановане на майбутнє.

Ключовою особливістю є методи due? та schedule_next_run, які відповідають за визначення актуальності запуску сканування та планування наступного запуску у випадку періодичних задач.

Таким чином, ScheduledScan є центральною частиною логіки автоматичного планування сканувань і підтримки їх регулярності, забезпечуючи гнучкість і контроль над процесом. Всю структури бази даних можна переглянути в додатку Б.

Після того як була створена база даних, варто розглянути інтеграцію створеного сервісу для сканування з користувацьким інтерфейсом, а саме – через API. Для забезпечення взаємодії з користувачем і надання можливості запуску та перегляду результатів сканувань була розроблено відповідний контролер, а саме: ScansController.

ScansController API відповідає за управління запитами, що стосуються сканувань мережі. Він забезпечує можливість ініціювати нові сканування, переглядати інформацію про вже виконані сканування та надавати детальні результати про кожне сканування.

Метод index дозволяє отримати список останніх 10 сканувань, відсортованих за датою створення. Це зручно для швидкого огляду статусу

сканувань, їхніх результатів і часу виконання. Відповідь містить основну інформацію про сканування, таку як його ID, введення (IP-адреси або діапазони) та статус.

Метод show надає детальну інформацію про конкретне сканування за його ID. Він включає в себе дані про хости, що були перевірені в рамках сканування: їх IP-адреси, статус доступності (активні чи неактивні), інформацію з IP-API, а також порти, які були скановані на цих хостах. Для кожного порту зазначається його стан (відкритий чи закритий), а також додаткові дані про сервіс, що працює на порту (наприклад, версія сервісу, тип продукту).

Метод scan відповідає за ініціювання нового сканування. Він створює новий запис у базі даних для кожного сканування, що містить введення (IP-адреси або діапазони) і діапазон портів для сканування. Після цього, через ScanJob, запускається асинхронне виконання сканування. Це дозволяє процесу сканування виконуватись у фоновому режимі, не блокуючи основний потік виконання програми. Після ініціації сканування користувач отримує перенаправлення на попередню сторінку.

Таким чином, ScansController забезпечує зручний доступ до результатів сканування через API, дозволяючи користувачам як переглядати поточні результати, так і запускати нові сканування. Система контролерів ефективно інтегрується з іншими модулями системи, що дозволяє забезпечити високу гнучкість та зручність для кінцевих користувачів.

Іншим важливим контролером є ScheduledScansController є частиною API і відповідно до назви відповідає за обробку CRUD-операцій, пов’язаних із плановими скануваннями. Він дозволяє створювати, переглядати, оновлювати та видаляти об’єкти моделі ScheduledScan. У відповідях повертається також інформація про пов’язані сканування (scans), зокрема їхній статус і дату створення.

Також був розроблений додатковий контролер NetworksController для API, що відповідає за отримання списку активних мереж, що доступні на сервері.

Метод index викликає сервіс CurrentNetworks.get, який збирає інформацію про всі доступні мережі. Для кожної мережі виводяться основні дані, такі як ім'я мережі, IP-адреса хоста, мережевий IP та префікс мережі. Результати формуються в зручний формат JSON, що дозволяє користувачам отримувати список всіх активних мереж.

3.3 Розробка клієнтської частини додатку

Клієнтська частина вебсистеми реалізована з використанням бібліотеки React, яка забезпечує ефективне створення інтерфейсів користувача на основі компонентного підходу. Основною ідеєю React є побудова інтерфейсу як набору незалежних, повторно використовуваних компонентів, кожен з яких описує певну частину сторінки. Компоненти можуть мати власний стан і взаємодіяти між собою через властивості (props), що дозволяє створювати гнучкі та масштабовані інтерфейси.

Компонент App є кореневим компонентом клієнтської частини застосунку. У ньому визначається базова структура маршрутизації за допомогою бібліотеки React Router. Основним контейнером виступає компонент Layout, який забезпечує спільне оформлення сторінок (наприклад, навігацію або заголовок), а всередині нього розміщені маршрути додатку. Зокрема, маршрут з кореневим шляхом ("/") відповідає за головну сторінку (Home), а маршрут "/scans/:id" дозволяє відображати детальну інформацію про певний сеанс сканування (ScanDetails). Такий підхід забезпечує логічну організацію сторінок та полегшує масштабування інтерфейсу.

Компонент Home відповідає за головну сторінку клієнтської частини додатку, яка містить ключову функціональність взаємодії з мережею. Він реалізований як функціональний компонент з використанням хуків useState та useEffect. Стан компонента зберігає список мереж (networks), доступних для

сканування, історію попередніх сканувань (scans) і значення вибраної мережі (selectedNetwork), яке використовується для попереднього заповнення відповідної форми. Графічний вигляд даної сторінки відображено на рисунку 3.1

Після першого завантаження сторінки виконується запит до серверного API для отримання списку мереж і сеансів сканування. Отримані дані передаються до дочірніх компонентів як властивості (props). Наприклад, список мереж передається в NetworkCard, історія сканувань – у ScanHistory, а ScanForm отримує вибрану мережу та функцію, яка оновлює список сканувань після запуску нового сеансу.

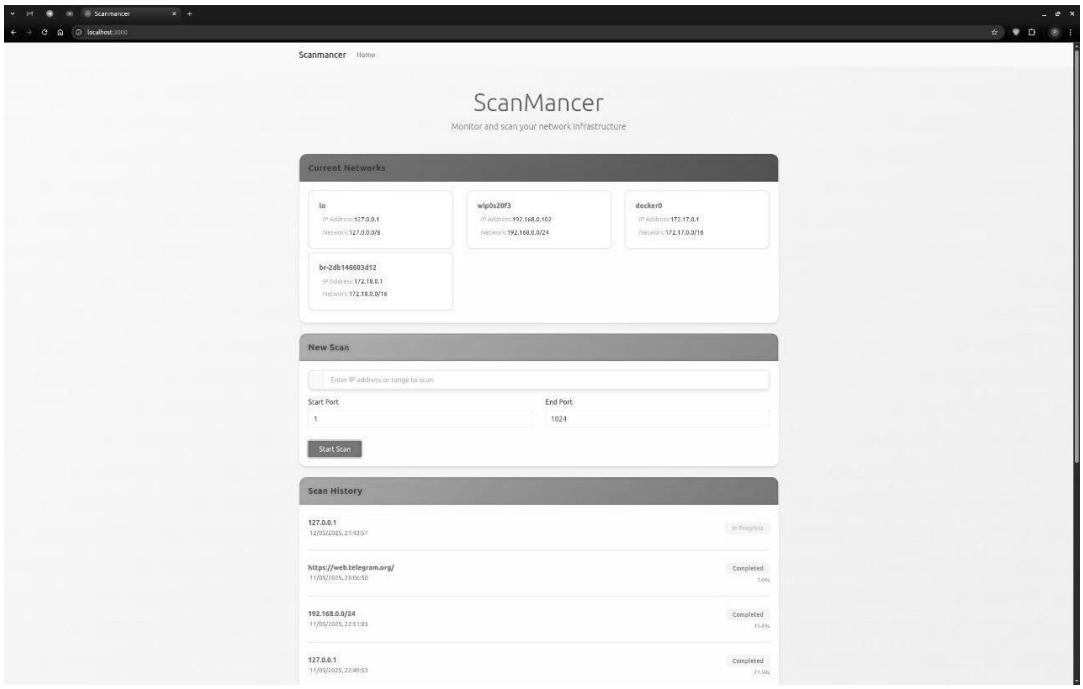


Рисунок 3.3 – Сторінка Home

Таким чином, Home об'єднує в собі логіку завантаження даних, керування станом і представлення інтерфейсу, що дозволяє користувачу переглядати поточні мережі, запускати нові сканування та переглядати історію попередніх сеансів.

