МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КОГУТ МАКСИМ ЯРОСЛАВОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА СИСТЕМИ ВІДДАЛЕНОГО АДМІНІСТРУВАННЯ КОМП'ЮТЕРІВ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Роман БАБАКОВ, професор кафедри
інформаційних технологій,
д-р техн. наук, доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Когут М. Я. Розробка системи віддаленого адміністрування комп'ютерів. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній роботі розроблено автономний клієнт-серверний комплекс для моніторингу та керування робочими станціями Windows в умовах NAT/Firewall. Сервер побудовано на базі Debian із мікрофреймворком Flask, клієнтські агенти реалізовано мовою Python з використанням системних бібліотек. Безпеку та P2P-зв'язок забезпечено через шифровані тунелі Mesh-мережі Tailscale. Впроваджено безсесійну автентифікацію, приховане перехоплення консольних потоків та RegEx-алгоритм фокусування вікон. Тестування підтвердило високу швидкість і низьке споживання ресурсів (0,2% ЦП у спокої).

ABSTRACT

M. Y. Kogut. Development of a Remote Computer Administration System. Major 122 “Computer Science,” Educational Program “Computer Science.” Vasyl Stus Donetsk National University, Vinnytsia, 2026.

This thesis presents a standalone client-server system designed to monitor and manage Windows workstations in NAT/firewall environments. The server is built on Debian using the Flask microframework, while the client agents are implemented in Python using system libraries. Security and P2P communication are ensured via encrypted tunnels of the Tailscale mesh network. Sessionless authentication, hidden console stream interception, and a RegEx-based window focusing algorithm have been implemented. Testing confirmed high speed and low resource consumption (0.2% CPU at idle).

ЗМІСТ

ВСТУП

Актуальність теми. У сучасну епоху глобальної цифровізації, стрімкого розвитку хмарних технологій та масштабної розбудови корпоративних інформаційно-комунікаційних інфраструктур завдання забезпечення ефективного, безперебійного та оперативного технічного обслуговування парку комп'ютерної техніки набуває критично важливого значення для підприємств будь-якого масштабу. Сучасний етап розвитку суспільства характеризується децентралізацією робочих місць, масовим переходом комерційних і державних організацій на віддалений або гібридний режими роботи, а також географічною розподіленістю філій та окремих співробітників. Це створює принципово нові, складні виклики для системних адміністраторів та інженерів технічної підтримки, перед якими постає завдання цілодобового контролю працездатності десятків або сотень віддалених робочих станцій, серверів та терміналів [1, 2].

Забезпечення належного рівня функціонування комп'ютерних систем потребує безперервного моніторингу їхнього поточного технічного стану у реальному часі, своєчасного виявлення та діагностики програмно-апаратних збоїв, контролю завантаження обчислювальних потужностей, а також можливості оперативного прямого втручання в роботу операційної системи для усунення виниклих неполадок. Використання класичних підходів, які передбачають безпосередню фізичну присутність технічного спеціаліста біля адміністрованого пристрою, у сучасних реаліях є абсолютно неефективним, оскільки призводить до катастрофічних часових витрат кваліфікаційного персоналу, значного збільшення фінансових витрат на логістику та тривалого простою критично важливих бізнес-процесів через затримки в обслуговуванні [3, 4].

Головним ускладнюючим інженерним фактором при спробі організації дистанційного керування є специфіка сучасного побудови мережевої архітектури інтернету. Переважна більшість кінцевих користувачів, працюючи поза межами централізованого офісу, підключається до глобальної мережі через

канали мобільних операторів зв'язку (3G, 4G, 5G LTE) або домашніх провайдерів. Такі постачальники послуг з метою економії адресного простору IPv4 масово використовують технології динамічного розподілу адрес та багаторівневі механізми трансляції мережевих адрес NAT (зокрема, Carrier-Grade NAT). Внаслідок цього робочі станції користувачів опиняються за закритими брандмауерами та отримують так звані сірі IP-адреси, які є абсолютно невидимими і недоступними для прямого підключення ззовні. Спроба ініціювати з'єднання з таким комп'ютером за допомогою класичних протоколів віддаленого робочого стола (як-от RDP чи VNC) зазнає невдачі, якщо на стороні клієнта попередньо не виконано складне, індивідуальне налаштування перенаправлення портів (Port Forwarding), що є неможливим у мобільних мережах або потребує високої кваліфікації від рядового користувача [2, 5, 6].

