

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

_____ Наталія АУШЕВА

« ____ » _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавра**

За освітньою програмою «Цифрові технології в енергетиці»

Спеціальності 122 «Комп'ютерні науки»

на тему: «Веб орієнтована інформаційна система обліку комп'ютерного обладнання»

Виконав: студент 4 курсу, групи ТР-22

Носенко Артем Олександрович

(прізвище, ім'я, по батькові)

_____ (підпис)

Керівник: старший викладач каф. цифрових технологій в енергетиці,

Ольга БЕСПАЛА

(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові)

_____ (підпис)

Рецензент: доцент каф. теплової та альтернативної енергетики,

к.т.н, доц. Олександр СІРИЙ

(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові)

_____ (підпис)

Н.контроль доцент каф. цифрових технологій в енергетиці,

д.ф Артем ГУРІН

(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові)

_____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

(підпис)

Київ — 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра

ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)

Спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

_____ Наталія АУШЕВА

“ ” _____ 2026 р.

З А В Д А Н Н Я
на дипломну роботу студенту

Носенку Артему Олександровичу

(прізвище, ім’я, по батькові)

1. Тема роботи «Веб орієнтована інформаційна система обліку комп’ютерного обладнання»

керівник роботи Беспала Ольга Миколаївна, старший викладач

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 р. № 1992 -с

2. Термін подання роботи студентом 08.06.2026 р.

3. Вихідні дані до роботи: мова програмування Python, фреймворк Flask, бібліотека React з TypeScript, система управління базами даних PostgreSQL, ORM-бібліотека SQLAlchemy, інструмент збірки Vite, HTTP-клієнт Axios, утиліта мережевого сканування Nmap, середовище розробки VS Code.

4. Перелік питань, які потрібно розробити: 1) визначити мету й задачі роботи; 2) проаналізувати існуючі системи обліку ІТ-активів; 3) визначити основні вимоги до веб-орієнтованої системи обліку комп’ютерного обладнання; 4) спроектувати архітектуру програмного забезпечення та структуру бази даних; 5) розробити серверну частину системи з REST API та модулем авторизації; 6) реалізувати програмний агент для автоматичного збору даних з робочих станцій; 7) впровадити модуль перевірки вразливостей програмного забезпечення через базу NVD; 8) розробити клієнтський інтерфейс для управління обладнанням та перегляду результатів сканування.

5. Орієнтовний перелік ілюстративного матеріалу: діаграма USE-CASE, схема архітектури системи, діаграма бази даних, блок-схема алгоритму реєстрації пристрою агентом, блок-схема алгоритму перевірки вразливостей, скріншоти інтерфейсу користувача.

6. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Затвердження теми роботи	27.03.2026	Виконано
2	Дослідження існуючих рішень для визначення вимог до системи	20 - 22.04.26	Виконано
3	Проектування архітектури системи та структури бази даних	23–24.04.2026	Виконано
4	Вибір технологій та засобів розробки	25.04.26	Виконано
5	Розробка серверної частини та програмного агента	27.04–03.05.2026	Виконано
6	Розробка клієнтського інтерфейсу	04–10.05.2026	
7	Тестування та налагодження системи	11–12.05.2026	Виконано
8	Оформлення пояснювальної записки	05-12.05.26	Виконано
9	Захист програмного продукту	13.05.26	Виконано
10	Передзахист	03.06.2026	Виконано
11	Захист	16.06.26	Виконано

Студент

Керівник роботи

(підпис)

(підпис)

Артем НОСЕНКО

(ім'я, ПРІЗВИЩЕ)

Ольга БЕСПАЛА

(ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 53 сторінках, містить 16 ілюстрацій, 1 таблицю, 1 додаток, 20 джерел в переліку посилань.

Метою роботи є проєктування та розробка веб-орієнтованої системи обліку комп'ютерного обладнання, що забезпечує автоматизований збір даних про ІТ-активи організації, моніторинг їх стану та виявлення вразливостей встановленого програмного забезпечення.

Основний зміст роботи включає аналіз існуючих систем обліку ІТ-активів та їх порівняльну характеристику, вибір технологічного стеку розробки, проєктування трирівневої архітектури системи та структури реляційної бази даних, розробку серверної частини з REST API та модулем авторизації, реалізацію програмного агента для автоматичного збору даних з робочих станцій, впровадження модуля мережевого сканування та перевірки вразливостей через базу NVD, а також розробку клієнтського інтерфейсу.

Методи та засоби: мова програмування Python, фреймворк Flask, бібліотека React з TypeScript, система управління базами даних PostgreSQL, ORM-бібліотека SQLAlchemy, інструмент збірки Vite, HTTP-клієнт Axios, утиліта мережевого сканування Nmap.

Результат: програмний застосунок для автоматизованого обліку комп'ютерного обладнання з агентним збором даних, мережевим скануванням, перевіркою вразливостей та QR-інвентаризацією.

Розроблений додаток може бути впроваджений в організаціях різного масштабу для підвищення ефективності управління ІТ-інфраструктурою та зниження ризиків інформаційної безпеки.

Ключові слова: ВЕБ-ЗАСТОСУНОК, ОБЛІК ОБЛАДНАННЯ, IT ASSET MANAGEMENT, ПРОГРАМНИЙ АГЕНТ, МЕРЕЖЕВЕ СКАНУВАННЯ, ВРАЗЛИВОСТІ, NVD, QR-КОД, FLASK, REACT, POSTGRESQL.

ABSTRACT

The diploma project consists of 53 pages and includes 16 illustrations, 1 table, 1 appendix, and 20 references.

The aim of the study is to design and develop a web-oriented computer equipment accounting system that provides automated collection of IT asset data, monitoring of their status, and detection of software vulnerabilities.

The main content of the thesis includes an analysis of existing IT asset management systems and their comparative characteristics, selection of the development technology stack, design of a three-tier system architecture and relational database structure, development of a server-side REST API with an authorization module, implementation of a software agent for automatic data collection from workstations, deployment of a network scanning module and vulnerability checking via the NVD database, as well as development of the client interface.

Methods and tools: Python programming language, Flask framework, React library with TypeScript, PostgreSQL database management system, SQLAlchemy ORM library, Vite build tool, Axios HTTP client, Nmap network scanning utility.

Result: a software application for automated computer equipment accounting with agent-based data collection, network scanning, vulnerability checking, and QR-based inventory management.

The developed application can be implemented in organizations of various sizes to improve IT infrastructure management efficiency and reduce information security risks.

Keywords: WEB APPLICATION, EQUIPMENT ACCOUNTING, IT ASSET MANAGEMENT, SOFTWARE AGENT, NETWORK SCANNING, VULNERABILITIES, NVD, QR CODE, FLASK, REACT, POSTGRESQL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	7
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Постановка задачі	10
1.2 Аналіз наявних рішень у сфері обліку комп'ютерного обладнання	12
1.2.1 Аналіз системи Snipe-IT	12
1.2.2 Аналіз системи GLPI	13
1.2.3 Аналіз системи Lansweeper.....	14
1.2.4 Результат аналізу сучасних систем обліку комп'ютерного обладнання	14
2 ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ СИСТЕМИ	17
2.1 Огляд засобів розробки	17
2.2 Інструменти та засоби розробки.....	18
2.3 Технології розробки серверної частини	19
2.4 Технології розробки клієнтського інтерфейсу.....	20
2.5 Використання PostgreSQL в розробці.....	21
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ.....	23
3.1 Сценарії використання системи	23
3.2 Загальна архітектура системи.....	24
3.3 Структура бази даних	26
3.3.1 Модель пристрою.....	27
3.3.2 Модель вразливостей.....	28
3.3.3 Модель журналу змін пристрою.....	29
3.3.4 Модель користувача	29
3.3.5 Модель адміністратор.....	30
3.3.6 Модель історії мережевого сканування	30
3.4 Реалізація бізнес-логіки.....	31
3.4.1 Логіка роботи з пристроями.....	31

3.4.2	Логіка агента.....	32
3.4.3	Перевірка вразливостей.....	34
3.4.4	Мережеве сканування та перевірка статусу онлайн.....	36
3.5	Реалізація клієнтського інтерфейсу	37
3.5.1	Авторизація та управління станом.....	37
3.5.2	Реалізація сторінок інтерфейсу	38
4	РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ	40
4.1	Встановлення та налаштування системи.....	40
4.2	Перша реєстрація пристрою агентом.....	42
4.3	Вхід до системи.....	43
4.4	Головна сторінка та список обладнання.....	44
4.5	Картка пристрою	44
4.6	Додавання пристрою вручну	47
4.8	Перегляд вразливостей.....	49
4.9	Мережеве сканування.....	51
4.10	Публічна картка пристрою	52
	ВИСНОВКИ.....	53
	СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	54
	ДОДАТОК А.....	56

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API — Application Programming Interface
CIDR — Classless Inter-Domain Routing
CORS — Cross-Origin Resource Sharing
CPE — Common Platform Enumeration
CRUD — Create, Read, Update, Delete
CSS — Cascading Style Sheets
CVE — Common Vulnerabilities and Exposures
CVSS — Common Vulnerability Scoring System
HTML — HyperText Markup Language
HTTP — HyperText Transfer Protocol
HTTPS — HyperText Transfer Protocol Secure
IP — Internet Protocol
ITAM — IT Asset Management
ITIL — Information Technology Infrastructure Library
JSON — JavaScript Object Notation.
JWT — JSON Web Token
MAC — Media Access Control
NVD — National Vulnerability Database
ORM — Object-Relational Mapping
PBKDF2 — Password-Based Key Derivation Function 2
PNG — Portable Network Graphics
QR — Quick Response
REST — Representational State Transfer
SHA-256 — Secure Hash Algorithm 256
SPA — Single Page Application
SQL — Structured Query Language

URL — Uniform Resource Locator

VBS — Visual Basic Script

IT — інформаційні технології.

СУБД — система управління базами даних.

ВСТУП

Стрімка цифровізація підприємств та організацій висуває нові вимоги до управління корпоративною інфраструктурою, яка щодня поповнюється новими одиницями комп'ютерного обладнання. Особливо це актуально для ІТ-відділів, де точність обліку техніки та своєчасне виявлення загроз інформаційної безпеки безпосередньо впливають на операційну ефективність організації.

В умовах постійного зростання кількості пристроїв та потреби у централізованому контролі над ними виникає необхідність в автоматизованій системі, яка дозволяє усунути ручний облік, отримати актуальні дані про стан інфраструктури та оперативно реагувати на виявлені вразливості. На відміну від типових корпоративних рішень, таких як Lansweeper, цей підхід дозволяє створити відкриту та адаптовану до реальних потреб систему без значних витрат на ліцензування.

У рамках дипломної роботи було спроектовано та реалізовано повнофункціональну веб-орієнтовану систему обліку комп'ютерного обладнання, яка автоматично збирає технічні характеристики робочих станцій через програмного агента, виявляє незареєстровані пристрої в мережі та перевіряє встановлене програмне забезпечення на наявність відомих вразливостей через базу NVD. Особливу увагу було приділено зручності роботи адміністратора — від автоматичної реєстрації пристроїв до QR-інвентаризації та наочного відображення критичних загроз безпеці. Уся система побудована з використанням сучасних технологій: Python та Flask для серверної логіки, PostgreSQL як надійна реляційна база даних, React з TypeScript для реалізації клієнтського інтерфейсу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У сучасних умовах цифровізації підприємств особливої актуальності набувають системи, що дозволяють ефективно управляти ІТ-інфраструктурою організації. Облік комп'ютерного обладнання є важливою складовою операційної діяльності будь-якого підприємства, від якої вимагається точність, оперативність та автоматизація рутинних процесів. Одним із перспективних підходів є побудова веб-орієнтованої системи обліку на основі агентного збору даних, що поєднує автоматичну інвентаризацію, моніторинг стану пристроїв та виявлення вразливостей програмного забезпечення. Це комплексна задача, що передбачає не лише технічну реалізацію серверної та клієнтської частин, а й розробку програмного агента для збору даних безпосередньо з робочих станцій. Важливою перевагою обраного підходу є можливість впровадження системи в організаціях різного масштабу без значних витрат на розгортання. Враховуючи обмежену кількість доступних рішень, що поєднують облік обладнання з автоматичною перевіркою вразливостей, розроблена система має практичну цінність для широкого кола підприємств, зацікавлених у підвищенні ефективності управління ІТ-інфраструктурою.

1.1 Постановка задачі

Сучасні організації потребують інструментів, що дозволяють у будь-який момент отримати актуальну інформацію про стан комп'ютерного обладнання без витрат часу на ручне оновлення даних. Розроблювана система має забезпечити автоматизований збір технічних характеристик пристроїв, виявлення незареєстрованої техніки в корпоративній мережі та перевірку встановленого програмного забезпечення на наявність відомих вразливостей. Додатково система має підтримувати механізм QR-інвентаризації, журналювання всіх змін у картках обладнання та фоновий моніторинг доступності пристроїв.

З урахуванням зазначених вимог, у рамках дипломної роботи поставлено такі задачі:

- проведення аналізу наявних рішень у сфері управління ІТ-активами, визначення їх переваг та недоліків і формулювання вимог до розроблюваної системи;
- проєктування архітектури системи та структури реляційної бази даних, що охоплює сутності пристроїв, співробітників, адміністраторів, вразливостей, журналу змін та історії сканувань;
- розробка серверної частини з реалізацією REST API, системи авторизації на основі JWT-токенів та фонового процесу моніторингу доступності пристроїв;
- реалізація програмного агента для автоматичного збору апаратних характеристик робочих станцій, визначення версій встановленого програмного забезпечення та перевірки їх на наявність вразливостей через Національну базу даних вразливостей NVD з визначенням рівня критичності за стандартом CVSS;
- розробка модуля мережевого сканування для виявлення пристроїв у корпоративній мережі, що не внесені до реєстру обладнання;
- розробка клієнтського інтерфейсу, що включає управління обліком обладнання, перегляд вразливостей з фільтрацією за рівнем критичності, відображення результатів мережевого сканування та QR-інвентаризацію;
- проведення комплексного тестування системи та перевірка коректності роботи всіх реалізованих функцій.