Компонент NetworkCard відповідає за візуалізацію окремої мережі у списку доступних для сканування. Він отримує два параметри – об’єкт мережі (network)

та функцію обробки вибору (onSelect), яка викликається при натисканні на IP-адресу або мережеву адресу.

Інтерфейс компонента оформлений у вигляді картки з назвою мережі, її IP-адресою та CIDR-позначенням. Обидва значення подані у вигляді “клікабельних” елементів, які дозволяють швидко підставити відповідну адресу у форму сканування. Це спрощує взаємодію користувача з додатком та зменшує ймовірність помилок при введенні даних.

Завдяки використанню Bootstrap-класів компонент добре адаптується до різних розмірів екранів, а власні стилі та іконки підвищують зручність сприйняття інформації.

Компонент ScanForm реалізує форму для ініціалізації нового сеансу сканування. Його основне призначення – забезпечити користувачу можливість вказати IP-адресу або діапазон адрес, а також портовий інтервал для сканування. Компонент отримує два параметри: preFilledValue, який автоматично заповнює поле IP-адреси при виборі мережі, та onScanComplete – функцію зворотного виклику, яка виконується після успішного завершення запиту.

Форма містить три поля: адреса хоста або підмережі (input), початковий порт (startPort) і кінцевий порт (endPort). Для зручності користувача поля портів мають обмеження в межах допустимого діапазону TCP/UDP (1–65535). Значення preFilledValue передається через пропси та синхронізується з локальним станом за допомогою хука useEffect.

Після натискання кнопки відправлення запиту (Start Scan) форма виконує POST-запит на сервер за маршрутом /scan, передаючи введені параметри у форматі JSON. У разі успішної відповіді викликається функція onScanComplete, яка оновлює список сканувань і очищає поле вводу. Усі запити захищені CSRF-токеном, що отримується з мета-тегу HTML-документу.

Компонент ScanHistory відповідає за відображення історії сканувань у вигляді списку. Він приймає масив об’єктів scans як параметр та для кожного сеансу формує окремий блок із короткою інформацією. Основний акцент

робиться на ідентифікації сеансу за введеною IP-адресою або діапазоном, часом запуску та поточним статусом.

Кожен сеанс сканування містить посилання на детальну сторінку, доступну за унікальним ідентифікатором. Відповідно до статусу сканування (completed, failed, processing, pending) виводиться інформативна мітка з піктограмою та, за потреби, додатковими деталями. Якщо сканування завершено, також вказується тривалість у секундах. У випадку помилки поруч з часом відображається повідомлення з поясненням.

Завдяки умовному рендерингу та структурованій верстці компонент наочно передає стан кожного сканування й дозволяє швидко перейти до перегляду результатів або виявлення проблем.

Компонент ScanDetails відповідає за відображення детальної інформації про конкретний сеанс сканування, що дозволяє користувачеві переглядати результати аналізу мережі. Після завантаження даних про сканування за його ідентифікатором з API, компонент відображає загальну інформацію про скан: введені дані, статус, час початку та завершення, а також тривалість і повідомлення про помилки, якщо вони є. Графічне відображення даної сторінки відображено на рисунку 3.2

Компонент також дозволяє переглядати знайдені хости, розділені на активні (Up) та неактивні (Down). Користувач може вибрати хост зі списку для відображення більш детальної інформації, яка включає статус хоста, IP-інформацію, відкриті та закриті порти, а також посилання на зовнішні ресурси для перевірки вразливостей, якщо вони доступні.

Основна логіка компонента полягає в завантаженні даних через API за допомогою useEffect і в керуванні станом компонента через хуки useState. Дані про порти аналізуються для підрахунку кількості відкритих і закритих портів, а для кожного порту генеруються посилання на можливі вразливості, що зберігається в базі даних зовнішніх ресурсів, таких як CVE Details або NVD.

Компонент також підтримує функціональність для показу або приховування закритих хостів і портів, щоб зробити інтерфейс більш гнучким і зручним для користувача.

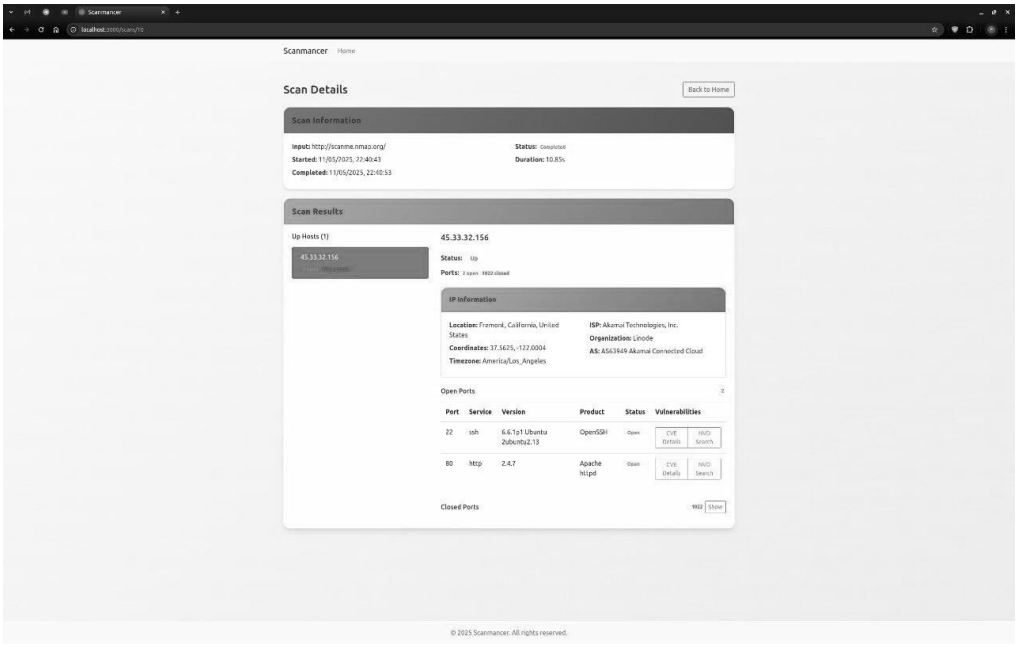


Рисунок 3.4 – Сторінка Scan Details

Компонент ScheduledScans реалізує функціонал управління запланованими скануваннями. Його основне призначення – надавати користувачу можливість створювати сканування, які будуть запускатися в майбутньому: один раз у конкретний час або періодично за обраним графіком. Вигляд даної сторінки можна переглянути на рисунку 3.3.

Компонент звертається до API для отримання списку наявних запланованих сканувань, сортує їх за полем next_run_at і відображає у вигляді таблиці з основною інформацією: назва, опис, ціль, тип розкладу, час наступного запуску, статус та можливість видалення.