Існуючі на ринку комерційні програмні комплекси для віддаленого адміністрування (наприклад, TeamViewer, AnyDesk) частково вирішують проблему обходу NAT, проте мають низку суттєвих недоліків, які обмежують їхнє застосування. Більшість таких систем функціонують на базі централізованої закритої інфраструктури сторонніх іноземних компаній. Пропускання конфіденційного корпоративного трафіку та системних команд через проміжні сервери третіх сторін створює серйозні ризики для інформаційної безпеки, компрометації даних та порушення вимог законодавства щодо захисту персональної інформації. Окрім цього, сучасна політика ліцензування комерційного софту передбачає високу регулярну плату, що стає непосильним фінансовим тягарем для підприємств малого бізнесу, стартапів, волонтерських організацій та бюджетних установ. Іншим вагомим мінусом комерційних утиліт є їхня надлишкова функціональність, яка призводить до високого навантаження агентами моніторингу на апаратні ресурси процесора та оперативної пам'яті адміністрованих комп'ютерів, що суттєво знижує продуктивність праці кінцевих користувачів на малопотужних пристроях [7, 8].

Ефективним та перспективним інженерним вирішенням зазначеного комплексу проблем є розробка та впровадження спеціалізованих систем віддаленого адміністрування, побудованих на основі сучасних децентралізованих віртуальних приватних мереж (Mesh-VPN) та кросплатформних веб-технологій. Інтеграція програмного комплексу із технологіями побудови захищених криптографічних тунелів (зокрема Tailscale) дозволяє повністю нівелювати будь-які обмеження, пов'язані з наявністю жорстких брандмауерів, корпоративних фаєрволів та багаторівневих трансляцій адрес NAT. У такій архітектурі агент-клієнт, що працює як ізольована фонова служба на цільовому комп'ютері під керуванням Windows, самостійно ініціює вихідні безпечні POST-запити до центрального веб-сервера. Це забезпечує залізобетонну стабільність та надійність каналу зв'язку незалежно від географічного розташування пристрою та типу його підключення до інтернету [9, 10, 11].

Зазначений підхід дозволяє реалізувати повноцінну концепцію кросплатформного віддаленого керування. Системний адміністратор позбавляється прив'язки до стаціонарного робочого місця і отримує миттєвий доступ до детальної телеметрії заліза, списку активних вікон та інтерактивної консолі керування операційною системою через звичайний веб-інтерфейс браузера мобільного телефона Apple iPhone чи будь-якого іншого мобільного пристрою з будь-якої точки світу. Написання серверної частини на базі операційної системи Debian із використанням мови програмування Python та легковагих асинхронних веб-фреймворків гарантує створення оптимізованого, швидкодійного програмного продукту з високим рівнем безпеки та мінімальним споживанням системних ресурсів [12, 13, 14].

З огляду на це, актуальність розробки системи віддаленого адміністрування комп'ютерів зумовлена об'єктивною потребою сучасної IT-індустрії у створенні безпечного, повністю автономного, економічно вигідного, ресурсоощадного та незалежного від сторонніх закритих сервісів інструменту

для централізованого моніторингу й керування розподіленими комп'ютерними мережами в умовах обмеженого прямого доступу через приватні мережеві канали.

Мета і завдання дослідження. Метою кваліфікаційної роботи є комплексне теоретичне обґрунтування, системне проектування, програмна реалізація та експериментальне дослідження архітектурно оптимізованого клієнт-серверного програмного комплексу віддаленого адміністрування, який забезпечує збір телеметричних даних апаратних ресурсів, інтерактивне керування процесами операційної системи та виконання консольних команд у реальному часі через кросплатформний веб-інтерфейс в умовах обмеженого доступу до обчислювальних вузлів через механізми NAT та брандмауери.

Для успішного досягнення поставленої мети дослідження та розв'язання сформульованої інженерної проблеми необхідно послідовно виконати такі науково-практичні завдання:

1. Провести глибокий системний аналіз предметної області, виконати порівняльне дослідження архітектури та функціональних можливостей існуючих комерційних аналогів систем дистанційного керування, чітко визначити їхні обмеження при роботі в ізольованих мережах та проаналізувати вразливості у сфері конфіденційності даних.

2. Дослідити сучасні мережеві протоколи, технології організації децентралізованих Mesh-VPN та методи побудови захищених тунелів, які дозволяють здійснювати надійний обхід обмежень сірих IP-адрес без необхідності внесення змін до таблиць маршрутизації проміжного мережевого обладнання.