Реалізація перелічених задач дозволить створити повнофункціональний веб-застосунок з трирівневою архітектурою, що поєднує серверну частину, клієнтський інтерфейс та програмного агента для автономного збору даних. Проєктована система розробляється з урахуванням простоти розгортання для можливості подальшого впровадження в організаціях різного масштабу з метою підвищення ефективності управління ІТ-інфраструктурою.

1.2 Аналіз наявних рішень у сфері обліку комп'ютерного обладнання

У даному підрозділі розглянуто існуючі програмні рішення для управління комп'ютерним обладнанням організацій. Для кожного з них визначено ключові функціональні можливості, переваги та обмеження з точки зору поставлених вимог до розроблюваної системи. На основі проведеного порівняльного аналізу обґрунтовано доцільність розробки власного рішення.

1.2.1 Аналіз системи Snipe-IT

Snipe-IT – безкоштовна система управління IT-активами з відкритим вихідним кодом, реалізована на основі фреймворку Laravel. Система призначена для ефективного відстеження апаратного забезпечення, програмних ліцензій та периферійних пристроїв, забезпечуючи точний облік замість традиційних електронних таблиць. Серед ключових можливостей — відстеження призначень активів, управління ліцензіями, підтримка QR-кодів та штрих-кодів для швидкої ідентифікації, цифрові підписи для забезпечення відповідності вимогам та email-сповіщення про закінчення гарантій.

Система є легкою та налаштовуваною, а її відкритий API дозволяє автоматизувати процеси та виконувати інтеграції. Журнали аудиту є зручними та інформативними.

Проте система має суттєві обмеження. Snipe-IT повністю залежить від ручного введення даних, що збільшує ймовірність помилок через людський фактор. Система не має вбудованого мережевого сканування — всі пристрої необхідно вносити вручну або через імпорт. Можливості звітності та аналітики обмежені порівняно з конкурентами. Перевірка вразливостей встановленого програмного забезпечення у стандартному функціоналі повністю відсутня.

Таким чином, Snipe-IT є зручним та доступним інструментом для базового обліку активів, однак відсутність автоматичного збору даних та перевірки вразливостей суттєво знижує її практичну цінність для задач інформаційної безпеки[1].

1.2.2 Аналіз системи GLPI

GLPI – комплексна платформа управління IT-активами та IT-послугами з відкритим вихідним кодом. Система поєднує відстеження активів, операції служби підтримки та управління послугами відповідно до методології ITIL, що дозволяє IT-відділам оркеструвати повні робочі процеси від вирішення інцидентів до управління змінами.

GLPI забезпечує централізоване відстеження апаратних активів, програмних ліцензій та пов'язаних послуг, дозволяючи адміністраторам вести детальні записи про пристрої – комп'ютери, сервери та мережеве обладнання. Система підтримує різні категорії та статуси заявок, забезпечуючи повну підзвітність і чітку комунікацію між підрозділами.

Однак GLPI має ряд недоліків. Складне початкове налаштування вимагає технічної експертизи та значного часу на конфігурацію. Інтерфейс може здаватися застарілим порівняно з сучасними рішеннями, що негативно впливає на адаптацію користувачів. Дані про активи доводиться вносити вручну. Автоматичний збір даних з робочих станцій можливий лише через окремих плагін FusionInventory, що потребує додаткового встановлення та налаштування. Перевірка вразливостей програмного забезпечення також відсутня.

Отже, GLPI є потужним інструментом для великих організацій з розгалуженою IT-службою, однак надмірна складність і відсутність автоматизованого збору даних роблять її недоцільною для підприємств, що потребують простого і ефективного рішення[2].

1.2.3 Аналіз системи Lansweeper

Lansweeper – комерційне рішення для автоматичної інвентаризації та управління IT-активами. Система забезпечує повну видимість IT, OT та IoT активів, включаючи апаратне та програмне забезпечення в усьому технологічному середовищі, автоматично виявляє та інвентаризує кожен актив і збирає дані про використання, життєвий цикл та ризики.

Найціннішою функцією є автоматичне виявлення активів — просте у встановленні та використанні. Система дозволяє проводити звіти та бачити всі наявні активи, що є критично важливим у сфері кібербезпеки. Рішення підтримує агентів, які можуть надсилати дані у хмару навіть якщо пристрій знаходиться поза мережею.

Проте система має суттєві обмеження. Lansweeper пропонує рівневе ціноутворення на основі кількості активів: план Starter починається від 2400 доларів на рік для до 2000 активів, план Pro коштує близько 4308 доларів на рік. Ціна значно зросла за останні роки, а такі функції як сканування вразливостей додатково захищені платним рівнем вищого рівня. Крім того, система є закритою і не дозволяє адаптувати функціонал під специфічні потреби конкретної організації.

Таким чином, Lansweeper забезпечує високий рівень автоматизації виявлення пристроїв, проте висока вартість ліцензування та додаткова оплата за функції безпеки роблять її недоступною для більшості малих і середніх організацій[3].

1.2.4 Результат аналізу сучасних систем обліку комп'ютерного обладнання

Дослідивши ринок, було виявлено що на даний момент існує декілька рішень для організацій, які забезпечують функціонал обліку комп'ютерного обладнання та управління IT-активами. Кожна з розглянутих систем має свої особливості, переваги та недоліки, які наведено у таблиці 1.1.

Таблиця 1.1 — Порівняльна характеристика сучасних систем обліку ІТ-активів

Характеристика	Snipe-IT	GLPI	Lansweeper
Відкритий вихідний код	Так	Так	Ні
Автоматичний збір даних	Ні	Частково	Так
Перевірка вразливостей	Ні	Ні	Платно
Мережеве сканування	Ні	Частково	Так
Генерація QR-кодів	Так	Ні	Ні
Простота розгортання	Середня	Низька	Середня
Безкоштовне використання	Так	Так	Ні
Журнал змін	Так	Так	Так
Агент для збору даних	Ні	Ні	Так

У порівнянні з розглянутими системами можна зробити такі висновки щодо проектування розроблюваного рішення.

З позитивного досвіду існуючих рішень варто перейняти такі підходи. Snipe-IT демонструє що підтримка QR-кодів та зручний журнал аудиту є корисними функціями для щоденної роботи адміністратора, тому обидва елементи реалізовано і в розроблюваній системі. GLPI підтверджує що журналювання всіх змін є необхідним елементом корпоративної системи, тому механізм фіксації кожної модифікації у картці пристрою з зазначенням автора і часу реалізовано як обов'язкову функцію. Lansweeper доводить, що автоматичне виявлення пристроїв у мережі та агентний збір даних є значною перевагою, тому розроблювана система включає обидва ці механізми.

З недоліків існуючих рішень були зроблені такі висновки. Повна відсутність автоматичного збору даних у Snipe-IT та GLPI призводить до значних трудових витрат і помилок, тому в розроблюваній системі реалізовано програмний агент, який самостійно збирає характеристики робочих станцій без участі адміністратора. Надмірна складність GLPI ускладнює впровадження, тому розроблювана система

орієнтована виключно на задачі обліку без зайвого функціоналу. Відсутність безкоштовної перевірки вразливостей у всіх розглянутих системах залишає організації незахищеними, тому інтеграція з базою вразливостей NVD є ключовою перевагою розроблюваного рішення. Висока вартість Lansweeper робить його недоступним для малих організацій, тому розроблювана система є повністю безкоштовною.

Таким чином, розроблювана система усуває основні недоліки існуючих аналогів і поєднує їх кращі практики, забезпечуючи автоматизований збір даних через агента, безкоштовну перевірку вразливостей через базу вразливостей NVD, мережеве сканування та просте розгортання в рамках єдиного відкритого рішення.

2 ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ СИСТЕМИ

Розроблювана система не прив'язана до жодної конкретної платформи, що надає повну свободу у виборі технологій розробки. Разом з тим кожен компонент системи — серверна частина, клієнтський інтерфейс, агент та база даних — вимагав ретельного підбору інструментів з урахуванням функціональних вимог, зрілості екосистеми та можливості безкоштовного використання. Обрані технології є сучасними, добре задокументованими та активно підтримуються спільнотою розробників, що забезпечує стабільність системи та можливість її подальшого масштабування.

2.1 Огляд засобів розробки

При розробці системи обліку комп'ютерного обладнання технології розробки підбирались з урахуванням функціональних вимог до кожного компонента: серверної частини, агента, бази даних та клієнтського інтерфейсу. Основою серверної частини є мікрофреймворк Flask на мові Python. На відміну від повноцінних фреймворків, Flask не накладає жорстких обмежень на структуру проєкту, що дозволило організувати API у вигляді незалежних модулів, окремо для пристроїв, авторизації, сканування та публічного доступу.

Програмний агент також реалізовано на Python і призначений для розгортання безпосередньо на робочих станціях організації. Він функціонує у фоновому режимі та виконує два основних завдання: збір апаратних характеристик машини через бібліотеки psutil і py-cpuinfo, а також перевірку встановленого програмного забезпечення на наявність відомих вразливостей через публічний API бази NVD. Для виявлення пристроїв у корпоративній мережі задіяно утиліту nmap, яка виконує сканування заданого діапазону IP-адрес без встановлення агента на кожен пристрій.

Дані системи зберігаються у реляційній базі PostgreSQL, обраній через нативну підтримку типу JSON та коректну роботу при паралельних запитах від кількох підключень одночасно. Для взаємодії з базою застосовано бібліотеку SQLAlchemy у режимі ORM, кожна таблиця описана як окремий Python-клас, що виключає написання SQL-запитів вручну та унеможливорює SQL-ін'єкції завдяки параметризованим запитам.

Клієнтський інтерфейс розроблено на React з TypeScript. Статична типізація TypeScript дозволяє виявляти невідповідності типів між компонентами ще під час написання коду, що суттєво скорочує кількість помилок при масштабуванні проєкту. HTTP-комунікація з API реалізована через Axios з автоматичним додаванням токена авторизації до кожного запиту, маршрутизація між сторінками через React Router, а збірка через Vite. Стилзація виконана засобами Bootstrap 5 з підтримкою адаптивного відображення.

Поєднання перелічених інструментів забезпечило реалізацію всіх заявлених функцій системи в межах єдиного стеку без залучення сторонніх платних сервісів.

2.2 Інструменти та засоби розробки

Для розробки системи обліку комп'ютерного обладнання використовувалось середовище Visual Studio Code – один з найпоширеніших редакторів коду з відкритим вихідним кодом. Редактор відзначається широкою екосистемою розширень, вбудованою підтримкою Git, інтегрованим терміналом та інтелектуальним автодоповненням коду через IntelliSense. Для роботи з Python та TypeScript встановлено відповідні розширення, які забезпечують підсвічування синтаксису, статичний аналіз коду та автоматичне форматування.

Контроль версій проєкту здійснювався засобами системи Git з розміщенням репозиторію на платформі GitHub. Використання гілок дозволило розробляти окремі функціональні модулі ізольовано та інтегрувати їх до основної гілки після

перевірки. Це забезпечило збереження історії змін та можливість повернення до попередніх версій у разі виникнення помилок.

Тестування серверного API виконувалось за допомогою інструменту Postman. Він дозволяє формувати HTTP-запити довільного типу, передавати заголовки авторизації та тіло запиту, а також переглядати відповіді сервера у зручному форматі. Завдяки цьому вдалося перевірити коректність роботи кожного маршруту API до початку розробки клієнтського інтерфейсу.

Для управління залежностями серверної частини використовувався менеджер пакетів `pip` з фіксацією версій бібліотек у файлі `requirements.txt`, а для клієнтської частини `npm` з файлом `package.json`. Такий підхід гарантує відтворюваність середовища розробки на будь-якій машині та спрощує розгортання системи.

2.3 Технології розробки серверної частини

Мова програмування Python є однією з найпоширеніших мов для розробки серверних застосунків, що характеризується зрозумілим синтаксисом та розвиненою екосистемою бібліотек[4].

Серверний API побудовано на основі мікрофреймворку Flask. Кожен маршрут відповідає конкретній операції системи – отриманню списку пристроїв, оновленню характеристик, запуску сканування тощо[5]. Для авторизації підключено бібліотеку `PyJWT`, яка забезпечує генерацію та перевірку JWT-токенів[6]. Хешування паролів реалізовано засобами бібліотеки `werkzeug` із застосуванням алгоритму `PBKDF2-SHA256`[7]. Міжсайтові запити між клієнтським інтерфейсом і сервером обробляються за допомогою Flask-CORS.

Взаємодія з базою даних реалізована через бібліотеку `SQLAlchemy` у режимі ORM[8]. Кожна таблиця представлена окремим класом-моделлю, операції створення, читання, оновлення та видалення виконуються через об'єкти Python без написання SQL-запитів вручну. Такий підхід виключає можливість SQL-ін'єкцій,

завдяки параметризованим запитам та спрощує підтримку коду при зміні структури бази даних. Підключення до PostgreSQL здійснюється через драйвер `psycopg` з налаштованим пулом з'єднань, що забезпечує стабільну роботу сервера в умовах паралельного виконання запитів.

Програмний агент реалізовано окремим Python-скриптом, що встановлюється на робочі пристрої організації та виконується у фоновому режимі. Для збору апаратних характеристик комп'ютера використовуються бібліотеки `psutil` та `py-cpuinfo`[9]. Бібліотека `psutil` повертає дані про обсяг оперативної пам'яті та кількість фізичних ядер процесора, `py-cpuinfo` – повну назву моделі процесора. Серійний номер материнської плати зчитується через PowerShell на Windows або `dmidecode` на Linux і слугує унікальним ідентифікатором пристрою в системі. Для перевірки вразливостей встановленого програмного забезпечення, агент формує CPE-рядки відповідно до стандарту NIST та надсилає запити до публічного API бази даних NVD за допомогою бібліотеки `requests`[10].