Форма створення сканування дозволяє вводити ціль, порти, тип графіка, дату запуску або періодичність, і надсилає ці дані на сервер. Для одноразових сканувань обов’язковим є поле дати запуску. Для періодичних – частота, дата

початку та, за потреби, дата завершення. Перед відправкою значення дат переводяться у формат UTC.

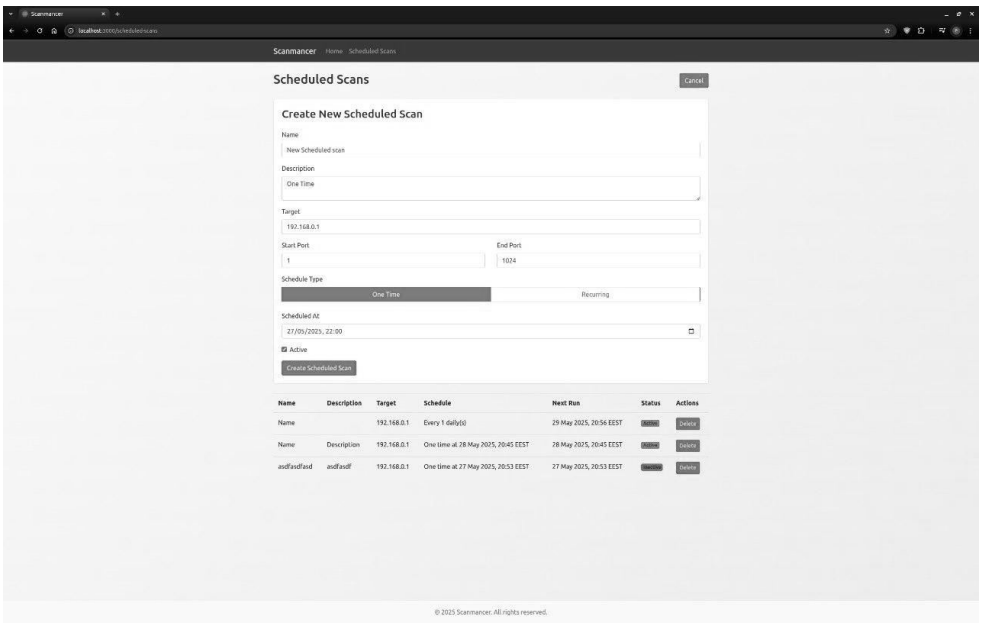


Рисунок 3.5 – Сторінка Scheduled Scans

У результаті компонент забезпечує зручний інтерфейс для створення та адміністрування розкладу сканувань у системі.

Таким чином було розроблено достатньо зручні сторінки для виконання та відображення вже існуючих сканувань, що дозволить швидко орієнтуватися в інтерфейсі та взаємодіяти з додатком.

3.4 Тестування додатку

Розроблена вебсистема сканування хостів і портів проходила тестування з метою перевірки її функціональності, точності виявлення активних вузлів і відкритих портів, а також оцінки стабільності роботи в умовах реального використання. Окрему увагу було приділено порівнянню продуктивності розробленого рішення з існуючим інструментом командного рядка Nmap, що

дозволило оцінити ефективність реалізованих алгоритмів сканування в межах веборієнтованої архітектури.

Для прикладу, було перевірено незалежний модуль UpScanner, для перевірки чи піднятий хост. Тестування було проведено за допомогою середовища консолі Ruby on Rails. Скануvалася підмережа 192.168.0.0/24, яка містить кілька активних вузлів. Результати запуску модуля можна переглянути на рисунку 3.6.

```
3.3.7 :034 > begin
3.3.7 :035 >   before = Time.current
3.3.7 :036 >   ips = IpRetrieve.retrieve_ips("192.168.0.0/24").values.flatten
3.3.7 :037 >   result = UpScanner.new(ips: ips).scan
3.3.7 :038 >   after = Time.current
3.3.7 :039 >   puts "Time in seconds: #{after - before}"
3.3.7 :040 >   puts "Available hosts: #{result[:up]}"
3.3.7 :041 > end
Time in seconds: 1.132701036
Available hosts: ["192.168.0.1", "192.168.0.101", "192.168.0.102"]
```

Рисунок 3.6 – Сканування активних хостів за допомогою UpScanner.

У результаті сканування виявлено три активних хости: "192.168.0.1", "192.168.0.101" та "192.168.0.102". Час виконання становив приблизно 1.13 секунди, що є досить прийнятним для мережевого сканування в локальній підмережі.

Для порівняння було використано аналогічну команду в Nmap, яка відповідає за виявлення доступних хостів без сканування портів. Для прискорення сканування було використано ключі -n (відключення DNS) та -T4 (підвищення агресивності сканування). Результат виконання можна переглянути на рисунку 3.7.

```
softer@softer-Legion-5-Pro-16ITH6H:~/Univer/course/scanmancer$ nmap -sn -n -T4 192.168.0.1/24
Starting Nmap 7.80 ( https://nmap.org ) at 2025-05-28 16:03 EEST
Nmap scan report for 192.168.0.1
Host is up (0.00097s latency).
Nmap scan report for 192.168.0.101
Host is up (0.000073s latency).
Nmap done: 256 IP addresses (2 hosts up) scanned in 2.50 seconds
```

Рисунок 3.7 – Сканування активних хостів за допомогою nmap.

У цьому тесті nmap виявив два активні хости – 192.168.0.1 і 192.168.0.101 – за 2.50 секунди. Різниця в кількості виявлених хостів може бути спричинена різними методами перевірки доступності. nmap використовує комплексний набір мережевих запитів, які можуть бути блоковані на деяких хостах або фільтрами мережі.

Таким чином, розроблений модуль демонструє досить високу швидкість роботи та адекватну точність у виявленні активних хостів у локальній мережі. Це підтверджує його практичну цінність для інтеграції у вебсистему сканування.

Доцільно також перевірити швидкість сканування всієї системи вцілому. Було виконано повне сканування одного хоста – 192.168.0.1. Загальна тривалість виконання сканування становила 10.04 секунди, що включало виявлення активності хоста, перевірку відкритих портів.

Хост було визначено як активний (Status: Up). Модулем визначення операційної системи встановлено, що пристрій працює на базі ядра Linux версій 2.6.23 – 2.6.38. У результаті сканування портів було перевірено 1024 TCP-порти, з яких 2 порти виявлено відкритими, решта – закриті.

Відкритими виявлено порти:

– 22 (ssh) – працює служба Dropbear sshd, версія 2012.55;

– 80 (http) – ідентифіковано інтерфейс конфігурації TP-LINK WR941ND (WAP http config).

Крім того, було здійснено повне сканування підмережі 192.168.0.0/24. У процесі перевірки система автоматично виявила 2 активні хости – 192.168.0.1 та 192.168.0.101. Загальна тривалість сканування становила 20.27 секунд. Як і у випадку з окремим хостом, система виконала послідовно виявлення активних IP-адрес, сканування портів, ідентифікацію операційної системи та аналіз служб.

Перший хост 192.168.0.1 так само мав відкриті порти 22 (ssh) та 80 (http). Другий хост 192.168.0.101 мав відкритим лише порт 80 (http). У межах цього порту працював HTTP-сервер, однак у цьому випадку не вдалося визначити точну версію або продукт, що стоїть за цим портом. Система також успішно визначила,

що обидва хости працюють на базі Linux (із зазначеним діапазоном версій для 192.168.0.1). Результат сканування підмережі можна переглянути на рисунку 3.8.