3. Сформулювати вичерпний перелік функціональних, нефункціональних та загальносистемних вимог до створюваного програмного забезпечення з урахуванням суворих критеріїв оптимізації обчислювальних навантажень на цільових робочих станціях.

4. Спроектувати масштабовану клієнт-серверну архітектуру системи та розробити алгоритмічні моделі інформаційного обміну між фоновим агентом і центральним веб-сервером на основі оптимізованих циклічних HTTP-запитів.

5. Розробити та програмно реалізувати алгоритми низькорівневого перехоплення метрик завантаження центрального процесора, оперативної пам'яті та дискової підсистеми, а також механізми динамічного зчитування списку активних вікон і процесів ОС Windows.

6. Спроектувати та впровадити алгоритм безпечного віддаленого виконання консольних команд операційної системи із можливістю перехоплення, обробки та повернення стандартних потоків виведення даних (stdout) та повідомлень про помилки (stderr).

7. Створити програмний комплекс, що складається з оптимізованого клієнтського додатка-агента для ОС Windows та серверної частини на базі веб-фреймворку Flask під керуванням серверної операційної системи Debian.

8. Розробити та інтегрувати надійну систему безсесійної авторизації користувачів на основі статичних унікальних мастер-токенів для забезпечення коректного та стабільного функціонування веб-панелі в обмежених браузерних середовищах мобільних операційних систем (зокрема iOS).

9. Провести комплексну серію експериментальних досліджень та функціонального тестування розробленої системи в реальних умовах функціонування віртуальної мережі Tailscale, оцінити стабільність передачі графічних даних та провести всебічний аналіз ефективності використання системних ресурсів обчислювальних вузлів.

Об'єкт, предмет та методи розробки.

Об'єктом дослідження є технологічний процес забезпечення дистанційного технічного обслуговування, оперативного моніторингу поточного стану апаратних ресурсів, контролю програмного середовища та системного

адміністрування географічно розподілених обчислювальних засобів і робочих станцій у межах сучасних гетерогенних інформаційно-комунікаційних інфраструктур.

Предметом розробки є архітектурні рішення, програмні методи, моделі інформаційної взаємодії, математичні підходи та алгоритми побудови клієнт-серверного програмного комплексу віддаленого керування, що забезпечують збір телеметрії заліза, трансляцію графічних даних екрана та виконання команд операційної системи через веб-інтерфейс в умовах відсутності прямої видимості пристроїв через брандмауери та механізми трансляції адрес NAT.

Методи розробки. Для вирішення поставлених у кваліфікаційній роботі наукових та практичних завдань, а також для забезпечення належної точності, обґрунтованості та об'єктивності отриманих результатів розробки, було застосовано комплексний методологічний апарат, що базується на інтеграції таких методів наукового пізнання:

- *Методи системного аналізу, декомпозиції та порівняльного огляду* — активно застосовувалися на початкових етапах дослідження предметної області під час вивчення архітектури існуючих комерційних засобів віддаленого доступу, аналізу їхніх вразливостей, а також при теоретичному дослідженні мережевих технологій обходу NAT та побудови VPN-тунелів.
- *Методи об'єктно-орієнтованого аналізу, проектування та UML-моделювання* — використовувалися для детального розробленої структури клієнт-серверної взаємодії, формування логіки побудови модулів системи, розробки діаграм прецедентів, послідовностей та компонентів програмного комплексу.
- *Методи теорії алгоритмів та регулярних виразів* — були задіяні при проектуванні та програмній реалізації логіки роботи клієнтського агента, зокрема для розробки алгоритмів фільтрації консольних команд, динамічного перехоплення потоків даних та ідентифікації цільових вікон операційної системи Windows для їхнього фокусування.

- *Методи мережевого програмування, веб-інженерії та API-проектування* — застосовувалися для організації надійного, захищеного та оптимізованого каналу передачі даних між фоновим клієнтом та веб-сервером Flask, а також для проектування інтерфейсу користувача та побудови безсесійного механізму автентифікації.
- *Методи експериментального дослідження, функціонального тестування та комп'ютерного моніторингу* — використовувалися на завершальних етапах роботи під час розгортання сервера на базі Debian, запуску клієнтської частини, проведення натурних випробувань системи у віртуальній мережі Tailscale через мобільний інтернет, а також для точного статистичного вимірювання навантаження софту на центральний процесор та оперативну пам'ять комп'ютера.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНЕ СИСТЕМНЕ АДМІНІСТРУВАННЯ

1.1. Дослідження існуючих аналогів систем віддаленого адміністрування та їхніх обмежень у приватних мережах

Проектування програмного забезпечення для дистанційного керування потребує аналізу існуючих технологічних рішень для виявлення ключових інженерних проблем та формування базису для розробки власного продукту [1, 2, 7].