Мережеве сканування виконується через виклик утиліти `ntar` засобами стандартної бібліотеки `subprocess`[11].

Генерація QR-кодів виконується бібліотекою `qrcode` з підтримкою збереження зображень у форматі PNG[12].

2.4 Технології розробки клієнтського інтерфейсу

Клієнтський інтерфейс системи розроблено з використанням мови TypeScript та бібліотеки React версії 19[13]. TypeScript є розширенням JavaScript з підтримкою статичної типізації, що дозволяє виявляти помилки на етапі написання коду та забезпечує надійну документацію типів даних, якими обмінюються компоненти інтерфейсу. Усі структури даних системи – пристрої, співробітники, вразливості, результати сканування – описано через інтерфейси TypeScript[14], що унеможливорює передачу некоректних даних між частинами застосунку.

React забезпечує компонентний підхід до побудови інтерфейсу. Кожна сторінка системи реалізована як окремий компонент зі своїм станом та логікою отримання даних. Стан авторизації керується через React Context і доступний будь-якому компоненту без передачі через props. Маршрутизація між сторінками реалізована засобами бібліотеки React Router 7[15], HTTP-запити до серверного API виконуються через Axios[16] з автоматичним додаванням токена авторизації до кожного запиту. Для збірки проєкту використовується Vite[17], стилізація інтерфейсу виконана з використанням Bootstrap 5 та React-Bootstrap.

2.5 Використання PostgreSQL в розробці

Для зберігання даних системи обрано PostgreSQL – реляційну СУБД з відкритим вихідним кодом, яка поєднує високу надійність, підтримку складних типів даних та стабільну роботу в умовах паралельних підключень. Саме ці характеристики визначили її вибір для даного проєкту[18]. PostgreSQL задіяно у таких аспектах роботи системи:

- збереження даних про пристрої – основна таблиця містить інвентарний номер, назву, серійний номер, тип, статус, ідентифікатор відповідального, технічні характеристики у форматі JSON, шлях до QR-коду та час останньої активності;
- підтримка типу JSON – характеристики різних категорій пристроїв зберігаються в одному полі без зміни схеми бази при додаванні нових параметрів, що забезпечує гнучкість структури даних;
- збереження журналу змін – кожна зміна у картці пристрою фіксується в окремій таблиці з прив'язкою до пристрою, полем що змінилось, старим і новим значеннями та ім'ям адміністратора;
- збереження вразливостей – виявлені агентом CVE зберігаються з прив'язкою до конкретного пристрою, кодом вразливості, рівнем критичності та описом;

- цілісність даних — зовнішні ключі між таблицями та каскадне видалення гарантують, що при видаленні пристрою автоматично видаляються всі пов'язані вразливості та записи журналу без додаткових операцій;

- робота в багатопотоковому середовищі — підключення через драйвер psycopg з налаштованим пулом з'єднань забезпечує коректну паралельну обробку запитів від веб-інтерфейсу, агентів та фонові перевірки зв'язку з пристроями;

Використання PostgreSQL дозволило побудувати надійну основу для зберігання всіх даних системи, незалежно від кількості підключених пристроїв та інтенсивності звернень до бази.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмний продукт реалізовано з дотриманням принципів модульної архітектури та чіткого розподілу відповідальностей між компонентами системи. Розроблено як структуру бази даних, так і серверну бізнес-логіку та клієнтський інтерфейс користувача. Кожен компонент використовує власний набір технологій, проте всі вони взаємодіють між собою через визначені інтерфейси — серверне API слугує єдиною точкою обміну даними між базою даних, агентом та браузером. Такий підхід забезпечує незалежність окремих частин системи та спрощує їх подальше розширення і супровід.

3.1 Сценарії використання системи

Для визначення ключових функціональних вимог та наочного представлення взаємодії учасників із системою обліку комп'ютерного обладнання було побудовано діаграму прецедентів (рисунок 3.1).

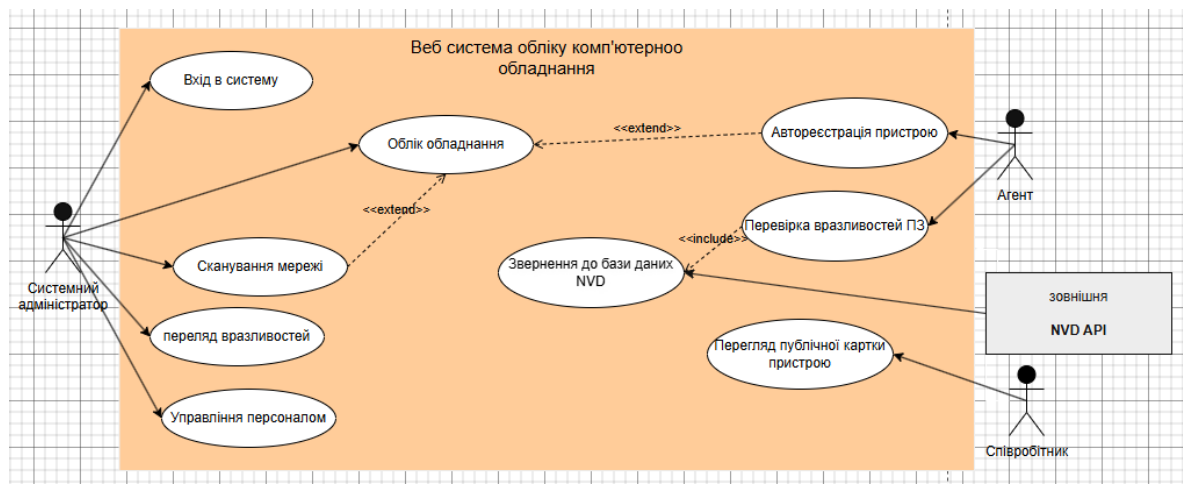


Рисунок 3.1 — Діаграма прецедентів

Діаграма відображає основних акторів системи та їхні зв'язки з функціональними блоками. Акторами виступають системний адміністратор, агент, співробітник організації та зовнішня система NVD API.

Адміністратор є основним користувачем системи. Він здійснює вхід до веб-інтерфейсу, переглядає та редагує картки обладнання, призначає відповідальних співробітників, змінює статуси пристроїв, запускає мережеве сканування та аналізує виявлені вразливості. Уся взаємодія адміністратора із системою фіксується в журналі змін, що забезпечує прозорість обліку у відділі.

Агент функціонує у фоновому режимі на робочих станціях організації та виконує два основних сценарії. Перший – автоматична реєстрація пристрою: під час першого запуску агент збирає технічні характеристики пристрою та передає їх на сервер, після чого система створює картку пристрою і генерує для нього QR-код. Під час повторного запуску оновлюються характеристики та час останньої активності. Другий сценарій – перевірка вразливостей: агент збирає перелік встановленого програмного забезпечення, визначає його версії та звертається до бази NVD для отримання актуальних даних про відомі вразливості, після чого результати зберігаються в системі.

Співробітник організації взаємодіє із системою опосередковано – через сканування QR-коду, розміщеного на наклейці пристрою. Це відкриває публічну сторінку з основними відомостями про обладнання без необхідності авторизації, що значно спрощує процес інвентаризації та швидкого контактування з співробітниками.

3.2 Загальна архітектура системи

Архітектуру системи побудовано за трирівневою моделлю, що включає серверну частину, клієнтський інтерфейс та програмний агент як самостійний компонент збору даних(рисунок 3.2).

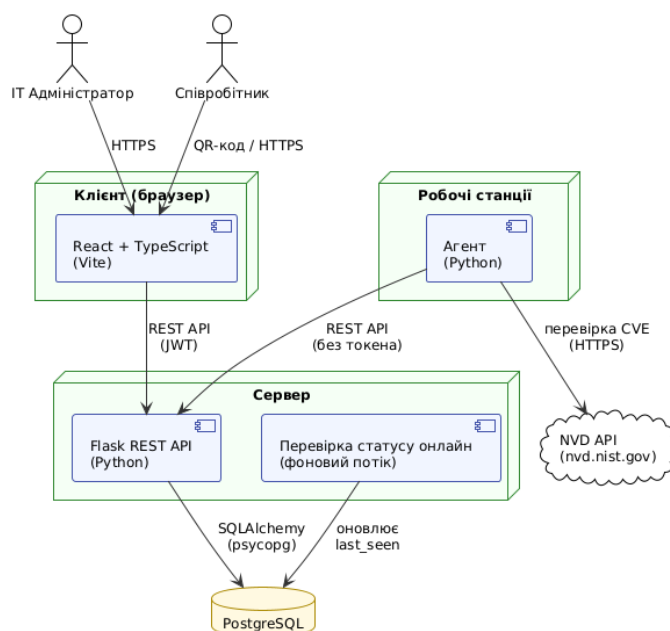


Рисунок 3.2 — Загальна архітектура системи

Серверна частина реалізована мовою Python з використанням Flask і відповідає за обробку всіх запитів, взаємодію з базою даних та виконання бізнес-логіки. Сервер надає REST API, через який клієнтський інтерфейс отримує та надсилає дані. Окремі маршрути відкриті для агента без авторизації, для реєстрації пристроїв та збереження вразливостей. Поряд із основним процесом Flask на сервері запускається фоновий потік, який кожні п'ять хвилин перевіряє доступність пристроїв у мережі та оновлює час їх останньої активності.

Клієнтський інтерфейс розроблено на React з TypeScript і являє собою односторінковий застосунок, що взаємодіє із сервером виключно через HTTP-запити. Інтерфейс не містить власної бізнес-логіки – він відображає дані, отримані від сервера, та надсилає зміни назад.

Агент – окремий Python-скрипт, що виконується на робочих станціях організації. Він не залежить від клієнтського інтерфейсу і взаємодіє безпосередньо із серверним API. Агент самостійно збирає технічні характеристики пристрою, перевіряє встановлене програмне забезпечення на вразливості та передає результати на сервер.

Для зберігання всіх даних використовується PostgreSQL. Структура бази підтримується через моделі SQLAlchemy, що дозволяє описувати таблиці та зв'язки між ними як Python-класи.

3.3 Структура бази даних

База даних системи складається з шести таблиць, кожна з яких відповідає за зберігання певного типу даних(рисунк 3.3).

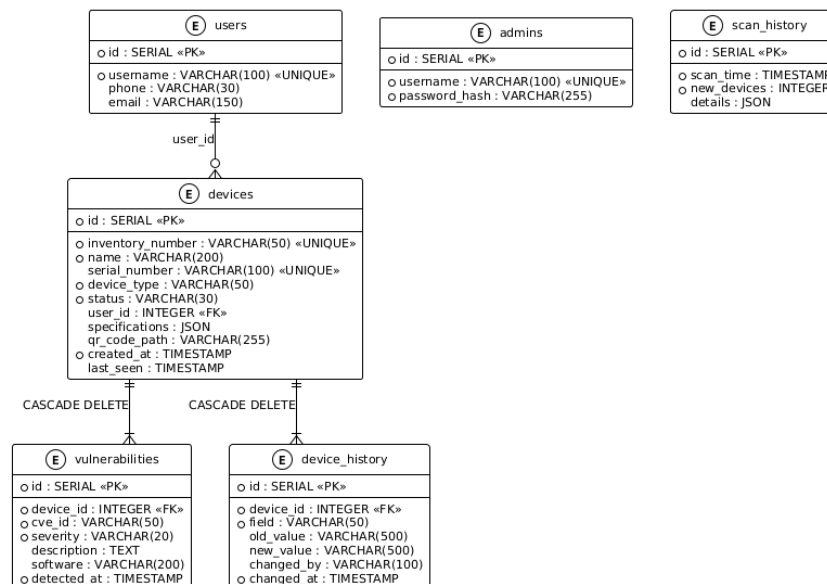


Рисунок 3.3 — ER-діаграма бази даних

Між таблицями визначено зовнішні ключі та каскадне видалення для забезпечення цілісності даних. Операції над таблицями виконуються за допомогою відповідних моделей написаних на мові програмуванні Python. Це дозволяє створювати та редагувати таблиці за допомогою запитів до бази даних.

3.3.1 Модель пристрою

Модель Device є центральною в структурі бази даних і відповідає таблиці devices. Вона зберігає всю інформацію про обладнання організації і є відправною точкою для всіх інших моделей.

Модель містить такі поля:

id – первинний ключ, генерується автоматично.

inventory_number – унікальний інвентарний номер пристрою, не може бути порожнім. Формується агентом на основі MAC-адреси у форматі.

name – назва пристрою, не може бути порожньою. При автоматичній реєстрації заповнюється іменем пристрою в мережі.

serial_number – серійний номер материнської плати, унікальний, може бути порожнім. Унікальність поля запобігає створенню дублікатів при повторній реєстрації одного пристрою.

device_type – тип пристрою: комп'ютер, ноутбук, принтер, комутатор, маршрутизатор, монітор та інші.

status – поточний статус, за замовчуванням “в роботі”. Можливі значення: “в роботі”, “на складі”, “в ремонті”.

user_id – зовнішній ключ на таблицю users, може бути порожнім. Визначає відповідального співробітника.

Specifications – db.JSON, зберігає технічні характеристики пристрою, за замовчуванням порожній словник. Зберігає поля os, cpu, cpu_cores, ram_gb, ip, hostname, serial, os_version, architecture.

qr_code_path – шлях до PNG-файлу QR-коду на сервері, може бути порожнім.

created_at – дата і час створення запису, заповнюється автоматично поточним часом.

last_seen – час останнього контакту з пристроєм, може бути порожнім. Оновлюється агентом та фоновим потоком, що перевіряє онлайн-статус.

Модель також містить властивість `is_online`, яка обраховується динамічно. Якщо з моменту `last_seen` минуло менше 30 хвилин повертає `True`, інакше `False`.