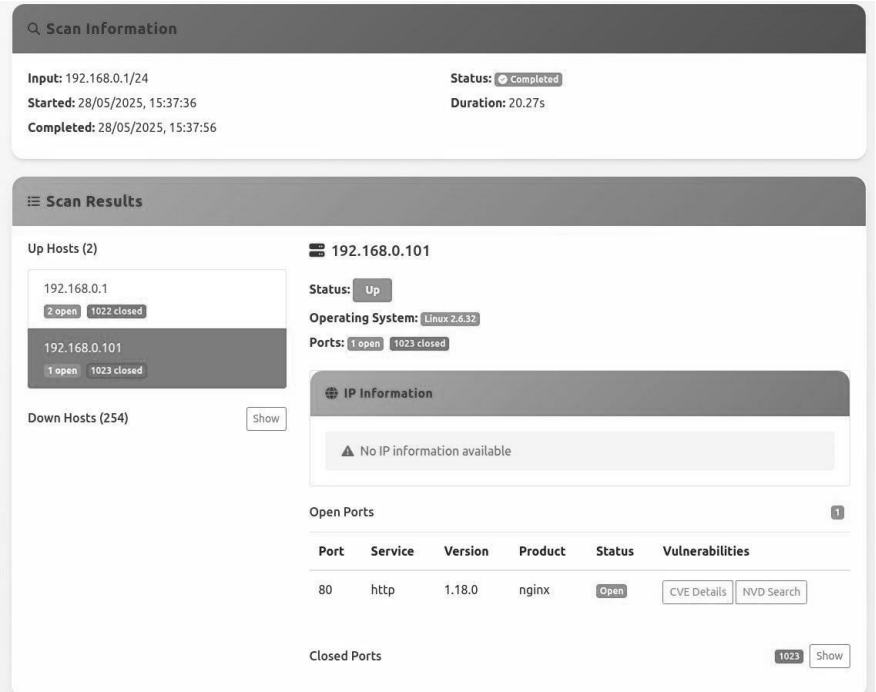


Рисунок 3.8 – Результат сканування підмережі

Попри збільшення обсягу обробки, час виконання сканування залишився прийнятним для локального аналізу, що свідчить про достатню ефективність реалізованого механізму навіть у межах повної підмережі.

Для порівняння було використано команду nmap з параметрами -p1-1024 -sV -O, яка виконала аналогічне сканування за 9 секунд, яке дало такі ж результати.

При перевірці всієї підмережі, команда nmap з тими ж параметрами виявила три активні хости – 192.168.0.1, 192.168.0.101 та додатково 192.168.0.100, який мав усі 1024 порти у фільтрованому стані. Повна перевірка зайняла 29.14 секунди. Збіг результатів між двома інструментами щодо ідентифікованих сервісів, портів та ОС також був на високому рівні. Таким чином, власна система показала не лише сумісність результатів із загальновизнаним інструментом nmap, але й виграш у швидкості виконання сканувань. Результат сканування підмережі за допомогою nmap можна переглянути на рисунку 3.9.

```
softer@softer-Legion-5-Pro-16ITH6H:~/Univer/course/scanmancer$ sudo nmap -p1-1024 -sV -O 192.168.0.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2025-05-28 16:28 EEST
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.0021s latency).
Not shown: 1022 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     Dropbear sshd 2012.55 (protocol 2.0)
80/tcp open  http    TP-LINK WR941ND WAP http config
MAC Address: 84:16:F9:C7:95:8E (Tp-link Technologies)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.23 - 2.6.38
Network Distance: 1 hop
Service Info: OS: Linux; Device: WAP; CPE: cpe:/o:linux:linux_kernel, cpe:/h:tp-link:wr941nd

Nmap scan report for 192.168.0.100
Host is up (0.031s latency).
All 1024 scanned ports on 192.168.0.100 are filtered
MAC Address: 04:27:28:C3:71:9E (Unknown)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

Nmap scan report for softer-Legion-5-Pro-16ITH6H (192.168.0.101)
Host is up (0.000028s latency).
Not shown: 1023 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.18.0 (Ubuntu)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
Network Distance: 0 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 256 IP addresses (3 hosts up) scanned in 29.14 seconds
```

Рисунок 3.9 – Результат сканування підмережі за допомогою nmap

На основі даних сканувань складемо діаграму, яка відображатиме швидкодію нашого додатку та nmap, яку можна переглянути на рисунку 3.10

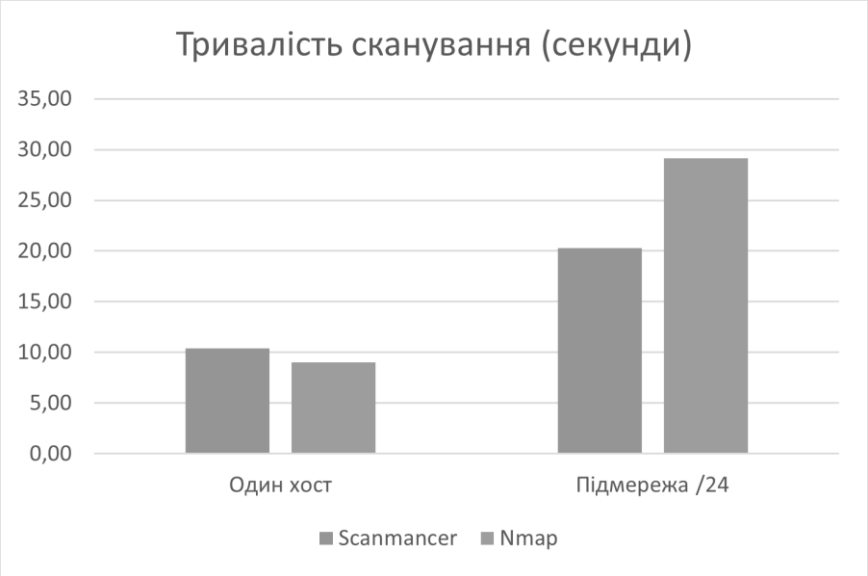


Рисунок 3.10 – Діаграма порівняння швидкодії

У результаті проведеного тестування можна зробити висновок, що розроблений сервіс показав цілком достойні результати. В окремих випадках, зокрема при скануванні підмережі, він навіть продемонстрував кращу швидкодію

порівняно з nmap. Водночас загальна продуктивність залежить від низки чинників – кількості хостів, якості з’єднання, типу цільових пристроїв та специфіки реалізації кожного інструменту. Тому остаточна ефективність може змінюватися в залежності від конкретного середовища використання.

3.5 Висновки

У цьому розділі було здійснено безпосередню реалізацію основних компонентів вебсистеми сканування хостів та портів, розробленої на основі попереднього етапу проєктування.

На першому етапі реалізовано модулі для сканування, які забезпечують виконання різних типів перевірок. Модулі інтегровано із системою через фонові задачі Sidekiq, що дозволяє запускати сканування асинхронно без блокування основного потоку програми. Передбачена гнучкість у параметризації запитів, що забезпечує підтримку різних режимів аналізу мережі.