Найбільш репрезентативними аналогами на ринку засобів віддаленого доступу є: [7, 8]

Remote Desktop Protocol (RDP) від Microsoft — пропрієтарний протокол прикладного рівня, інтегрований в операційні системи Windows NT.

TeamViewer — комерційна хмарна платформа для дистанційного керування робочими станціями без прив'язки до локальної інфраструктури.

AnyDesk — пропрієтарне рішення на базі кодека DeskRT, оптимізованого для передачі графічного інтерфейсу через канали з низькою пропускною здатністю.

Протокол RDP. Технологія побудована за архітектурою клієнт-сервер і приймає підключення на TCP-порт 3389. Перевагою є глибока інтеграція з підсистемами ОС, що забезпечує трансляцію графічних примітивів замість піксельного потоку [15, 16]. Проте протокол має критичні обмеження:

— Проблема NAT: цільовий комп'ютер потребує білої IP-адреси, а обхід вимагає складних VPN або ручного налаштування переадресації портів [5, 17].

— Ризики безпеки: відкритий порт 3389 є постійним об'єктом brute-force атак та експлуатації вразливостей автентифікації Windows [18, 19].

— *Обмеження мобільного керування:* протокол оптимізований під мишу та клавіатуру, що унеможливлює комфортну роботу через сенсорний екран смартфона.

Платформи TeamViewer та AnyDesk використовують архітектуру з проміжними Relay-серверами в хмарі розробника [4, 7]. Агент самостійно відкриває вихідну TCP-сесію через порти 80/443, після чого сервер ретранслює трафік між пристроями [8, 20]. Критичні недоліки:

— *Безпека:* весь трафік, включаючи команди та знімки екрана, проходить через інфраструктуру третіх сторін без можливості перевірки відсутності backdoor-доступу.

— *Ліцензування:* алгоритми виявляють комерційне використання і блокують сесії без придбання підписки.

— *Навантаження на ресурси:* надлишкові модулі (голосовий зв'язок, друк, конференції) постійно споживають пам'ять і знижують продуктивність на слабких машинах.

— *Нестабільність на iOS:* хмарна сесія прив'язана до ідентифікатора підключення і розривається при найменшому коливанні мобільного сигналу.

Обмеження у Mesh-мережах. В інфраструктурі Tailscale/WireGuard пристрої отримують стабільні внутрішні адреси 100.x.x.x і взаємодіють напряму через p2p-тунелі [9, 10]. Використання хмарних Relay-серверів TeamViewer чи AnyDesk у такому середовищі є надлишковим — трафік здійснює зайві переходи через зовнішній інтернет, збільшуючи затримки [10, 11]. RDP всередині Tailscale частково вирішує проблему безпеки портів, однак не забезпечує кросплатформного доступу через веб-браузер і повністю блокує сесію локального користувача Windows.

Для наочного порівняння характеристик розглянутих систем сформуємо зведену таблицю їхніх параметрів.

Таблиця 1.1. Порівняльний аналіз характеристик систем віддаленого адміністрування

Критерій порівняння	**Microsoft RDP**	**TeamViewer / AnyDesk**	**Проєктована клієнт-серверна система**
Архітектурний тип	Клієнт-сервер (вхідні запити)	З проміжним Relay-сервером сторонньої компанії	Клієнт-сервер (вихідні запити агента на власний сервер)
Залежність від NAT та сірих IP	Критична (потребує білої IP або прокидання портів)	Відсутня (трафік ретранслюється через хмару розробника)	Відсутня (прямий обмін всередині захищеної мережі Tailscale)
Контроль над сервером та безпека даних	Повний (локальний контроль інфраструктури)	Нульовий (дані проходять через закриті сторонні сервери)	Повний (сервер розгорнуто на власному Debian-сервері)
Вплив на локальну сесію користувача	Блокує екран поточного користувача Windows	Спільне використання або перехоплення сесії	Повністю прихований фоновий моніторинг без блокування
Споживання системних ресурсів (ОЗУ/ЦП)	Низьке (інтегровано в ядро ОС)	Високе (через надлишкову додаткову функціональність)	Мінімальне (оптимізований скрипт на Python)
Доступність інтерфейсу з мобільних пристроїв	Тільки через спеціалізований важкий додаток	Тільки через пропрієтарний мобільний клієнт	Кросплатформний доступ через будь-який веб-браузер
Фінансова вартість експлуатації	Безкоштовно (включено в ліцензію Windows Pro)	Висока регулярна плата за комерційні лісензії	Абсолютно безкоштовно (відкритий власний код)