Метод `generate_qr()` формує URL публічної картки пристрою, генерує QR-зображення через бібліотеку `qrcode` та зберігає його як PNG-файл у директорії `static/qr/`.

Метод `to_dict()` повертає словник з усіма полями пристрою включно з даними відповідального, онлайн-статусом та списком вразливостей.

3.3.2 Модель вразливостей

Модель `Vulnerability` відповідає таблиці `vulnerabilities` і зберігає результати перевірки програмного забезпечення на вразливості. Вона дозволяє користувачу бачити на яких конкретних пристроях встановлено небезпечне ПЗ. Модель містить такі поля:

`id` – первинний ключ.

`device_id` – зовнішній ключ на таблицю `devices`, не може бути порожнім. Має відношення `many2one` з моделлю пристроїв `Device`. Кожен пристрій може мати безліч вразливостей.

`cve_id` – ідентифікатор вразливості, визначається номером вразливості взятим з відкритої бази вразливостей `NVD`.

`severity` – рівень критичності: `CRITICAL`, `HIGH` або `MEDIUM`.

`description` – опис вразливості отриманий з бази `NVD`.

`software` – назва та версія програми для якої виявлено вразливість. Отримується за допомогою агента, який збирає дані про встановлене програмне забезпечення.

`detected_at` – дата і час виявлення, заповнюється автоматично.

3.3.3 Модель журналу змін пристрою

Модель DeviceHistory відповідає таблиці device_history і забезпечує журналювання всіх змін у картках пристроїв. Кожна зміна будь-якого поля пристрою автоматично створює новий запис у цій моделі з зазначенням автора і часу. Модель містить такі поля:

id – первинний ключ, створюється автоматично.

device_id – зовнішній ключ на таблицю devices, не може бути порожнім. Має відношення many2one з моделлю пристроїв. Кожен запис журналу закріплюється за певним пристроєм.

field – назва зміненого поля: name, status, device_type, user_id або serial_number.

old_value – попереднє значення пристрою.

new_value – нове значення у тому ж форматі, що і old_value.

changed_at – дата і час внесення змін, заповнюється автоматично.

changed_by – ім'я адміністратора який вніс зміну.

3.3.4 Модель користувача

Модель User відповідає таблиці users і зберігає дані співробітників організації, за якими закріплена техніка. Співробітники не мають доступу до системи – вони є лише об'єктами прив'язки до певного пристрою. Модель містить такі поля:

id – первинний ключ, створюється автоматично.

username – ім'я співробітника, унікальне, не може бути порожнім.

phone – номер телефону, може бути порожнім.

email – електронна адреса, може бути порожньою.

Модель користувача має відношення one2many з моделлю пристроїв. Один працівник, може мати довільну кількість техніки. При видаленні співробітника поле з ідентифікатором обнуляється і пристрої при цьому не видаляються.

3.3.5 Модель адміністратор

Модель Admin відповідає таблиці admins і зберігає облікові записи адміністраторів системи. Лише особи, записи яких є в цій таблиці, можуть увійти до веб-інтерфейсу. Модель не має зв'язків з іншими моделями. Вона містить такі поля:

id – первинний ключ, створюється автоматично.

username – логін адміністратора, унікальний, не може бути порожнім.

password_hash – хеш пароля, не може бути порожнім. Пароль в первинному вигляді не зберігається, лише його зашифрована версія.

Модель містить метод set_password(), який хешує переданий пароль через бібліотеку werkzeug алгоритмом PBKDF2-SHA256 та зберігає результат у поле password_hash.

Метод check_password() приймає чистий пароль і порівнює його з хешем – повертає True або False, в залежності від правильності набору.

3.3.6 Модель історії мережевого сканування

Модель ScanHistory відповідає таблиці scan_history і фіксує результати кожного запуску мережевого сканування. Вона не має зв'язків з іншими моделями і зберігає повністю самостійні записи. Модель містить такі поля:

id – первинний ключ, створюється автоматично.

scan_time – дата і час виконання сканування, заповнюється автоматично.

new_devices – кількість невідомих пристроїв виявлених під час сканування, за замовчуванням 0.

details – повний список знайдених пристроїв, за замовчуванням порожній список. Кожен елемент містить поля IP-адреси, ім'я хоста, MAC-адреси та постачальника.

3.4 Реалізація бізнес-логіки

Бізнес-логіку системи реалізовано у трьох основних блоках: логіка роботи з пристроями на стороні сервера, логіка агента на пристроях та фоновий процес моніторингу. Кожен блок вирішує свою задачу і взаємодіє з іншими через базу даних або REST API.

3.4.1 Логіка роботи з пристроями

Основна логіка обліку пристроїв зосереджена в модулі routes/devices.py. Усі маршрути, крім реєстрації нового пристрою, захищені декоратором token_required і доступні лише авторизованим адміністраторам.

Реєстрація пристрою виконується через маршрут POST /api/devices без авторизації, оскільки цей маршрут викликається агентом. При отриманні запиту сервер перевіряє чи існує вже пристрій з таким серійним номером або інвентарним номером через об'єднаний фільтр з оператором OR. Якщо пристрій знайдено, тоді оновлюються його характеристики та час активності. Якщо ні, тоді створюється новий об'єкт пристрою, після чого зберігається в базі і для нього генерується QR-код через метод generate_qr()[5].

Оновлення пристрою реалізовано через маршрут PUT /api/devices/<inv>. Перед збереженням змін сервер перебирає кожне з полів що може змінюватись: name, status, device_type, user_id, serial_number і порівнює поточне значення з новим. Якщо значення відрізняються, то через допоміжну функцію _format_history_value() обидва значення перетворюються у зручний вигляд для

читання людини і створюється запис в таблиці історії змін пристрою. Це забезпечує автоматичне журналювання без окремого виклику з боку клієнта[5].

Функція `_format_history_value()` обробляє технічні значення перед записом у журнал. Для поля `user_id` вона запитує ім'я співробітника з бази замість числового ідентифікатора. Для полів `status` та `device_type` замінює технічні коди текстовими позначеннями зі словників `STATUS_LABELS` та `TYPE_LABELS`. Порожні значення замінюються на «не вказано».

Журнал змін повертається через маршрут `GET /api/devices/<inv>/history`. Записи сортуються від найновішого до найстарішого і обмежуються п'ятдесятьма останніми записами.

3.4.2 Логіка агента

Агент реалізовано, як нескінченний цикл з інтервалом між ітераціями, що визначається змінною середовища `SEND_INTERVAL`, за замовчуванням це 5 хвилин. На початку кожного циклу функція `get_device_info()` збирає технічні характеристики пристрою. Ім'я в мережі отримується через функцію `platform.node()`, параметри процесора, через бібліотеку `py-cpuinfo`, обсяг пам'яті та кількість ядер через бібліотеку `psutil`[9]. IP-адреса визначається через UDP-сокет: агент відкриває з'єднання до зовнішньої адреси без реального відправлення пакету, операційна система сама вибирає правильний мережевий інтерфейс і повертає локальну IP-адресу через функцію `getsockname()`. Серійний номер материнської плати зчитується через PowerShell на Windows або `dmidecode` на Linux.

Для наочного представлення роботи агента побудовано блок-схему(рисунок 3.4).

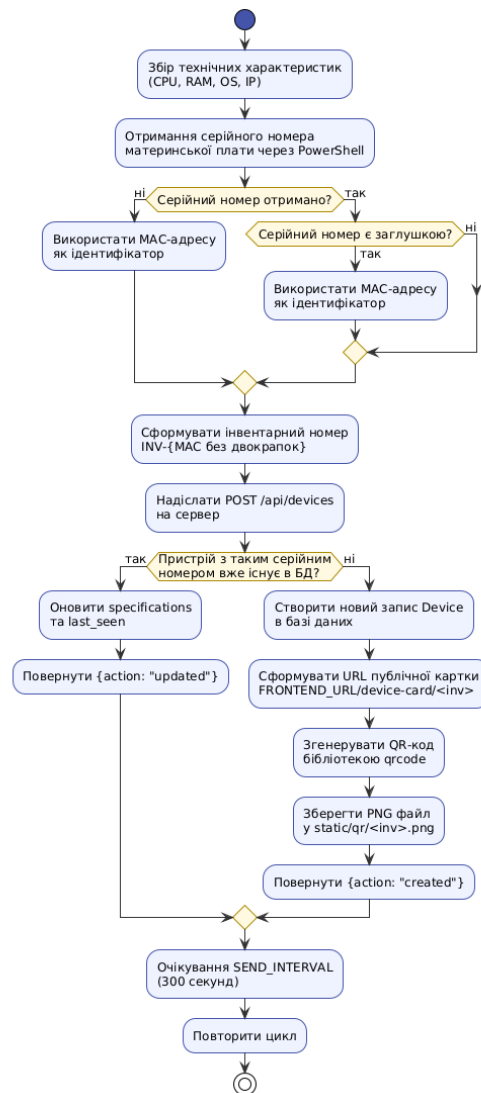


Рисунок 3.4 — Блок схема алгоритму роботи агента

Деякі виробники залишають заглушки замість реального серійного номера, тому агент перевіряє результат на відповідність списку невалідних значень. Якщо серійний номер отримати не вдалось, використовується MAC-адреса через функцію `uuid.getnode()`.

Після збору даних функція `register_device()` надсилає POST-запит на сервер з усіма характеристиками. Інвентарний номер формується з MAC-адреси у форматі `INV-{MAC без двокрапок}`.

Список встановленого програмного забезпечення збирається по-різному залежно від операційної системи. На Windows агент виконує PowerShell-команду,

яка зчитує записи з двох гілок реєстру, для 64-бітних і 32-бітних програм і повертає список з іменами та версіями у форматі CSV. На Linux використовується `dpkg-query` для Debian-подібних систем або `rpm` для Red Hat-подібних.

Функція `get_cpe_key()` визначає чи є програма у внутрішньому словнику `CPE_VENDORS`. Словник містить сімнадцять найпоширеніших продуктів: Chrome, Firefox, Python, Java, Node.js, OpenSSL, OpenSSH, PuTTY та інші з відповідними ідентифікаторами `vendor`(постачальник) та `product`(продукт) за стандартом NVD. Спочатку назва програми переводиться у нижній регістр, після чого з неї видаляються всі пробіли, дефіси, підкреслення та крапки через регулярний вираз. Наприклад, назва «Google Chrome 120» перетворюється у «googlechrome120». По цьому рядку виконується перший прохід: перевіряється чи містить він будь-який ключ зі словника. Цифри при першому проході зберігаються навмисно – це потрібно щоб коректно знаходити продукти з цифрами в назві, наприклад «7zip». Якщо перший прохід нічого не знайшов – з очищеного рядка додатково видаляються всі цифри і виконується другий прохід. Це дозволяє знайти «chrome» у рядку «googlechrome120» навіть якщо номер версії потрапив у назву. Якщо жоден прохід не дав результату, функція повертає порожній рядок і програма не перевіряється на вразливості.

3.4.3 Перевірка вразливостей

Перевірка вразливостей виконується функцією `check_cve()` для кожної знайденої програми зі словника `CPE_VENDORS`. Функція формує CPE-рядок у форматі `cpe:2.3:a:{vendor}:{product}:{version}:::.*`, відповідно до стандарту NIST[19] та надсилає GET-запит до публічного API бази NVD[10]. Між запитами витримується пауза шість секунд відповідно до обмеження NVD у п'ять запитів за тридцять секунд без API-ключа. Якщо сервер NVD повертає код 429 (перевищення ліміту), програма пропускається і цикл продовжується.

Для кожної знайденої CVE агент перевіряє чи версія встановленої програми потрапляє у вразливий діапазон. Рядки версій перетворюються у кортежі цілих чисел для коректного порівняння: рядок «120.0.6099» перетворюється у (120, 0, 6099). При порівнянні версій різної глибини коротший кортеж доповнюється нулями. Підтримуються чотири типи діапазонів: versionStartIncluding, versionStartExcluding, versionEndIncluding та versionEndExcluding. Вразливості раніше 2010 року не враховуються, для зменшення діапазону пошуку. Рівень критичності визначається з метрик CVSS v3.1, v3.0 або v2 у порядку пріоритету[20].

Для наочного представлення алгоритму роботи перевірки вразливостей створено блок-схему(рисунок 3.7).

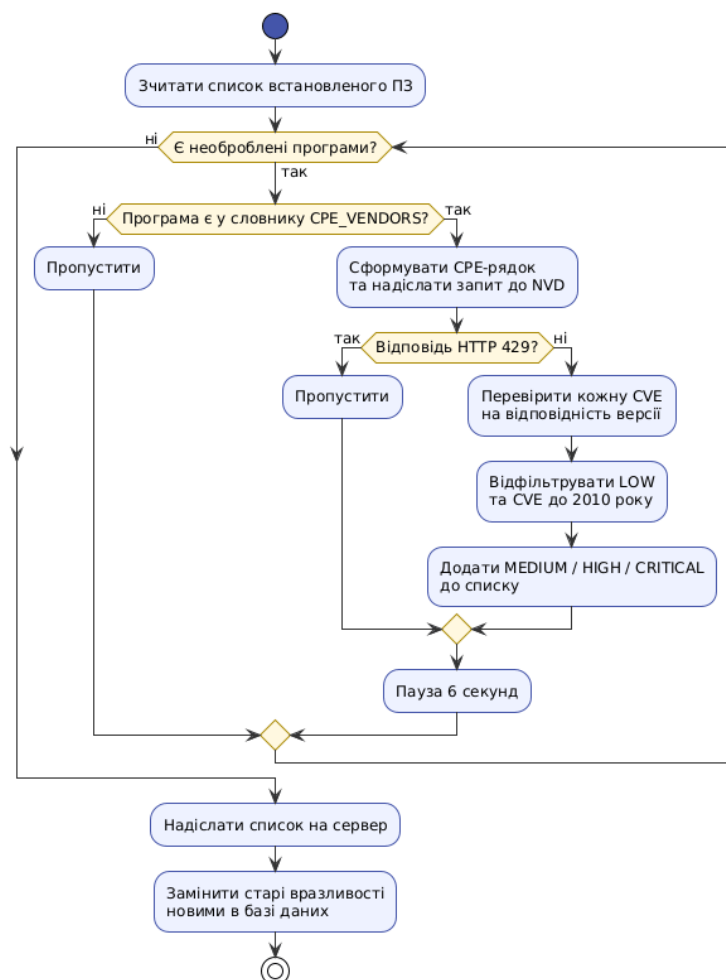


Рисунок 3.7 — Блок схема алгоритму перевірки вразливостей

Зібраний список вразливостей передається функцією `send_vulnerabilities()` на сервер через `POST /api/vulnerabilities`. Сервер повністю замінює попередній список для даного пристрою на новий – це гарантує, що дані завжди відповідають поточному стану встановленого ПЗ.