У рамках серверної частини було реалізовано RESTful API на базі Ruby on Rails, що забезпечує обробку запитів клієнтської частини, керування чергою задач, збереження результатів сканування у базі даних, а також авторизацію та автентифікацію користувачів. Реалізація взаємодії між компонентами відбувається з урахуванням вимог безпеки та розширюваності.

Клієнтська частина додатку реалізована з використанням React, що дозволило створити сучасний односторінковий інтерфейс (SPA). Забезпечено інтерактивність, миттєве оновлення статусу сканувань за допомогою сервісу Pusher, а також зручну навігацію та візуалізацію результатів. Особлива увага приділена ергономіці інтерфейсу та простоті взаємодії користувача із системою.

Таким чином, реалізовано повнофункціональну програмну систему, яка відповідає поставленим вимогам та забезпечує ефективне сканування мережевих хостів і портів через зручний вебінтерфейс.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено вебсистему сканування хостів і портів для аналізу вразливостей мережі. Реалізоване рішення дозволяє здійснювати базову мережеву розвідку та виявлення потенційно вразливих сервісів на відкритих портах віддалених хостів або локальної системи.

У процесі роботи було проведено:

– аналіз існуючих засобів сканування мереж і визначення їхніх переваг та недоліків;

– проєктування архітектури вебсистеми, яка включає модуль сканування, вебінтерфейс користувача та базу даних для зберігання результатів;

– реалізацію функціоналу сканування з використанням мови програмування Ruby та фреймворку Ruby on Rails;

Розроблена система забезпечує:

– сканування хостів за вказаними IP-адресами або доменними іменами;

– виявлення відкритих портів та ідентифікацію сервісів, що на них працюють;

– збереження результатів сканування в базі даних для подальшого аналізу;

– зручний інтерфейс для перегляду результатів та керування скануванням.

Таким чином, поставлені завдання були успішно виконані, а мета роботи – створення вебсистеми для сканування хостів і портів – досягнута. Система може бути використана в якості допоміжного інструменту для проведення аудиту мережевої безпеки та попереднього аналізу вразливостей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Макнаб, К. Network Security Assessment: Know Your Network. – 3rd ed. – Beijing: O'Reilly Media, 2016. 464 с.
2. What is a Port Scanner and How Does it Work? Varonis. URL: https://www.varonis.com/blog/port-scanning-techniques (дата звернення: 18.03.2025)
3. What is CVE? Red hat. URL: https://www.redhat.com/en/topics/security/what-is-cve (дата звернення: 04.05.2025)
4. Types of Nmap scans and best practices. TechTarget. URL: https://www.techtarget.com/searchnetworking/tip/Types-of-Nmap-scans-and-best-practices (дата звернення: 18.03.2025)
5. What are Iaas, Paas, Saas and CaaS? Trend Micro. URL: https://www.trendmicro.com/en_gb/what-is/cloud-security/iaas-paas-saas.html (дата звернення: 04.05.2025)
6. Container Scanning. Gitlab Docs. URL: https://docs.gitlab.com/user/application_security/container_scanning/ (дата звернення: 04.05.2025)
7. What is a Network Security Audit? Types & Importance. SentinelOne. URL: https://www.sentinelone.com/cybersecurity-101/cybersecurity/network-security-audit/ (дата звернення: 08.05.2025)
8. Top 6 most widely used port scanners in cybersecurity. Netlas.io. URL: https://netlas.io/blog/port_scanner_in_cybersecurity/ (дата звернення: 22.03.2025)
9. What is Nmap and how to use it. Freecodecamp. URL: https://www.freecodecamp.org/news/what-is-nmap-and-how-to-use-it-a-tutorial-for-the-greatest-scanning-tool-of-all-time/ (дата звернення: 26.03.2025)
10. Nmap Scripting Engine. Nmap.org. URL: https://nmap.org/book/nse.html (дата звернення: 08.05.2025)
11. Zenmap. Nmap.org. URL: https://nmap.org/zenmap/ (дата звернення: 12.05.2025)

12. How masscan works. Rushter. URL: https://rushter.com/blog/how-masscan-works/ (дата звернення: 26.03.2025)

13. Masscan - 1000 Times Faster Than Nmap. Medium. URL: https://medium.com/@gayuuinfocerts/masscan-1000-times-faster-than-nmap-1459012b60c5 (дата звернення: 15.05.2025)

14. OpenVAS: Vulnerability Scanning. URL: https://belcyber.medium.com/openvas-vulnerability-scanning-installation-guide-f1d561337cd (дата звернення: 26.03.2025)

15. OpenVAS vs Nessus: Top Vulnerability Scanners Compared. Datamation. URL: https://www.datamation.com/security/openvas-vs-nessus/ (дата звернення: 15.05.2025)

16. OWASP ZAP: A Comprehensive Guide to Web Application Security Testing. Medium. URL: https://medium.com/@1200km/owasp-zap-a-comprehensive-guide-to-web-application-security-testing-6c247f4be39b (дата звернення: 30.03.2025)

17. Automate ZAP. OWASP ZAP Documentation. URL: https://www.zaproxy.org/docs/automate/ (дата звернення: 15.05.2025)

18. How do I handle a False Positive? OWASP ZAP Documentation. URL: https://www.zaproxy.org/faq/how-do-i-handle-a-false-positive/ (дата звернення: 15.05.2025)

19. The ZMap Project. Zmap.io. URL: https://zmap.io/about (дата звернення: 30.03.2025)

20. Брукс, Ч. Дж., Гроу, К., Крейг, Ф., Шорт, Д. Cybersecurity Essentials. – Hoboken: Wiley, 2018. 528 с.

21. What is the Main Goal of Penetration Testing? Cyber Defense Magazine. URL: https://www.cyberdefensemagazine.com/what-is-the-main (дата звернення: 17.05.2025)

22. OS Fingerprinting. Firewalls. URL: https://www.firewalls.com/blog/security-terms/os-fingerprinting/ (дата звернення: 08.05.2025)

23. Client-Server Model. GeeksforGeeks. URL: https://www.geeksforgeeks .org/client-server-model/ (дата звернення: 17.05.2025)

24. JSON API: Best Practices. Botpenguin URL https://botpenguin.com/glossary /json-api (дата звернення: 17.05.2025)

25. What is a Single Page Application? Kadiska. URL: https://kadiska.com/what-is-a-single-page-application-spa/ (дата звернення: 30.03.2025)

26. WebsSockets. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 17.05.2025)

27. Scalable Real-time Communication With Pusher. Velotio. URL: https://www.velotio.com/engineering-blog/scalable-real-time-communication-with-pusher (дата звернення: 04.04.2025)

28. The ultimate guide to Sidekiq scheduled jobs. Honeybadger. URL: https://www.honeybadger.io/blog/sidekiq-background-jobs/ (дата звернення: 08.04.2025)

29. Best NodeJS frameworks for seamless backend development. Ably. URL: https://ably.com/blog/best-nodejs-frameworks (дата звернення: 08.04.2025)

30. What is Django? IBM. URL: https://www.ibm.com/think/topics/django (дата звернення: 08.04.2025)

31. What is Spring Boot? IBM. URL: https://www.ibm.com/think/topics/django (дата звернення: 12.04.2025)

32. Laravel – Overview. Tutorialspoint. URL: https://www.tutorialspoint.com/laravel/laravel_overview.htm (дата звернення: 12.04.2025)