Таким чином, проведений системний аналіз існуючих аналогів доводить необхідність розробки спеціалізованої системи віддаленого адміністрування

комп'ютерів. Створюваний програмний комплекс має поєднувати в собі низьке споживання ресурсів, притаманне вбудованим утилітам, з гнучкістю обходу брандмауерів хмарних платформ, функціонуючи при цьому виключно на власних серверних потужностях Debian всередині приватної мережі Tailscale з наданням доступу через універсальний веб-інтерфейс [1, 2, 7].

1.2. Порівняльний аналіз протоколів та методів обходу NAT (STUN, TURN, VPN, Mesh-мережі)

Для побудови надійної архітектури системи віддаленого адміністрування, яка здатна функціонувати в будь-яких мережевих умовах, необхідно виконати глибокий аналіз існуючих методів подолання обмежень трансляції мережевих адрес NAT. Оскільки кінцеві пристрої часто знаходяться за закритими брандмауерами або в мережах мобільних операторів, вибір технології обходу NAT безпосередньо впливає на стабільність, швидкість передачі графічних даних та загальну безпеку програмного комплексу [1, 2, 20].

У сучасній мережевій інженерії для вирішення проблеми встановлення зв'язку між вузлами, що знаходяться за NAT, використовуються кілька фундаментальних підходів та протоколів. Кожен із них має власну логіку роботи, специфіку маршрутизації трафіку та інфраструктурні обмеження.

Протокол STUN (Session Traversal Utilities for NAT)

Протокол STUN є легковажним мережевим інструментом, який дозволяє клієнту, що знаходиться за NAT, визначити свою публічну IP-адресу, тип застосованого NAT на проміжному маршрутизаторі, а також зовнішній порт, який шлюз виділив для конкретного внутрішнього UDP-потоку [6, 21].

Принцип роботи STUN базується на тому, що клієнт відправляє запит на зовнішній STUN-сервер із відомою публічною адресою. Сервер зчитує IP-адресу та порт відправника з отриманого пакета і відправляє ці дані назад клієнту. Дізнавшись свої публічні реквізити, клієнт може передати їх іншому пристрою для встановлення прямого з'єднання (Peer-to-Peer) [21].

Обмеження STUN у приватних мережах:

- Неефективність із симетричним NAT (Symmetric NAT): Це головний недолік протоколу. Більшість мобільних операторів зв'язку та корпоративних брандмауерів використовують саме симетричний тип NAT. При його застосуванні маршрутизатор виділяє новий зовнішній порт для кожного нового напрямку з'єднання. Публічний порт, який клієнт дізнався через STUN-сервер, буде абсолютно іншим при спробі підключитися до цільового адміністрованого комп'ютера, що робить встановлення прямого p2p-зв'язку неможливим [5, 6].
- **Орієнтація на UDP:** Протокол оптимізований для роботи з UDP-трафіком, що не завжди підходить для систем керування, де потрібна залізобетонна гарантія доставки пакетів (TCP) для передачі консольних команд.

Протокол TURN (Traversal Using Relays around NAT) [22]

Протокол TURN є розширенням концепції STUN і використовується як резервний залізобетонний метод встановлення зв'язку, коли пряме p2p-з'єднання неможливо реалізувати через наявність жорстких симетричних брандмауерів на обох кінцях каналу [6, 22].

Логіка TURN полягає в залученні виділеного проміжного сервера ретрансляції. Клієнт підключається до TURN-сервера і отримує на ньому тимчасову адресу. Другий пристрій також підключається до цього сервера. Після цього TURN-сервер починає приймати пакети від одного учасника і пересилати їх іншому, виступаючи в ролі повноцінного медіатора трафіку.