3.4.4 Мережеве сканування та перевірка статусу онлайн

Мережеве сканування запускається адміністратором через маршрут `POST /api/scan/run`. Функція `_run_nmap()` виконує команду `nmap -sn` з вказаним мережевим діапазоном[11]. Прапор `-sn` означає `ping`-сканування без перебору портів, що значно прискорює процес і мінімізує навантаження на мережу. На Windows шлях до `nmap` визначається автоматично через функцію `_find_nmap()`, яка перевіряє стандартні директорії встановлення. Результат виводу `nmap` розбирається рядок за рядком: рядки «`Nmap scan report for`» ідентифікують новий хост, рядки «`MAC Address:`» додають апаратну адресу та виробника.

Після сканування функція `_build_known_ids()` збирає ідентифікатори всіх відомих пристроїв з бази: серійні номери, мережеві імена з суфіксами і без (`.local`, `.lan`, `.home`), інвентарні номери та IP-адреси. Після цього, знайдені хости порівнюються з цим набором через функцію `_filter_unknown()`, пристрої, що не збіглися жодним ідентифікатором, повертаються адміністратору як невідомі.

Перевірку онлайн статусу реалізовано як фоновий потік, що запускається разом із сервером через функцію `start()`.

Кожні п'ять хвилин він збирає список пристроїв з IP-адресами в полі `specifications` та надсилає системний `ping` до кожного. На Windows використовується команда `ping -n 1 -w 1000`, на Linux `ping -c 1 -W 1`. Якщо пристрій відповів – оновлюється поле `last_seen` в базі даних. Перевірка виконується в окремому потоці і не блокує обробку запитів основним сервером.

3.5 Реалізація клієнтського інтерфейсу

Клієнтський інтерфейс системи реалізовано у вигляді односторінкового веб-застосунку на основі бібліотеки React 19 з TypeScript. Інтерфейс взаємодіє з серверною частиною виключно через HTTP-запити до REST API і не містить власної бізнес-логіки, він відображає дані отримані від сервера та надсилає зміни назад. Маршрутизація між сторінками реалізована через React Router 7, стилізація Bootstrap 5 та React-Bootstrap. Збірка проєкту виконується інструментом Vite.

3.5.1 Авторизація та управління станом

Стан авторизації централізовано керується через React Context. Компонент AuthProvider обгортає весь застосунок і надає дочірнім компонентам доступ до токена, даних адміністратора та функцій login і logout.

При ініціалізації провайдер зчитує збережений токен з localStorage та перевіряє його актуальність запитом до серверного маршруту /api/auth/me. Якщо токен є дійсним дані адміністратора завантажуються в стан, якщо ні токен видаляється з localStorage і користувач вважається не авторизованим.

Для автоматичного додавання токена до кожного запиту та обробки відповідей з кодом 401 налаштовано Axios-interceptors. Request interceptor додає заголовок Authorization до всіх запитів крім публічних маршрутів. Response interceptor при отриманні відповіді 401 очищає токен та перенаправляє адміністратора на сторінку входу.

Функція login надсилає логін і пароль на сервер та при успішній відповіді зберігає токен і дані адміністратора в стані та localStorage. При помилці повертає повідомлення про причину відмови.

Захист сторінок від несанкціонованого доступу реалізовано через компонент PrivateRoute

Він перевіряє наявність токена і при його відсутності перенаправляє на сторінку входу. Поки виконується перевірка токена відображається індикатор завантаження.

3.5.2 Реалізація сторінок інтерфейсу

Головна сторінка завантажує повний список пристроїв при монтуванні компонента через виклик `devicesApi.getAll()`. Фільтрація виконується повністю на стороні клієнта без додаткових запитів до сервера одночасно за трьома критеріями: рядком пошуку по назві, інвентарному номеру та імені відповідального, фільтром за статусом пристрою та фільтром за онлайн-станом.

Для відображення актуальних лічильників у навігаційному меню реалізовано хук `useNavbarAlerts`.

Він підписаний на зміну маршруту і при кожному переході між сторінками паралельно запитує кількість критичних вразливостей рівня `CRITICAL` та `HIGH`, а також кількість нових пристроїв з останнього мережевого сканування. Результати відображаються у вигляді кольорових лічильників поруч з відповідними пунктами меню, що дозволяє адміністратору миттєво бачити поточний стан системи без переходу на відповідні сторінки.

Сторінка вразливостей завантажує всі CVE системи та надає фільтрацію за рівнем критичності. Кількість вразливостей кожного рівня підраховується через метод `reduce` і відображається у вигляді карток-лічильників: `CRITICAL`, `HIGH`, `MEDIUM` та `LOW`. Натискання на картку встановлює фільтр, повторне натискання скидає його. Колір відображення кожного рівня визначається окремою функцією `sevColor`, яка повертає відповідну CSS-змінну.

Сторінка мережевого сканування при завантаженні паралельно отримує історію сканувань та список усіх зареєстрованих пристроїв через `Promise.all`, що скорочує час очікування порівняно з послідовними запитами. Для визначення чи пристрій зі сканування вже є в базі формується список відомих ідентифікаторів, що

включає серійні номери, назви, інвентарні номери, IP-адреси та мережеві імена. Функція `isNowKnown` порівнює кожен знайдений пристрій з цим набором по кількох параметрах одночасно, включаючи варіанти мережевого імені з суфіксами та без. Запуск нового сканування обробляє як успішний результат так і помилки, відображаючи їх у модальному вікні.

Таким чином, клієнтський інтерфейс системи не лише відображає дані, але й тісно взаємодіє з серверною логікою, токеном авторизації та станом застосунку. Такий підхід дозволяє створити не статичну сторінку, а адаптивне середовище для роботи адміністратора, яке автоматично оновлюється та реагує на зміни в інфраструктурі організації.

4 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

У даному розділі описано процес встановлення та налаштування системи, а також основні сценарії роботи адміністратора з веб-інтерфейсом. Розглянуто кроки розгортання серверної частини, клієнтського інтерфейсу та програмного агента, наведено скріншоти основних сторінок системи з поясненням їх функціоналу.

4.1 Встановлення та налаштування системи

Система складається з трьох незалежних компонентів, кожен з яких встановлюється окремо: серверна частина, клієнтський інтерфейс та агент. Серверна частина і клієнтський інтерфейс розгортаються на одному сервері або робочій станції адміністратора, агент на кожному комп'ютері організації який потрібно взяти в облік.

Встановлення серверної частини

Перед встановленням на сервері має бути встановлено Python версії 3.10 або вище та PostgreSQL версії 14 або вище. Для роботи мережевого сканування також потрібна утиліта nmap.

Першим кроком створюється база даних. У PostgreSQL створюється нова база з назвою `it_asset` та користувач з правами на неї. Параметри підключення записуються у файл `.env` у корені проєкту. Файл середовища `.env` містить такі параметри:

`DATABASE_URL` – рядок підключення до PostgreSQL у форматі `postgresql+psycopg://користувач:пароль@хост:порт/назва_бази`. Відповідає за те з якою базою даних працюватиме сервер.

`FRONTEND_URL` – адреса клієнтського інтерфейсу, наприклад `http://192.168.1.10:5173`. Використовується при генерації QR-кодів, саме ця адреса кодується в QR і повинна бути доступна з телефонів у мережі організації.

`SECRET_KEY` – довільний рядок не менше 32 символів для підпису JWT-токенів. Обов'язково змінюється перед розгортанням, якщо залишити значення за замовчуванням, будь-хто з доступом до коду зможе генерувати валідні токени авторизації.

`NETWORK_RANGE` – діапазон IP-адрес для мережевого сканування у форматі CIDR, наприклад `192.168.1.0/24`. Визначає яку підмережу сканує система при пошуку невідомих пристроїв.

`PG_BIN` – шлях до папки `bin` PostgreSQL на Windows, наприклад `C:\Program Files\PostgreSQL\18\bin`. Потрібен лише на Windows, щоб Python міг знайти бібліотеки PostgreSQL.

Після заповнення файлу середовища `.env` встановлюються Python-залежності командою `pip install -r requirements.txt` у папці `backend`. Команда встановлює Flask, SQLAlchemy, PyJWT, pycorg, qrcode та інші необхідні бібліотеки.

Далі запускається сервер командою `python app.py` з папки `backend`. При першому запуску сервер автоматично створює всі таблиці в базі даних та обліковий запис адміністратора з логіном `admin` і паролем `admin123`. Сервер запускається на порту `5001` і приймає підключення з будь-якої адреси в мережі.

Встановлення клієнтського інтерфейсу

У папці `frontend` встановлюються Node.js-залежності командою `npm install`. У файлі `.env` у папці `frontend` вказується адреса сервера через змінну `VITE_API_URL`, наприклад `http://192.168.1.10:5001`. Ця змінна визначає на який сервер надсилатиме запити браузер. Після цього інтерфейс запускається командою `npm run dev` для режиму розробки або збирається командою `npm run build` для розгортання як статичний сайт. У режимі розробки інтерфейс доступний на порту `5173`.

Встановлення агента

Папка `agent` разом із файлом середовища копіюється на комп'ютер організації. У файлі агента вказуються два параметри:

`API_URL` – адреса серверної частини, наприклад `http://192.168.1.10:5001`. Агент надсилає дані про пристрій саме на цю адресу.

SEND_INTERVAL – інтервал між циклами роботи агента в секундах, за замовчуванням 5 хвилин. Визначає як часто агент оновлює дані про пристрій на сервері.

Для запуску агента без вікна консолі на Windows використовується VBS-скрипт, який знаходиться у папці agent. Подвійний клік на файлі start_agent_hidden.vbs запускає агента у прихованому режимі через pythonw.exe. Перевірити що агент запущений можна через Диспетчер завдань — у списку процесів має бути присутній pythonw.exe.

Для автоматичного запуску агента при кожному вході користувача в систему необхідно розмістити ярлик на VBS-скрипт у папці автозапуску. Папка відкривається через команду shell:startup у вікні «Виконати» (Win + R). Після цього агент стартуватиме автоматично без жодних дій з боку користувача.

Якщо потрібен запуск агента до входу користувача в систему — наприклад для комп'ютерів, що працюють без постійного входу, використовується Планувальник завдань Windows. Створюється нове завдання з тригером «При запуску комп'ютера», дією запуску pythonw.exe з аргументом у вигляді шляху до agent.py та обліковим записом SYSTEM. Це забезпечує роботу агента незалежно від того чи виконав хтось вхід у систему.

4.2 Перша реєстрація пристрою агентом

Після запуску агент протягом кількох секунд збирає характеристики пристрою і надсилає їх на сервер. Пристрій автоматично з'являється в списку обладнання на головній сторінці інтерфейсу без жодних дій з боку системного адміністратора. Одночасно з реєстрацією пристрою на сервері генерується QR-код для нього. Якщо агент вже реєстрував цей комп'ютер раніше, тоді запис оновлюється, а новий не створюється.

Після реєстрації агент переходить до перевірки встановленого програмного забезпечення. Він зчитує список програм з реєстру Windows і звіряє кожну зі своїм

словником відомих продуктів. Для знайдених програм надсилаються запити до бази вразливостей NVD. Весь процес займає кілька хвилин залежно від кількості встановленого ПЗ. Після завершення виявлені вразливості з'являються в системі і стають доступними адміністратору.

Надалі агент повторює цей цикл кожні п'ять хвилин — оновлює характеристики пристрою, час активності та список вразливостей.

4.3 Вхід до системи

Для входу адміністратор відкриває браузер та переходить на адресу веб-інтерфейсу. На сторінці входу необхідно ввести логін і пароль (рисунок 4.1).