33. Ruby on Rails Projects: Introduction for Beginners. Microverse. URL: https://www.microverse.org/blog/ruby-on-rails-projects-introduction-for-beginners (дата звернення: 16.04.2025)

34. Layout and Rendering in Rails. RubyOnRails. URL: https://guides. rubyonrails.org/layouts_and_rendering.html (дата звернення: 16.04.2025)

35. Is JQuery Still Relevant? Geeksforgeeks. URL: https://www.geeks forgeeks.org/is-jquery-still-relevant/ (дата звернення: 20.04.2025)

36. What is Angular? Angular. URL: https://www.geeksforgeeks.org/is-jquery-still-relevant/ (дата звернення: 20.04.2025)

37. Overview of React.js Patterns.dev. URL: https://www.patterns.dev/react/ (дата звернення: 20.04.2025)

38. Що таке MVC? Rubydevelopers. URL: https://rubydevelopers.org/t/mvc/59 (дата звернення: 01.05.2025)

39. Understanding ActiveRecord in Ruby on Rails. Medium. URL: https://bhartee-tech-ror.medium.com/understanding-activerecord-in-ruby-on-rails-b11f3d3a3061 (дата звернення: 01.05.2025)

40. Ruby on Rails – Controller. Tutorialspoint. URL: https://www.tutorialspoint.com/ruby-on-rails/rails-controllers.htm (дата звернення: 01.05.2025)

41. Migrations. Rubyonrails Guides. URL: https://guides.rubyonrails.org/v2.3/migrations.html (дата звернення: 04.05.2025)

ДОДАТОК А

Код програми

```
class IpInfo
  require 'net/http'
  require 'json'
  def self.get_info(ips)
    return {} if ips.empty?
    results = {}
    ips.each do |ip|
      begin
        uri = URI("http://ip-api.com/json/#{ip}")
        response = Net::HTTP.get_response(uri)
        if response.is_a?(Net::HTTPSuccess)
          data = JSON.parse(response.body)
          results[ip] = data
        else
          results[ip] = { error: "Failed to fetch data" }
        end
      rescue StandardError => e
        results[ip] = { error: "Failed to fetch data" }
      end
    end
    results
  end
end
class IpRetrieve
  class << self
    def retrieve_ips(input)
      values = input.split(",") if input.is_a?(String)
      (values || []).map(&:strip).reduce({}) do |res, host|
        res[host] = process_host(host)
        res
      end
```

```
    end
    def process_host(host)
      return get_ips_from_cird(host) if valid_cidr?(host)
      host = try_get_host_from_url(host)
      [ Resolv.getaddress(host.to_s) ]
    rescue Resolv::ResolvError
      nil
    end
    private
    def try_get_host_from_url(host)
      uri = URI.parse(host)
      uri.host || host
    rescue URI::InvalidURIError
      host
    end
    def get_ips_from_cird(host)
      IPAddr.new(host).to_range.to_a.map(&:to_s)
    rescue IPAddr::InvalidAddressError
      nil
    end
    def valid_cidr?(value)
      value.match?(/^\d{1,3}(\.\d{1,3}){3}\/\d{1,2}$/)
    end
  end
end
require 'nmap/xml'
require 'tempfile'
class OsScanner
  def initialize(hosts)
    @hosts = Array(hosts)
  end
  def scan
    results = []
    temp_file = Tempfile.new(['nmap_scan', '.xml'])
```

```
    begin
      # Run nmap with privileged flag for all hosts at once
      hosts_list = @hosts.join(' ')
      system("nmap --privileged -O -v #{hosts_list} -oX #{temp_file.path}")
      Nmap::XML.open(temp_file.path) do |xml|
        xml.each_host do |host|
          # Get the most accurate OS match
          os_name = host.os&.matches&.first&.name || "Unknown"
          host_result = {
            ip: host.ip,
            status: host.status.state,
            os: os_name
          }
          results << host_result
        end
      end
    ensure
      temp_file.close
      temp_file.unlink
    end
    results
  end
end
require "async/io"
require "async/await"
require "async/semaphore"
class PortScanner
  include Async::Await
  include Async::IO
  def initialize(host: "127.0.0.1", ports: (1..1024), timeout: 0.5)
    @host = host
    @ports = ports
    @timeout = timeout
    @semaphore = Async::Semaphore.new(1024)
```

```
    @results = nil
  end
  def scan_port(port)
    Timeout.timeout(@timeout) do
      Async::IO::Endpoint.tcp(@host, port).connect do |peer|
        peer.close
        @results[:open] << port
      end
    end
  rescue Errno::ECONNREFUSED, Timeout::Error
    @results[:closed] << port
  rescue Errno::EMFILE
    sleep @timeout
    retry
  end
  def scan
    async_scan.wait
  end
  private
  async def async_scan
    @results = { open: [], closed: [] }
    tasks = @ports.map do |port|
      @semaphore.async do
        scan_port(port)
      end
    end
    tasks.each(&:wait)
    @results[:open].sort!
    @results[:closed].sort!
    @results
  end
end
require 'nmap/command'
require 'nmap/xml'
```