Обмеження TURN у приватних мережах:

Високе навантаження на сервер: Весь мережевий трафік, включаючи важкі графічні потоки (скріншоти екрана), проходить безпосередньо через проміжний вузол. Це потребує великої пропускної здатності каналу сервера та призводить до значних фінансових витрат на утримання інфраструктури. У разі одночасної роботи великої кількості користувачів навантаження на сервер

зростає пропорційно обсягу переданих даних. Крім того, виникає необхідність використання потужнішого серверного обладнання та каналів зв'язку з високою пропускною здатністю, що збільшує експлуатаційні витрати та ускладнює масштабування системи.

Збільшення затримок (Latency): Трафік здійснює додаткові переходи (hops) через інтернет до TURN-сервера, що суттєво уповільнює відгук інтерактивної консолі та оновлення графіків телеметрії на екрані телефона адміністратора. Оскільки дані не передаються безпосередньо між кінцевими пристроями, а проходять через проміжний вузол, збільшується час доставки пакетів та ймовірність виникнення затримок у разі перевантаження мережі або самого сервера. Це особливо помітно під час віддаленого керування системами в режимі реального часу, коли швидкість реакції має критичне значення.

Класичні VPN-технології (Virtual Private Network)

Традиційні системи VPN (на базі OpenVPN, IPsec або L2TP) використовують топологію «зірка» (Hub-and-Spoke). У такій архітектурі в мережі розгортається один центральний VPN-сервер із виділеною білою IP-адресою, а всі інші клієнти підключаються до нього, створюючи захищені віртуальні тунелі. Клієнти всередині такої мережі отримують внутрішні IP-адреси і можуть взаємодіяти між собою, оскільки сервер бере на себе функцію маршрутизації всього внутрішнього трафіку [17, 20].

VPN-технології забезпечують високий рівень захисту даних шляхом використання сучасних алгоритмів шифрування та механізмів автентифікації. Вони широко застосовуються для організації захищеного доступу до корпоративних мереж, об'єднання віддалених офісів та забезпечення безпечної роботи співробітників через інтернет. Проте архітектурні особливості класичних VPN-рішень створюють низку обмежень при побудові масштабованих систем віддаленого адміністрування.

Обмеження класичних VPN у приватних мережах:

Централізація трафіку: Як і у випадку з TURN, весь інформаційний обмін між адміністратором та адмініструванним комп'ютером примусово проходить через центральний VPN-сервер. Якщо адміністратор знаходиться поруч із комп'ютером, але обидва підключені до віддаленого сервера, трафік все одно піде через зовнішній інтернет-канал сервера, створюючи надлишкове навантаження. Такий підхід збільшує використання мережевих ресурсів, створює додаткові точки відмови та може призводити до погіршення продуктивності системи при збільшенні кількості активних підключень.

Складність конфігурації: Створення, випуск та керування сертифікатами безпеки (SSL/TLS) для кожного нового клієнта потребує значних трудовитрат та автоматизації, що ускладнює масштабування системи. Адміністратору необхідно підтримувати інфраструктуру сертифікації, контролювати терміни дії сертифікатів та виконувати їх оновлення. У великих мережах це збільшує навантаження на персонал і підвищує ризик помилок під час налаштування або обслуговування системи.

Сучасні Mesh-мережі (на прикладі Tailscale/WireGuard)

Технологія Mesh-VPN представляє собою децентралізовану архітектуру побудови віртуальних приватних мереж. Яскравим представником цього підходу є система Tailscale, яка використовує сучасний, швидкий та безпечний протокол шифрування WireGuard [9, 10].

На відміну від класичних VPN-рішень, Mesh-мережі не вимагають постійного проходження трафіку через центральний вузол. Кожен пристрій виступає рівноправним учасником мережі та може встановлювати прямі захищені з'єднання з іншими вузлами. Такий підхід забезпечує вищу продуктивність, менші затримки та кращу масштабованість системи без необхідності постійного нарощування серверних ресурсів.

У Mesh-мережі замість маршрутизації всього трафіку через один центральний сервер використовується інтелектуальна система координації

(Coordination Server). Координаційний сервер допомагає пристроям обмінятися публічними ключами шифрування та метаданими, а також використовує вбудовані алгоритми STUN/ICE для пробиття NAT. Після того як пристрої знайшли один одного, вони встановлюють пряме зашифроване p2p-з'єднання (Peer-to-Peer) безпосередньо між собою, пускаючи трафік найкоротшим маршрутом [9, 10].

Завдяки такому механізму основне навантаження переноситься з центральної інфраструктури безпосередньо на кінцеві вузли мережі. Це дозволяє значно зменшити витрати на утримання серверів, а також забезпечує більш стабільну роботу системи навіть при значному збільшенні кількості підключених пристроїв.