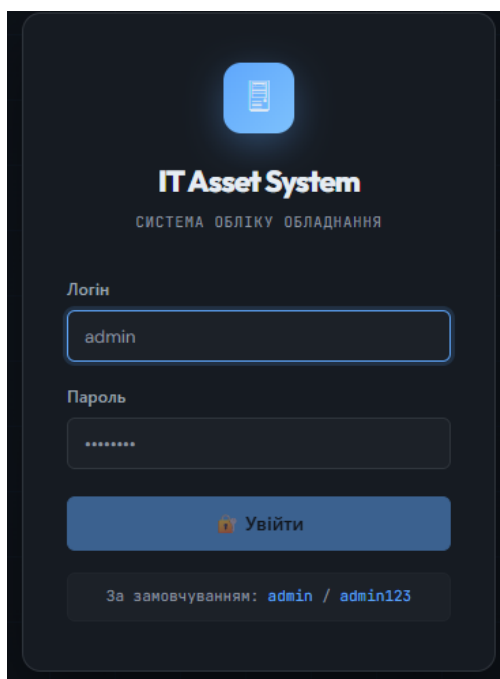
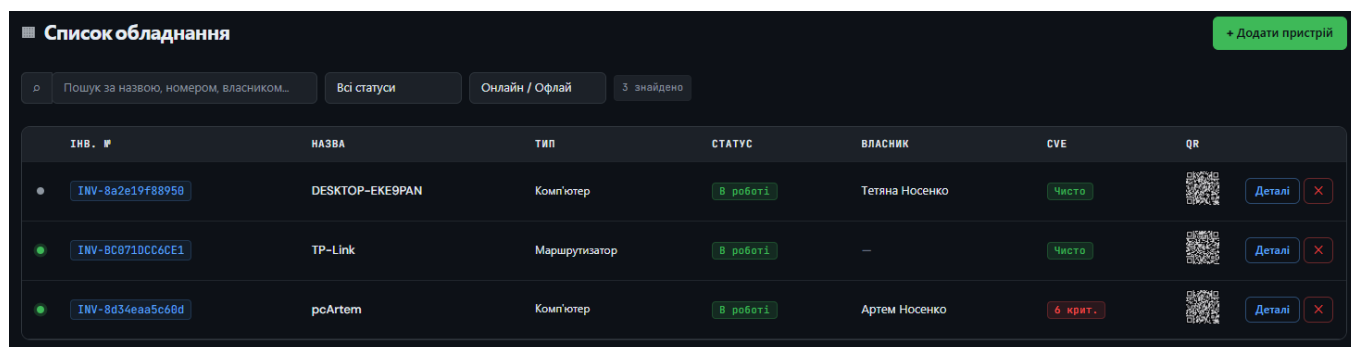


Рисунок 4.1 — Сторінка входу до системи

Кнопка входу недоступна поки поля не заповнені. Після успішної авторизації токен зберігається в браузері і адміністратор залишається в системі протягом 24 годин без необхідності повторного входу. При введенні неправильних даних відображається повідомлення про помилку.

4.4 Головна сторінка та список обладнання

Після входу відкривається головна сторінка зі списком усього обладнання організації (рисунок 4.2).



ІНВ. №	НАЗВА	ТИП	СТАТУС	ВЛАСНИК	CVE	QR
INV-8a2e19f88950	DESKTOP-EKE9PAN	Комп'ютер	В роботі	Тетяна Носенко	Чисто	 Деталі X
INV-BC0710CC6CE1	TP-Link	Маршрутизатор	В роботі	—	Чисто	 Деталі X
INV-8d34eaa5c6bd	pcArtem	Комп'ютер	В роботі	Артем Носенко	В крит.	 Деталі X

Рисунок 4.2 — Головна сторінка зі списком обладнання

Таблиця відображає інвентарний номер, назву пристрою, тип, статус, відповідального співробітника, кількість вразливостей та онлайн-статус. Зелена позначка біля пристрою означає, що він активний і зараз працює, сіра, що пристрій недоступний або вимкнений.

Вгорі сторінки розміщено рядок пошуку та два фільтри. Пошук виконується одночасно за назвою пристрою, інвентарним номером та іменем відповідального. Фільтр за статусом дозволяє відобразити лише пристрої з певним статусом: в роботі, на складі або на ремонті. Фільтр онлайн/офлайн звужує список до активних або недоступних пристроїв. Пошук і фільтрація виконуються миттєво без звернення до сервера.

4.5 Картка пристрою

Для перегляду детальної інформації адміністратор натискає на рядок пристрою в таблиці. Відкривається картка пристрою (рисунок 4.3), яка складається з двох панелей.

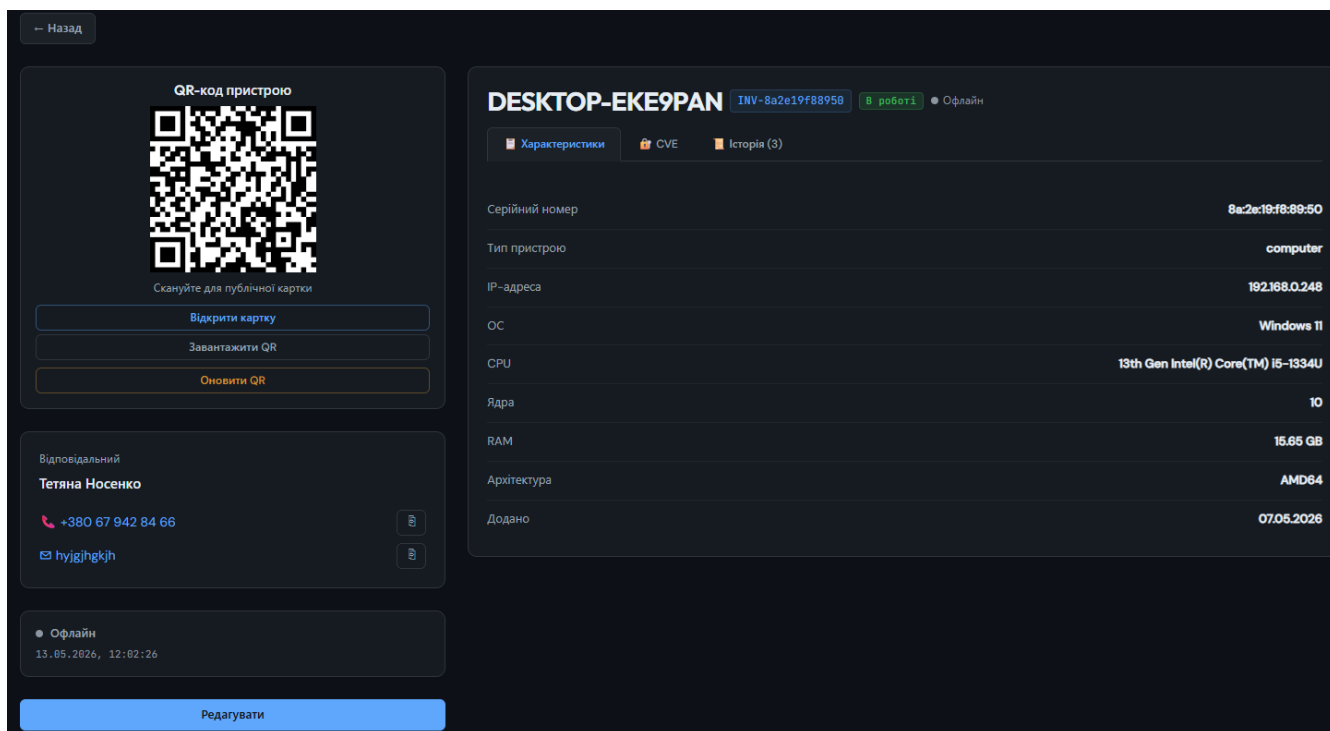


Рисунок 4.3 — Картка пристрою

На лівій панелі відображається QR-код пристрою з кнопками для відкриття публічної сторінки в браузері та завантаження зображення QR у вигляді PNG-файлу. Також тут показуються контактні дані відповідального співробітника: ім'я, телефон та email з кнопками копіювання поруч з кожним полем. При натисканні кнопки дані копіюються в буфер обміну. Нижче відображається онлайн-статус пристрою та час останнього контакту.

Права панель містить три вкладки. Вкладка «Характеристики» показує операційну систему, модель процесора, кількість ядер, обсяг оперативної пам'яті, IP-адресу та дату додавання до системи.

Вкладка «Вразливості» містить таблицю знайдених CVE з рівнем критичності, назвою програми та описом(рисунок 4.4).

pcArtem	INV-8d34eaa5c60d	В роботі	Онлайн
Характеристики	CVE (6)	Історія (4)	
CVE	РІВЕНЬ	ПЗ	ОПИС
CVE-2022-29072	Висока	7-Zip 19.00	7-Zip through 21.07 on Windows allows privilege escalation and command execution when a file with th...
CVE-2023-31102	Висока	7-Zip 19.00	Ppmd7.c in 7-Zip before 23.00 allows an integer underflow and invalid read operation via a crafted 7...
CVE-2024-11477	Висока	7-Zip 19.00	7-Zip Zstandard Decompression Integer Underflow Remote Code Execution Vulnerability. This vulnerabil...
CVE-2025-0411	Висока	7-Zip 19.00	7-Zip Mark-of-the-Web Bypass Vulnerability. This vulnerability allows remote attackers to bypass the...
CVE-2025-53816	Висока	7-Zip 19.00	7-Zip is a file archiver with a high compression ratio. Zeroes written outside heap buffer in RAR5 h...
CVE-2025-53817	Висока	7-Zip 19.00	7-Zip is a file archiver with a high compression ratio. 7-Zip supports extracting from Compound Docu...

Рисунок 4.4 — Таблиця знайдених вразливостей

Якщо виявлено вразливості рівня CRITICAL або HIGH — вгорі вкладки відображається червоне попередження. Кожен код CVE є посиланням на офіційну сторінку NVD.

Вкладка «Журнал змін» показує хронологію всіх змін у картці пристрою: яке поле змінили, попереднє та нове значення, хто і коли вніс зміну(рисунок 4.5).

pcArtem

INV-8d34eaa5c60d

В роботі

Онлайн

Характеристики

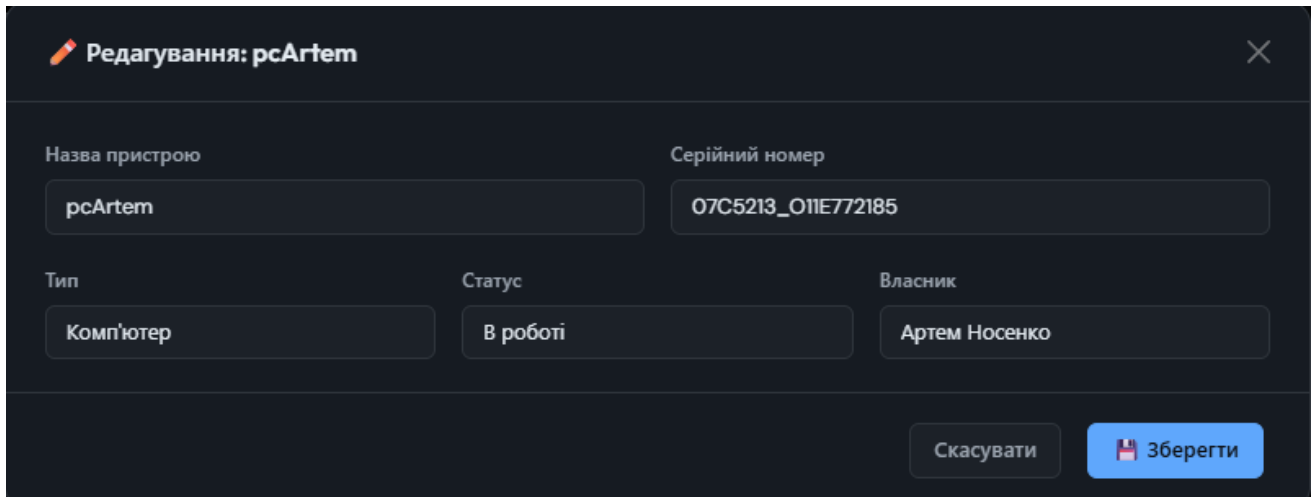
CVE (6)

Історія (4)

ПОЛЕ	БУЛО	СТАЛО	ХТО	КОЛИ
Власник	не вказано	Артем Носенко	admin	13.05.2026, 12:01:42

Рисунок 4.5 — Журнал змін пристрою

Для редагування пристрою адміністратор натискає кнопку «Редагувати». Відкривається форма де можна змінити назву, серійний номер, тип, статус та відповідального співробітника (рисунок 4.6).



Назва пристрою	Серійний номер	Тип	Статус	Власник
pcArtem	07C5213_011E772185	Комп'ютер	В роботі	Артем Носенко

Рисунок 4.6 — Форма редагування пристрою

Усі зміни автоматично фіксуються в журналі змін без додаткових дій.

4.6 Додавання пристрою вручну

Для ручного додавання пристрою адміністратор натискає кнопку «Додати пристрій», розташовану на головній сторінці системи управління обладнанням. Ця функція призначена для випадків, коли автоматичне виявлення пристрою в мережі неможливе або недоцільне, наприклад, якщо пристрій ще не підключений до мережі або потребує попередньої реєстрації в системі до фізичного введення в експлуатацію.

Після натискання кнопки відкривається діалогова форма, яка містить такі поля для заповнення: інвентарний номер, назва пристрою, серійний номер, IP-адреса, тип пристрою та його поточний статус (рисунок 4.7).

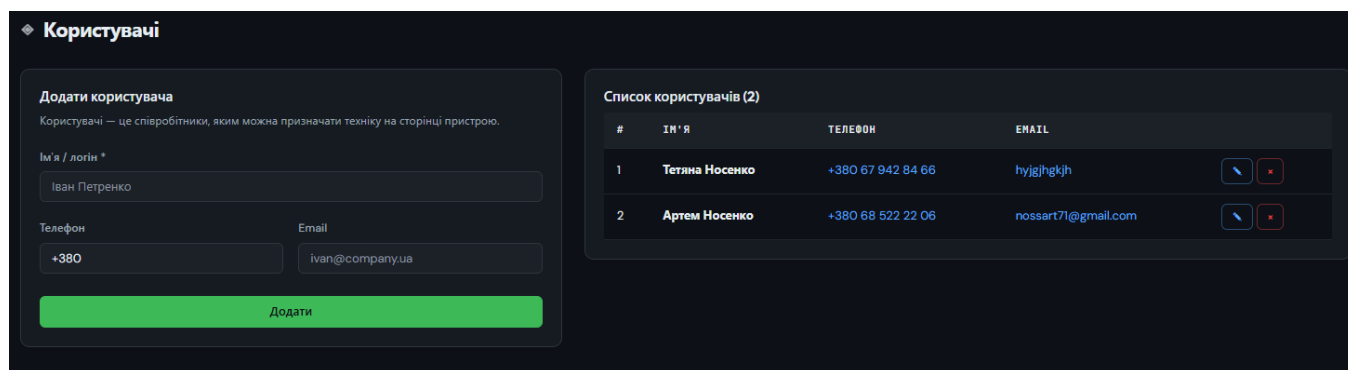
Рисунок 4.7 — Форма додавання пристрою вручну

Форма розроблена з урахуванням зручності використання та мінімізації часу на введення даних: поля згруповані логічно за своїм призначенням, що дозволяє адміністратору швидко та без помилок внести всі необхідні відомості про новий пристрій.