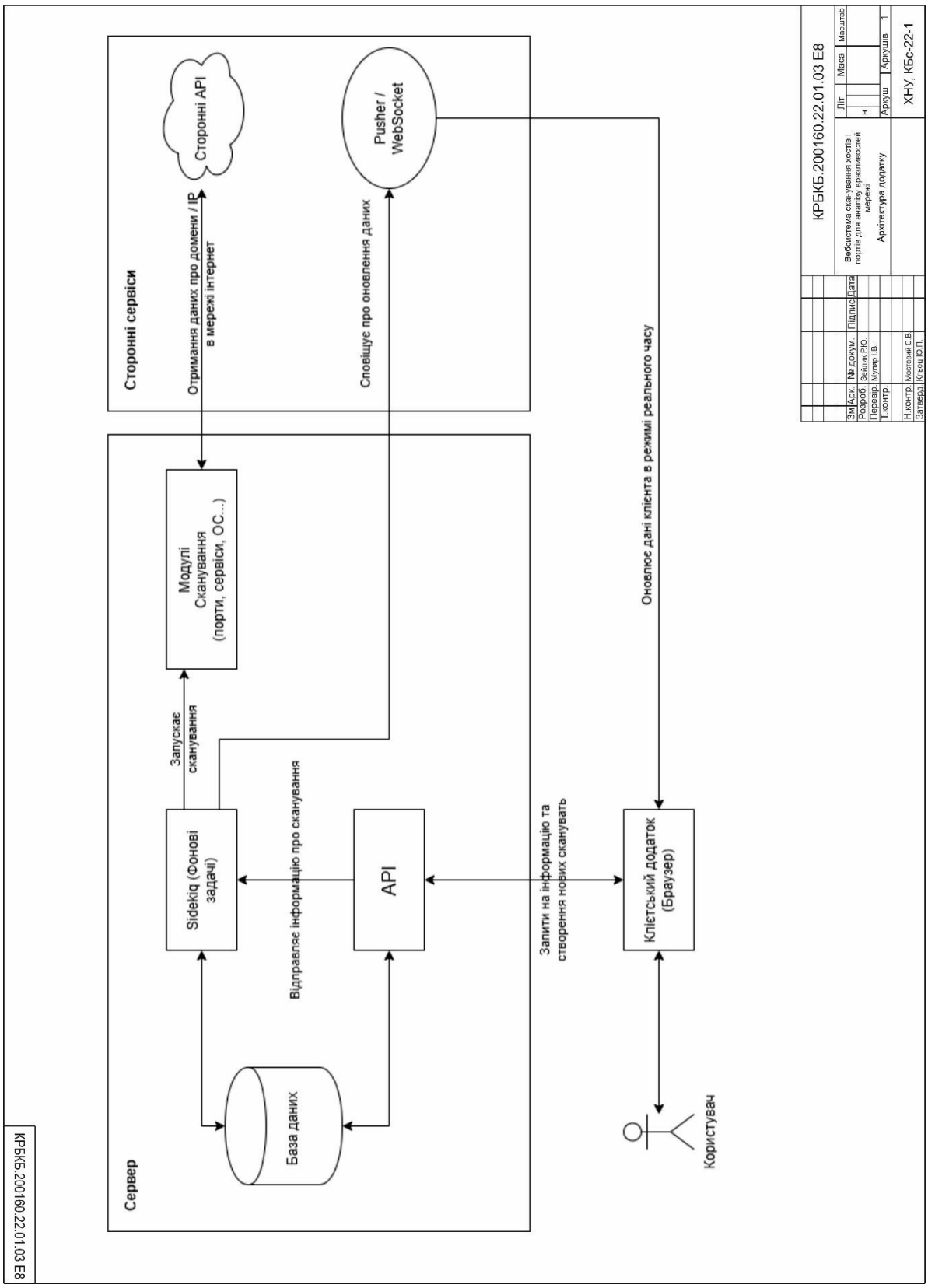
```
require 'tempfile'
class ServiceScanner
  COMMON_SERVICES = {
    21 => 'FTP',
    22 => 'SSH',
    23 => 'Telnet',
    25 => 'SMTP',
    53 => 'DNS',
    80 => 'HTTP',
    110 => 'POP3',
    143 => 'IMAP',
    443 => 'HTTPS',
    3306 => 'MySQL',
    5432 => 'PostgreSQL',
    27017 => 'MongoDB',
    6379 => 'Redis'
  }.freeze
  def initialize(host)
    @host = host
  end
  def detect_services(ports)
    results = []
    temp_file = Tempfile.new(['nmap_scan', '.xml'])
    begin
      Nmap::Command.run do |nmap|
        nmap.service_scan = true
        nmap.verbose = true
        nmap.ports = ports
        nmap.targets = @host
        nmap.output_xml = temp_file.path
      end
      Nmap::XML.open(temp_file.path) do |xml|
        xml.each_host do |host|
          host.each_port do |port|
```

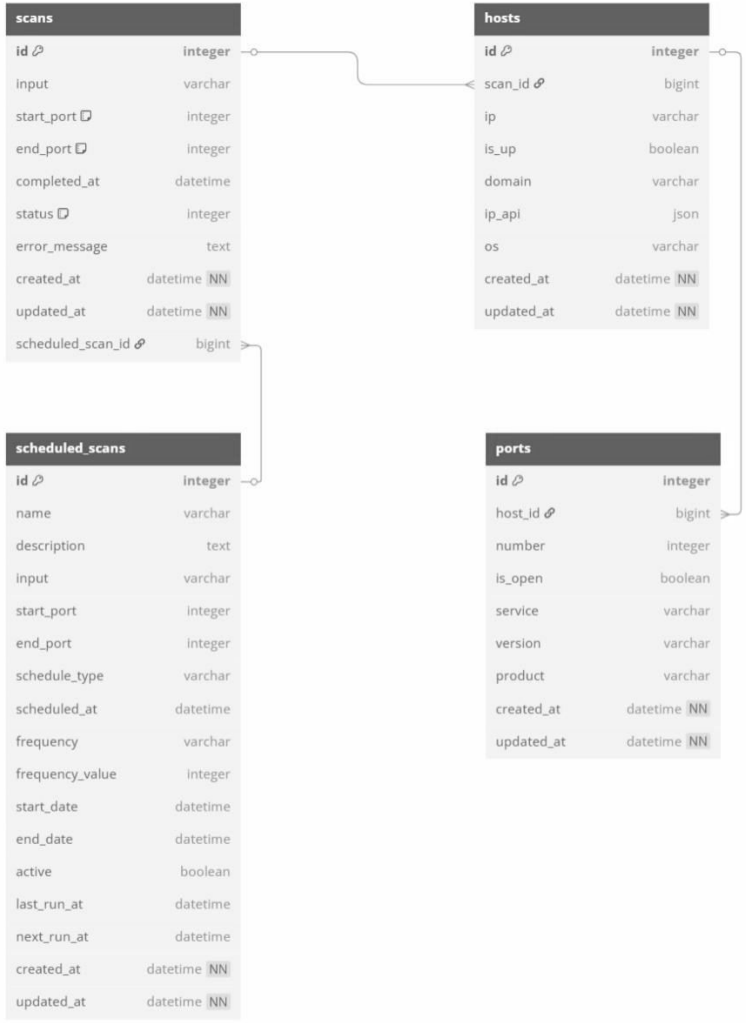
```
            results << {
              port: port.number,
              status: port.state,
              service: port.service&.name || detect_service(port.number),
              version: port.service&.version,
              product: port.service&.product
            }
          end
        end
      end
    ensure
      temp_file.close
      temp_file.unlink
    end
    results
  end
  private
  def detect_service(port)
    COMMON_SERVICES[port] || 'unknown'
  end
end
require 'net/ping'
require 'concurrent-ruby'
class UpScanner
  def initialize(ips:, timeout: 1, concurrency: 1024)
    @ips = ips
    @timeout = timeout
    @pool = Concurrent::FixedThreadPool.new(concurrency)
  end
  def scan
    result = {
      up: [],
      down: []
    }
```

```
    @ips.each do |ip|
      @pool.post do
        if is_up(ip)
          result[:up] << ip
        else
          result[:down] << ip
        end
      end
    end
    @pool.shutdown
    @pool.wait_for_termination
    result[:up].sort!
    result[:down].sort!
    result
  end
  private
  def is_up(ip)
    checker = Net::Ping::External.new(ip, nil, @timeout)
    checker.ping
  rescue StandardError => e
    "#{ip} ERROR - #{e.message}"
    false
  end
end
```