Якщо брандмауери занадто жорсткі (симетричний NAT з обох сторін), Tailscale автоматично перемикається на використання власних легковажних серверів ретрансляції DERP (аналог TURN), що гарантує 100% доступність пристрою за будь-яких умов [10]. При цьому DERP використовується лише як резервний механізм у випадках, коли встановлення прямого p2p-з'єднання є неможливим. За статистикою розробників, більшість підключень здійснюється напряму, що дозволяє зберігати високу швидкість передачі даних та мінімізувати затримки під час роботи користувачів із віддаленими пристроями.

Переваги використання Mesh-мереж для проєктуємої системи:

- **Прямий зв'язок із мінімальними затримками:** Трафік моніторингу та скріншотів летить напряму від ноутбука до сервера або телефона адміністратора, не навантажуючи сторонні сервіси та забезпечуючи максимальну швидкість відгуку інтерфейсу.
- **Стабільні внутрішні адреси:** Кожен пристрій у Mesh-тунелі отримує постійну внутрішню IP-адресу (наприклад, із простору 100.x.x.x). Це дозволяє зафіксувати адресу Debian-сервера в коді клієнтського агента

ноутбука, повністю нівелюючи проблему динамічних IP-адрес мобільних операторів.

- Безпека з коробки: Весь трафік всередині Mesh-мережі повністю зашифрований на рівні протоколу WireGuard за допомогою сучасних криптографічних примітивів, що дозволяє відмовитися від налаштування складних SSL-сертифікатів на самому веб-сервері Flask [10, 11].

Для узагальнення технічних характеристик розглянутих методів обходу NAT сформуємо порівняльну таблицю.

Таблиця 1.2. Порівняльний аналіз методів обходу NAT та організації зв'язку

Параметр порівняння	**Протокол STUN**	**Протокол TURN**	**Класичний VPN (OpenVPN)**	**Сучасні Mesh-мережі (Tailscale)**
Принцип роботи	Визначення публічних реквізитів для p2p	Ретрансляція трафіку через проміжний сервер	Централізований сервер-шлюз для всіх клієнтів	Децентралізована мережа прямого зв'язку p2p
Ефективність із симетричним NAT	Нульова (повністю блокується)	Залізобетонна (100% пробиття через реле)	Висока (потребує білої IP на сервері)	Залізобетонна (автоперемикання p2p / DERP)
Маршрутизація трафіку	Пряма (Peer-to-Peer)	Через сторонній сервер ретрансляції	Примусово через центральний VPN-сервер	Пряма (найкоротший маршрут між вузлами)
Рівень затримок (Latency)	Мінімальний	Високий (залежить від сервера ретрансляції)	Середній/Високий (залежить від локації сервера)	Мінімальний (завдяки прямим p2p-каналам)
Навантаження на мережеву інфраструктуру	Відсутнє після встановлення сесії	Критично високе (сервер прокачує весь трафік)	Високе (сервер обробляє трафік усіх пристроїв)	Мінімальне (сервер виконує лише координацію ключами)
Криптографічний захист даних	Відсутній (визначає лише координати)	Залежить від налаштувань (вбудоване TLS)	Високий (шифрування OpenSSL, сертифікати)	Максимальний (вбудоване WireGuard-шифрування)

Таким чином, проведений порівняльний аналіз доводить, що інтеграція системи віддаленого адміністрування комп'ютерів із технологією Mesh-мереж є найбільш раціональним та інженерно обґрунтованим рішенням. Це дозволяє отримати стабільні приватні IP-адреси для серверної частини на Debian, повністю обійти обмеження сірих адрес мобільних операторів та забезпечити

безпечну передачу телеметрії без проміжних затримок трафіку на сторонніх комерційних серверах [5, 9, 10].

1.3. Визначення функціональних, нефункціональних та системних вимог до архітектури системи

На основі аналізу предметної області та виявлених недоліків комерційних аналогів сформовано три групи вимог до проєктованої системи віддаленого адміністрування [23, 24].

Функціональні вимоги визначають конкретні дії, які система виконує у відповідь на запити користувача. Функціонал поділено на два рівні: сервіси агента та можливості веб-інтерфейсу [23].