Обов'язковими для заповнення є лише поля інвентарного номера та назви пристрою — ці два атрибути є мінімально необхідними для однозначної ідентифікації обладнання в системі. Решта полів, зокрема серійний номер, IP-адреса, тип та статус, є необов'язковими і можуть бути заповнені або уточнені пізніше, у процесі подальшого обслуговування пристрою. Після збереження введених даних пристрій одразу відображається в загальній таблиці обліку обладнання, а система автоматично генерує для нього унікальний QR-код, який у подальшому використовується для швидкої ідентифікації та отримання інформації про пристрій.

4.7 Управління персоналом

Сторінка управління персоналом дозволяє додавати, редагувати та видаляти записи співробітників(рисунок 4.8).



Користувачі

Додати користувача
Користувачі — це співробітники, яким можна призначати техніку на сторінці пристрою.

Ім'я / логін *

Іван Петренко

Телефон

+380

Email

ivan@company.ua

Додати

Список користувачів (2)

#	ІМ'Я	ТЕЛЕФОН	EMAIL	
1	Тетяна Носенко	+380 67 942 84 66	hyjglhgkjh	 
2	Артем Носенко	+380 68 522 22 06	nossart71@gmail.com	 

Рисунок 4.8 — Сторінка управління персоналом

Зліва розміщено форму для додавання нового співробітника з полями імені, телефону та email. Праворуч таблиця з усіма співробітниками. Редагування виконується безпосередньо в рядку таблиці. Телефонний номер форматується автоматично у міжнародний формат при введенні. При видаленні співробітника система попереджає що закріплена за ним техніка залишиться без відповідального.

4.8 Перегляд вразливостей

Сторінка вразливостей відображає всі знайдені CVE по всіх пристроях, зареєстрованих у системі. Це централізоване представлення дозволяє адміністратору отримати повну картину стану захищеності інфраструктури без необхідності переглядати кожен пристрій окремо. Вгорі сторінки розміщено чотири інформаційні картки-лічильники, які відображають кількість вразливостей кожного рівня критичності: критичний, високий, середній та низький. Натискання на будь-яку з карток миттєво фільтрує таблицю за обраним рівнем критичності, що

значно спрощує пріоритизацію усунення загроз; повторне натискання на ту саму картку скидає фільтр і повертає повний список вразливостей (рисунок 4.9).

Вразливості безпеки

0 КРИТИЧНА

6 ВИСОКА

0 СЕРЕДНЯ

0 НИЗЬКА

⚠ Виявлено 6 критичних та високих вразливостей. Потрібне негайне оновлення безпеки на відповідних пристроях!

Всього: 6

CVE ID	РІВЕНЬ	ПЗ	ПРИСТРІЙ	ОПИС	ВИЯВЛЕНО
CVE-2025-53817	Висока	7-Zip 19.00	INV-8d34eaa5c60d	7-Zip is a file archiver with a high compression ratio. 7-Zip supports extracting from Compound Documents. Prior to vers...	13.05.2026
CVE-2025-53816	Висока	7-Zip 19.00	INV-8d34eaa5c60d	7-Zip is a file archiver with a high compression ratio. Zeros written outside heap buffer in RAR5 handler may lead to m...	13.05.2026
CVE-2025-0411	Висока	7-Zip 19.00	INV-8d34eaa5c60d	7-Zip Mark-of-the-Web Bypass Vulnerability. This vulnerability allows remote attackers to bypass the Mark-of-the-Web pro...	13.05.2026
CVE-2024-11477	Висока	7-Zip 19.00	INV-8d34eaa5c60d	7-Zip Zstandard Decompression Integer Underflow Remote Code Execution Vulnerability. This vulnerability allows remote at...	13.05.2026
CVE-2023-31102	Висока	7-Zip 19.00	INV-8d34eaa5c60d	Ppmd7.c in 7-Zip before 23.00 allows an integer underflow and invalid read operation via a crafted 7z archive.	13.05.2026
CVE-2022-29072	Висока	7-Zip 19.00	INV-8d34eaa5c60d	7-Zip through 2107 on Windows allows privilege escalation and command execution when a file with the 7z extension is d...	13.05.2026

Рисунок 4.9 — Сторінка вразливостей пристроїв

Такий підхід до організації інтерфейсу дозволяє адміністратору зосередитись насамперед на найбільш небезпечних вразливостях і опрацьовувати їх у порядку зменшення ступеня ризику. Класифікація за рівнями критичності відповідає загальноприйнятій системі оцінювання CVSS, що забезпечує сумісність із зовнішніми інструментами аналізу безпеки та звітності.

Кожен рядок таблиці містить інформацію про виявлену вразливість: назву пристрою, унікальний код CVE, рівень критичності, назву та версію програмного забезпечення, в якому виявлено вразливість, а також дату її виявлення. Для зручності навігації назва пристрою є активним посиланням, що веде безпосередньо на його детальну картку в системі, а код CVE посиланням на офіційну сторінку бази даних NVD, де адміністратор може ознайомитися з технічним описом вразливості, рекомендаціями щодо її усунення та додатковими відомостями від виробника програмного забезпечення.

4.9 Мережеве сканування

Сторінка мережевого сканування має двопанельний вигляд. Ліворуч розміщено список усіх попередніх сканувань з датою та кількістю знайдених невідомих пристроїв. Праворуч відображаються деталі обраного сканування, таблиця з усіма знайденими хостами, їх IP, мережевим іменем та MAC-адресою. Пристрої які вже є в базі позначаються зеленим маркером «В базі», невідомі кнопкою «До бази». Для запуску нового сканування адміністратор натискає кнопку «Сканувати мережу». Система попереджає що процес може зайняти деякий час.

Вигляд сторінки мережевого сканування зображено на рисунку 4.10.

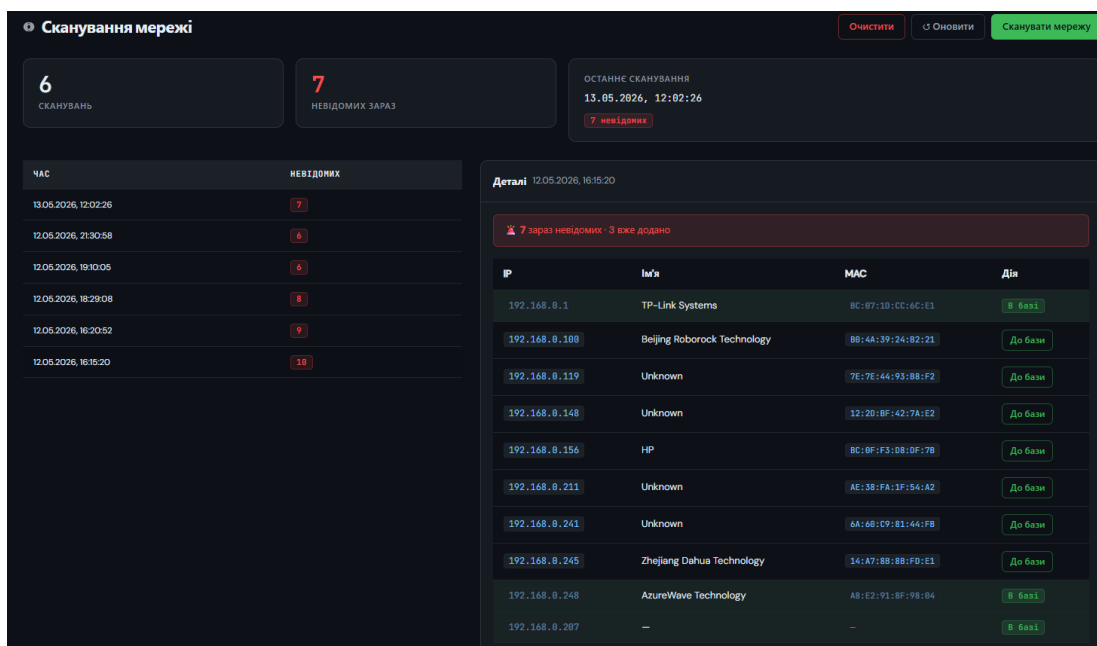


Рисунок 4.10 — Сторінка мережевого сканування

Після завершення відображається вікно з підсумком: скільки пристроїв знайдено і скільки з них невідомих. Для кожного невідомого пристрою кнопка «До бази» відкриває форму додавання з автоматично заповненими полями: IP-адресою, мережевим іменем та MAC-адресою як серійним номером.

4.10 Публічна картка пристрою

Публічна картка відкривається при скануванні QR-коду і не потребує авторизації. Сторінка відображає назву пристрою, інвентарний номер, тип, статус, ім'я відповідального співробітника з телефоном та email, основні технічні характеристики та онлайн-статус з часом останнього контакту.

Вигляд публічної картки пристрою зображено на рисунку 4.11.

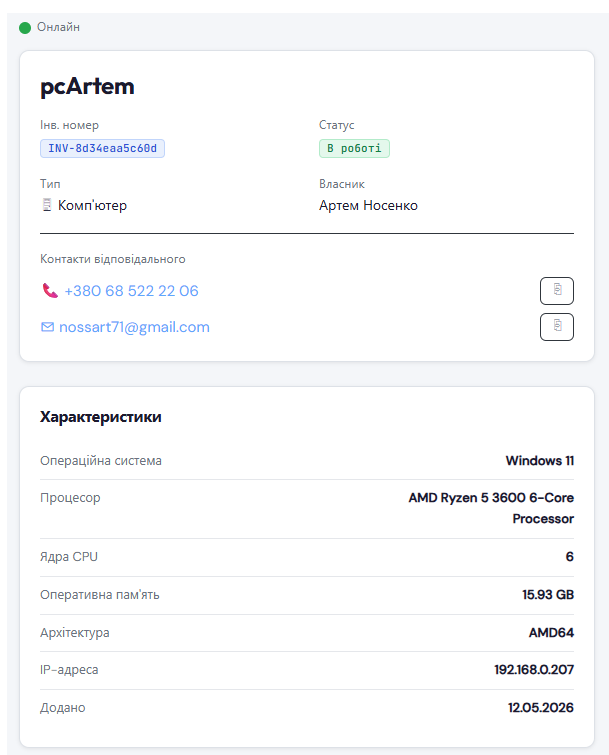


Рисунок 4.11 — Публічна картка пристрою

Поруч з телефоном та email розміщено кнопки копіювання. Якщо пристрій не знайдено в базі відображається повідомлення про помилку.

ВИСНОВКИ

У ході виконання дипломної роботи було успішно спроектовано та реалізовано веб-орієнтовану систему обліку комп'ютерного обладнання, яка охоплює всі ключові функціональні напрями від автоматичної реєстрації пристроїв до виявлення вразливостей програмного забезпечення. Створено зручний інтерфейс адміністратора, адаптований під потреби ІТ-відділів організацій, та реалізовано повноцінну серверну логіку з підтримкою агентного збору даних, мережевого сканування та інтеграції з міжнародною базою вразливостей NVD.

У процесі роботи було поглиблено знання в таких напрямках: проектування трирівневої архітектури веб-застосунків, розробка REST API на Python з використанням Flask, побудова клієнтських інтерфейсів на React з TypeScript, робота з реляційною базою даних PostgreSQL через ORM SQLAlchemy, а також реалізація фонових процесів та мережевого сканування засобами Python. Значну увагу було приділено безпеці системи: хешуванню паролів, захисту маршрутів через JWT та розмежуванню публічного і приватного доступу.

Результатом стало повнофункціональне рішення, яке не лише відповідає поставленим вимогам, а й може бути розширене та адаптоване під потреби організацій різного масштабу. Проект продемонстрував, що поєднання агентного підходу до збору даних, автоматичної перевірки вразливостей та зручного веб-інтерфейсу дозволяє ефективно вирішити задачу управління ІТ-інфраструктурою без залучення дорогих комерційних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Snipe-IT. URL: <https://snipeitapp.com/> (дата звернення: 10.05.2026).
2. Офіційний сайт GLPI Project. URL: <https://glpi-project.org/> (дата звернення: 10.05.2026).
3. Офіційний сайт Lansweeper. URL: <https://www.lansweeper.com/> (дата звернення: 10.05.2026).
4. Python 3 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 12.05.2026).
5. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. USA, Sebastopol : O'Reilly Media, 2018. 316 p.
6. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 13.05.2026).
7. Kaliski B. PKCS #5: Password-Based Cryptography Specification Version 2.0: RFC 2898. URL: <https://datatracker.ietf.org/doc/html/rfc2898> (дата звернення: 13.05.2026).
8. Copeland R. Essential SQLAlchemy: Mapping Python to Databases. 2nd ed. USA, Sebastopol : O'Reilly Media, 2016. 322 p.
9. psutil Documentation. URL: <https://psutil.readthedocs.io/> (дата звернення: 12.05.2026).
10. National Vulnerability Database. NVD Developers – Vulnerabilities API. NIST, 2024. URL: <https://nvd.nist.gov/developers/vulnerabilities> (дата звернення: 15.05.2026).
11. Lyon G. F. Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. USA, Sunnyvale : Insecure.com LLC, 2009. 468 p.
12. qrcode. QR Code Image Generator for Python. URL: <https://pypi.org/project/qrcode/> (дата звернення: 12.05.2026).

13. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. USA, Sebastopol : O'Reilly Media, 2020. 310 p.
14. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. USA, Sebastopol : O'Reilly Media, 2019. 352 p.
15. React Router Documentation. URL: <https://reactrouter.com/> (дата звернення: 14.05.2026).
16. Axios HTTP Client Documentation. URL: <https://axios-http.com/docs/intro> (дата звернення: 14.05.2026).
17. Vite Documentation. URL: <https://vite.dev/> (дата звернення: 14.05.2026).
18. Obe R., Hsu L. PostgreSQL: Up and Running. 3rd ed. USA, Sebastopol : O'Reilly Media, 2017. 346 p.
19. NIST. Common Platform Enumeration (CPE) Specification. URL: <https://nvd.nist.gov/products/cpe> (дата звернення: 15.05.2026).
20. FIRST. Common Vulnerability Scoring System v3.1: Specification Document. URL: <https://www.first.org/cvss/specification-document> (дата звернення: 15.05.2026).

ДОДАТОК А

**Веб орієнтована інформаційна система обліку комп'ютерного
обладнання**

Текст програми

НТУУ «КПІ ім. Ігоря Сікорського»

Аркушів 13

Київ — 2026

Додаток А

Програмні засоби:

- мова програмування — Python;
- середовище розробки — Visual Studio Code;
- взаємодія з веб-сервісами та REST API — Requests;
- збір детальної інформації про процесор — Py-cpuinfo;
- отримання системних характеристик — Psutil;
- взаємодія з ОС та виконання системних команд — Subprocess;
- ідентифікація мережевих інтерфейсів пристрою — Socket, uuid;
- зовнішнє джерело даних про вразливості — NVD REST API.

agent.py

```
import os
import sys
sys.stdout.reconfigure(encoding='utf-8', errors='replace')
sys.stderr.reconfigure(encoding='utf-8', errors='replace')
from pathlib import Path
from dotenv import load_dotenv
load_dotenv(Path(__file__).resolve().parents[1] / '.env')
import re
import socket
import platform
import subprocess
import time
import uuid
import requests
from cpuinfo import get_cpu_info
import psutil
```

```

API_URL    = os.getenv("API_URL", "http://localhost:5001")
NVD_API    = "https://services.nvd.nist.gov/rest/json/cves/2.0"
SEND_INTERVAL = int(os.getenv("SEND_INTERVAL", "300"))

```

```

CPE_VENDORS = {
    "python": ("python", "python"),
    "java": ("oracle", "jdk"),
    "node": ("nodejs", "node.js"),
    "nodejs": ("nodejs", "node.js"),
    "7zip": ("7-zip", "7-zip"),
    "winrar": ("rarlab", "winrar"),
    "chrome": ("google", "chrome"),
    "firefox": ("mozilla", "firefox"),
    "edge": ("microsoft", "edge_chromium"),
    "openssl": ("openssl", "openssl"),
    "openssh": ("openbsd", "openssh"),
    "zoom": ("zoom", "zoom"),
    "slack": ("slack", "slack"),
    "teams": ("microsoft", "teams"),
    "putty": ("simon_tatham", "putty"),
    "filezilla": ("filezilla_project", "filezilla"),
    "vlc": ("videolan", "vlc_media_player"),
}

```

```

def get_mac():
    raw = uuid.getnode()
    return ':'.join(f'{(raw >> i) & 0xff:02x}' for i in range(0, 48, 8))[:-1]

```

```

def get_motherboard_serial() -> str:

```

```

try:
    if platform.system() == "Windows":
        result = subprocess.run(
            ["powershell", "-NoProfile", "-Command",
             "Get-WmiObject Win32_BaseBoard | Select-Object -ExpandProperty
SerialNumber"],
            capture_output=True, text=True, timeout=10
        )
        serial = result.stdout.strip()
        invalid = {", 'to be filled by o.e.m.', 'default string', 'none', 'n/a', 'not
applicable'}}
        if serial and serial.lower() not in invalid:
            return serial
    elif platform.system() == "Linux":
        result = subprocess.run(
            ["sudo", "dmidecode", "-s", "baseboard-serial-number"],
            capture_output=True, text=True, timeout=10
        )
        serial = result.stdout.strip()
        if serial and serial.lower() not in (", 'none', 'n/a', 'to be filled by o.e.m.'):
            return serial
    except Exception:
        pass
    return get_mac()

def get_local_ip():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.settimeout(2)

```

```

        s.connect(('8.8.8.8', 80))
        ip = s.getsockname()[0]
        s.close()
        return ip
except Exception:
    try:
        return socket.gethostbyname(socket.gethostname())
    except Exception:
        return None

def get_device_info():
    cpu = get_cpu_info()
    return {
        "hostname": platform.node(),
        "os": f"{platform.system()} {platform.release()}",
        "os_version": platform.version(),
        "cpu": cpu.get("brand_raw", "Unknown"),
        "cpu_cores": psutil.cpu_count(logical=False),
        "ram_gb": round(psutil.virtual_memory().total / (1024**3), 2),
        "serial": get_motherboard_serial(),
        "ip": get_local_ip(),
        "architecture": platform.machine(),
    }

def get_installed_software():
    system = platform.system()
    software = []

    if system == "Windows":
        try:

```

```

ps_cmd = (
    "Get-ItemProperty "
    "HKLM:\\Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\*
, "
    "HKLM:\\Software\\WOW6432Node\\Microsoft\\Windows\\CurrentVe
rsion\\Uninstall\\* "
    "| Where-Object { $_.DisplayName } "
    "| Select-Object DisplayName, DisplayVersion "
    "| ConvertTo-Csv -NoTypeInfoInformation"
)
result = subprocess.run(["powershell", "-NoProfile", "-Command",
ps_cmd],
                        capture_output=True, text=True, timeout=60)
for line in result.stdout.strip().splitlines()[1:]:
    line = line.strip().strip('"')
    parts = line.split('" "')
    if len(parts) == 2 and parts[0] and parts[1]:
        software.append({"name": parts[0].strip(), "version": parts[1].strip()})
    print(f'PowerShell: {len(software)} программ')
except Exception as e:
    print(f'PS: {e}')

elif system == "Linux":
    try:
        r = subprocess.run(["dpkg-query", "-W", "-f=${Package} ${Version}\n"],
                        capture_output=True, text=True, timeout=30)
        for line in r.stdout.strip().splitlines():
            p = line.split(" ", 1)
            if len(p) == 2:

```

```

        software.append({"name": p[0], "version": p[1]})
except Exception:
    try:
        r = subprocess.run(["rpm", "-qa", "--queryformat", "%{NAME}
%{VERSION}\n"],
                            capture_output=True, text=True, timeout=30)
        for line in r.stdout.strip().splitlines():
            p = line.split(" ", 1)
            if len(p) == 2:
                software.append({"name": p[0], "version": p[1]})
except Exception as e:
    print(f"Linux: {e}")

return software[:50]

def get_cpe_key(software_name: str) -> str:
    n = software_name.lower()
    cleaned = re.sub(r'[\s\-\_\.]', "", n)
    for key in CPE_VENDORS:
        if key in cleaned:
            return key
    cleaned_nd = re.sub(r'\d+', "", cleaned)
    for key in CPE_VENDORS:
        if key in cleaned_nd:
            return key
    return ""

def extract_version_from_name(name: str) -> str:
    MSI-HOMEP
    matches = re.findall(r'(?:\^|\s)v?(\\d+(?:\\.\\d+){1,3})\\b', name)

```

```

if matches:
    return matches[0]
return ""

def get_real_version(name: str, registry_version: str) -> str:
    name_ver = extract_version_from_name(name)
    if name_ver:
        return name_ver
    nums = re.findall(r'\d+', registry_version)
    return '.'.join(nums[:3]) if nums else registry_version

def normalize_version(version: str, parts: int = 3) -> str:
    nums = re.findall(r'\d+', version)
    return '.'.join(nums[:parts]) if nums else version

def ver_tuple(v: str) -> tuple:
    nums = re.findall(r'\d+', v)
    return tuple(int(x) for x in nums[:4]) if nums else ()

def pad_versions(a: tuple, b: tuple) -> tuple:
    max_len = max(len(a), len(b))
    a_p = a + (0,) * (max_len - len(a))
    b_p = b + (0,) * (max_len - len(b))
    return a_p, b_p

def check_cve(software_name: str, version: str, cpe_key: str) -> list:
    cpe_info = CPE_VENDORS.get(cpe_key)
    if not cpe_info:
        return []
    vendor, product = cpe_info
    ver_norm = normalize_version(version)

```

```

inst_t = ver_tuple(version)

if not inst_t or len(inst_t) < 2:
    return []

cpe_str = f'cpe:2.3:a:{vendor}:{product}:{ver_norm}:*:*:*:*:*:*:*'
try:
    resp = requests.get(NVD_API,
                        params={"cpeName": cpe_str, "resultsPerPage": 100},
                        timeout=15)
    if resp.status_code == 429:
        print(f'NVD rate limit — пропускаемо {software_name}')
        return []
    if resp.status_code != 200:
        return []
    vulns = []
    for item in resp.json().get("vulnerabilities", []):
        cve = item.get("cve", {})
        cve_id = cve.get("id", "")
        year = int(cve_id.split("-")[1]) if "-" in cve_id else 0
        if year < 2010:
            continue
        configs = cve.get("configurations", [])
        if not configs:
            continue
        version_matches = False
        for config in configs:
            if version_matches: break
            for node in config.get("nodes", []):
                if version_matches: break

```

```

for entry in node.get("cpeMatch", []):
    if not entry.get("vulnerable", True):
        continue

    criteria = entry.get("criteria", "").lower()
    c_parts = criteria.split(":")
    if len(c_parts) < 6:
        continue
    if c_parts[3] != vendor or c_parts[4] != product:
        continue
    v_si = entry.get("versionStartIncluding")
    v_se = entry.get("versionStartExcluding")
    v_ei = entry.get("versionEndIncluding")
    v_ee = entry.get("versionEndExcluding")
    c_ver = c_parts[5]
    if v_si or v_se or v_ei or v_ee:
        in_range = True
        if v_si:
            a, b = pad_versions(inst_t, ver_tuple(v_si))
            if a < b: in_range = False
        if v_se:
            a, b = pad_versions(inst_t, ver_tuple(v_se))
            if a <= b: in_range = False
        if v_ei:
            a, b = pad_versions(inst_t, ver_tuple(v_ei))
            if a > b: in_range = False
        if v_ee:
            a, b = pad_versions(inst_t, ver_tuple(v_ee))
            if a >= b: in_range = False

```

```

        if in_range:
            version_matches = True
            break

    elif c_ver and c_ver not in ("*", "-", ""):
        cpe_t = ver_tuple(c_ver)
        if len(cpe_t) >= 2 and inst_t[:len(cpe_t)] == cpe_t:
            version_matches = True
            break

if not version_matches:
    continue

metrics = cve.get("metrics", {})
severity = "UNKNOWN"

if "cvssMetricV31" in metrics:
    severity = metrics["cvssMetricV31"][0]["cvssData"]["baseSeverity"]
elif "cvssMetricV30" in metrics:
    severity = metrics["cvssMetricV30"][0]["cvssData"]["baseSeverity"]
elif "cvssMetricV2" in metrics:
    severity = metrics["cvssMetricV2"][0]["baseSeverity"]

if severity not in ("HIGH", "CRITICAL", "MEDIUM"):
    continue

descriptions = cve.get("descriptions", [])
description = next((d["value"] for d in descriptions if d["lang"] == "en"),
    "")

vulns.append({
    "cve_id":    cve_id,
    "severity":  severity,
    "description": description[:500],
    "software":  f"{software_name.split()[0]} {ver_norm}",

```

```

    })

    return vulns
except Exception as e:
    print(f"CVE: {e}")
    return []

def scan_vulnerabilities(software_list: list) -> list:
    all_vulns = []
    seen = set()
    candidates = []
    for sw in software_list:
        cpe_key = get_cpe_key(sw["name"])
        if not cpe_key or cpe_key in seen:
            continue
        real_ver = get_real_version(sw["name"], sw["version"])
        if not real_ver:
            continue
        seen.add(cpe_key)
        candidates.append({"name": sw["name"], "version": real_ver, "cpe_key":
cpe_key})

    print(f"Перевірка {len(candidates)} відомих програм на CVE...")
    for idx, sw in enumerate(candidates):
        print(f" • Перевіряю {sw['cpe_key']} {sw['version']}...")
        vulns = check_cve(sw["name"], sw["version"], sw["cpe_key"])
        all_vulns.extend(vulns)
    if vulns:
        print(f"    Знайдено {len(vulns)} CVE")
    else:
        print(f"    Вразливостей не знайдено")

```

```

    if idx < len(candidates) - 1:
        time.sleep(6)

    print(f"Всього виявлено: {len(all_vulns)} вразливостей")
    return all_vulns

def register_device(info):
    try:
        resp = requests.post(f"{API_URL}/api/devices", json={
            "inventory_number": f"INV-{get_mac().replace(':', '')}",
            "name": info["hostname"],
            "device_type": "computer",
            "serial_number": info["serial"],
            "specifications": info,
        }, timeout=10)
        device_id = resp.json().get("id")
        print(f"Зареєстровано: ID={device_id}")
        return device_id
    except Exception as e:
        print(f"Реєстрація: {e}")
        return None

def send_vulnerabilities(device_id, vulns):
    try:
        resp = requests.post(f"{API_URL}/api/vulnerabilities",
                               json={"device_id": device_id, "vulnerabilities": vulns},
                               timeout=10)
        print(f"{resp.json().get('message')}")
    except Exception as e:
        print(f"Відправка: {e}")

```

```

def run():
    print("Агент запущено")
    while True:
        print("\n" + "-" * 50)
        print(f'{time.strftime("%H:%M:%S')}")

        info = get_device_info()
        print(f'{info["hostname"]} | {info["os"]} | RAM: {info["ram_gb"]} GB")

        device_id = register_device(info)
        if device_id:
            software = get_installed_software()
            vulns = scan_vulnerabilities(software)
            send_vulnerabilities(device_id, vulns)

        print(f"Наступна перевірка через {SEND_INTERVAL // 60} хв.")
        time.sleep(SEND_INTERVAL)

if __name__ == "__main__":
    run()

```