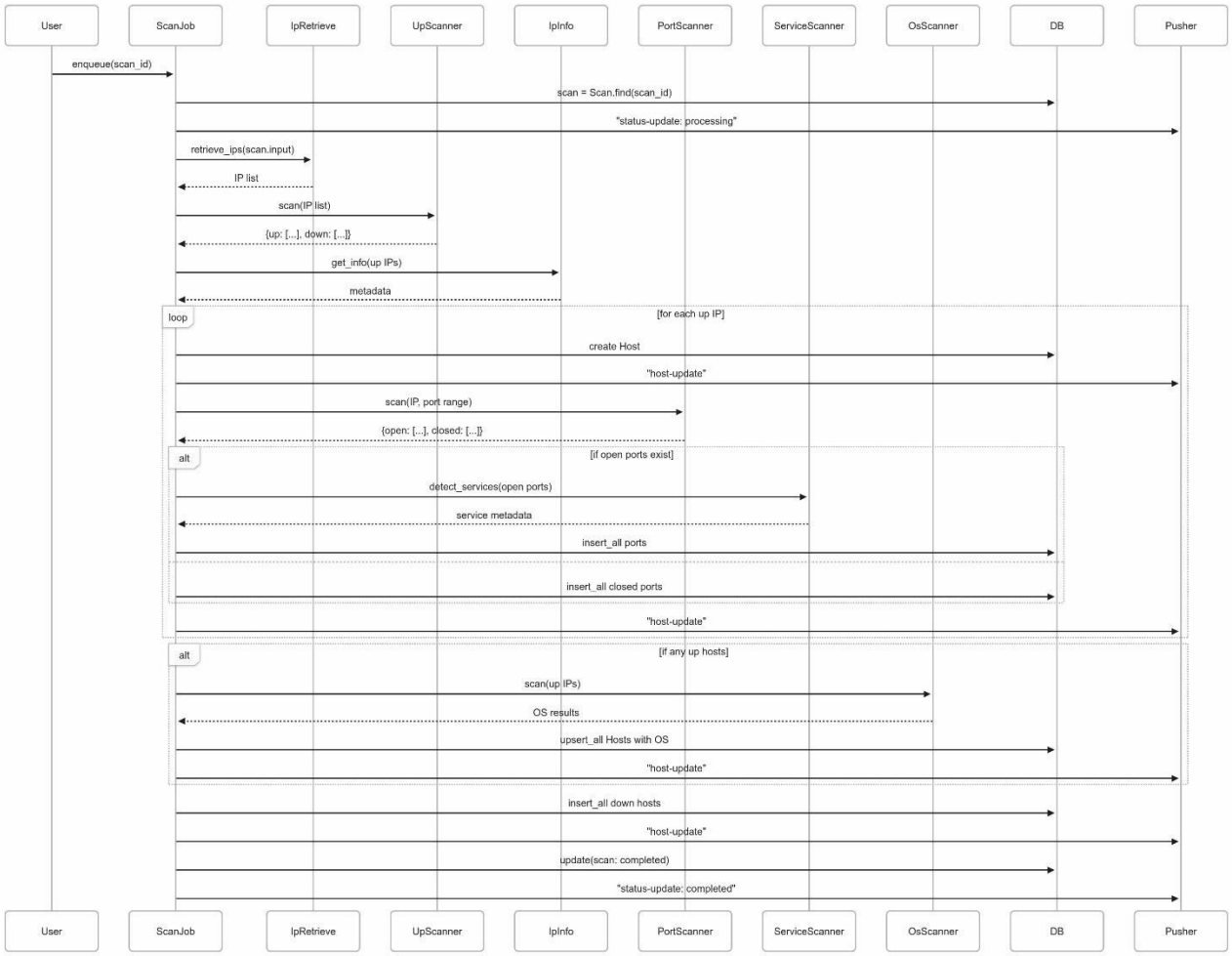
ДОДАТОК Б

Копія графічної частини





					КРБКБ.200160.22.01.03 Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Вебсистема сканування хостів і портів для аналізу вразливостей мережі	Літ / Маса / Масштаб
Розроб.		Зейлик Р.Ю.				Н
Перевір.		Муляр І.В.			Схема бази даних	Аркуш / Аркушів 1
Т.контр.						
Н.контр.		Мостовий С.В.				ХНУ, КБс-22-1
Затверд.		Кльоц Ю.П.				



								КРБКБ.200160.22.01.03 Е8	
								Вебсистема сканування хостів і портів для аналізу вразливостей мережі Діаграма послідовності сервісу сканування	Літ: н; Маса; Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата					Аркуш; Аркушів: 1
Розроб.		Зейлик Р.Ю.							
Перевір.		Муляр І.В.							ХНУ, КБс-22-1
Т.контр.									
Н.контр.		Мостовий С.В.							
Затверд.		Кльоц Ю.П.							

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Зейлика Романа Юрійовича
ПІБ здобувача вищої освіти

Студента ФІТ, 3 курсу, групи КБс-22-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

28.05.2025
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 10%**

	Document		Sum coincidence on the DB	
ID: 242397 Title: Вебсистема сканування хостів і портів для аналізу вразливостей мережі Added in a DB: 2025-05-29 Authors: Зейлик Роман Юрійович Heads: Муляр І.В. Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	92344	681	934 (1%)	12 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Зейлик Роман Юрійович

Співавтор:

Назва: Вебсистема сканування хостів і портів для аналізу вразливостей мережі

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:1.9%

Коефіцієнт подібності 2:0.3%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-29 13:07:50.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

01.06.2025р.

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Вебсистема сканування хостів і портів для аналізу вразливостей мережі.

Автор: Зейлик Роман Юрійович

Спеціальність: 125 – Кібербезпека та захист інформації

Освітня програма:Кібербезпека та захист інформації

Науковий керівник: Ігор Муляр канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 98.1%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99.0%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Виявлені модифікації стосуються математичних формул і не є порушенням академічної доброчесності.

Керівник роботи _____ Ігор Муляр

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Зейлик Роман Юрійович

Тема Вебсистема сканування хостів і портів для аналізу вразливостей мережі

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3 ; **кількість сторінок записки** 67 .

1. Короткий зміст роботи та прийнятих рішень Кваліфікаційну роботу присвячено розробці вебзастосунку для сканування хостів і портів з метою аналізу вразливостей комп'ютерних мереж. Система дозволяє користувачу ініціювати ручне сканування заданих хостів, переглядати результати у зручному інтерфейсі, а також взаємодіє з базою даних через фонові воркери на базі Sidekiq, що забезпечує масштабованість і стабільність роботи. В роботі особливу увагу приділено модульній побудові системи, що спрощує її подальший розвиток.

2. **Висновок про відповідність кваліфікаційної роботи завданню** У кваліфікаційній роботі повністю виконано поставлене завдання як у теоретичній, так і в практичній частині

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У роботі використано сучасні підходи до проєктування вебзастосунків, включаючи використання фреймворку Ruby on Rails, фонової обробки задач за допомогою Sidekiq, та побудову структурованої архітектури на основі принципів MVC. Перший розділ містить огляд предметної області та аналіз сучасних інструментів для сканування. У другому розділі детально описано архітектуру системи, обрано інструменти для реалізації системи. Третій розділ присвячено розробці модулів сканування, серверної та клієнтської частини, а також тестуванню працездатності системи

4. **Позитивні сторони роботи** Робота структурована чітко й логічно, охоплює всі етапи розробки – від аналізу предметної області до тестування. Використання Ruby on Rails разом із Sidekiq для фонового виконання задач забезпечує стабільну й ефективну роботу системи. Зручний інтерфейс і модульна архітектура сприяють простоті користування та можливості подальшого розширення функціоналу.

5. Негативні сторони роботи До незначних недоліків можна віднести відсутність підтримки експорту результатів для подальшої обробки. Також система поки що не передбачає механізмів для повного налаштування процесу сканування.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. В цілому, графічне оформлення є якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота присвячена актуальній темі мережевої безпеки. Розроблена система демонструє знання сучасних технологій веброзробки та основ інформаційної безпеки. Робота охоплює всі ключові етапи розробки: від аналізу предметної області до тестування. Незважаючи на певні недоліки, робота є цілісною, технічно грамотною та заслуговує на позитивну оцінку.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Пивовар Олег Сергійович
доцент кафедри ТМІТ, кандидат технічних наук

« 6 » ос 2025р (підпис)