Вимоги до клієнтського агента (agent_client.py):

— безперервний фоновий збір телеметрії ЦП, ОЗУ та дискової підсистеми;

— циклічне зчитування списку активних вікон і процесів Windows;

— виконання консольних команд (cmd/PowerShell) з перехопленням stdout/stderr;

— створення та передача скріншотів екрана за запитом сервера;

— самостійна ініціація вихідних POST-запитів для роботи з-за NAT.

Вимоги до веб-інтерфейсу сервера (server.py):

— динамічна візуалізація телеметрії на дашборді;

— керування вікнами ОС із примусовим фокусуванням на віддаленому ПК;

— інтерактивний термінал із виведенням результатів команд;

— авторизація за унікальним токеном доступу.

Нефункціональні вимоги описують критерії якості системи [23, 24].

— *Ресурсоощадність:* навантаження агента на ЦП у режимі очікування не перевищує 1–2%.

— *Зручність:* адаптивний веб-інтерфейс, оптимізований для керування з сенсорного екрана смартфона.

— *Надійність:* агент стійко переносить розриви зв'язку та циклічно відновлює підключення.

— *Автономність:* система повністю незалежна від сторонніх хмарних сервісів.

— *Затримки:* час від відправки команди до початку виконання не перевищує 2 секунд.

Системні вимоги описують сумісність компонентів із апаратним та програмним середовищем [24, 25].

Серверна частина (Debian): Linux Debian 11/12, Python 3.9+, Flask, активний демон Tailscale, мінімум 512 МБ ОЗУ та 2 ГБ на накопичувачі.

Клієнтська частина (Windows): Windows 10/11 x64, Python 3.10+, бібліотеки psutil, pywinauto та requests, доступ до інтернету та Tailscale.

Пристрій адміністратора: будь-який смартфон із сучасним браузером (Safari, Chrome, Firefox) та активним підключенням до Tailscale-мережі.

1.4. Обґрунтування вибору технологічного стеку

Успішна програмна реалізація розробленої архітектури та забезпечення її відповідності усім сформульованим функціональним і нефункціональним вимогам безпосередньо залежить від раціонального вибору інструментальних засобів, базової операційної системи, мов програмування та сторонніх бібліотек. Головними інженерними критеріями при виборі технологічного стеку для системи віддаленого адміністрування стали: висока швидкість обробки мережевих запитів, мінімальне споживання ресурсів процесора та оперативної пам'яті на адміністрованих робочих станціях, кросплатформна доступність інтерфейсу та високий рівень безпеки інформаційного обміну [12, 26].

Обґрунтування вибору мови програмування Python

Як основне інструментальне середовище для написання серверного та клієнтського компонентів системи було обрано високорівневу мову

програмування Python. Даний вибір зумовлений кількома важливими архітектурними та прагматичними чинниками: [12, 26]

- **Кросплатформність:** Python є інтерпретованою мовою, яка має розвинені середовища виконання як для сімейства операційних систем Windows (де функціонує клієнтський агент), так і для Linux-похідних платформ (на базі яких розгортається веб-сервер). Це дозволяє використовувати однакові структури даних та спрощує логіку взаємодії між модулями.
- **Багата екосистема бібліотек:** Наявність величезної кількості протестованих утиліт і модулів для роботи з низькорівневим API операційних систем, мережевими сокетами та HTTP-протоколами дозволяє суттєво скоротити час на розробку системи, фокусуючись на безпосередній інженерній логіці проєкту.
- **Легковажність та швидкодія:** Скрипти, написані на Python, мають оптимізовані механізми керування пам'яттю, що повністю задовольняє нефункціональну вимогу щодо створення ресурсоощадного клієнтського агента.

Обґрунтування вибору мікрофреймворку Flask

Для реалізації серверної частини та побудови центрального API інформаційного обміну було обрано веб-фреймворк Flask. На відміну від важковагових альтернатив (як-от Django), Flask є класичним мікрофреймворком, який надає лише базовий інструментарій для маршрутизації HTTP-запитів та обробки шаблонів сторінок [13, 14, 27].

Вибір Flask обґрунтований такими перевагами:

1. **Відсутність надлишкового функціоналу:** Фреймворк не містить інтегрованих за замовчуванням важких об'єктно-реляційних маперів (ORM) та складних адміністративних панелей, що забезпечує максимальну швидкість обробки вхідних POST-запитів телеметрії від агента та мінімізує затримки.

2. **Гнучкість налаштування:** Дозволяє розробнику самостійно проєктувати структуру API та інтегрувати специфічні безсесійні механізми авторизації за токенами, що є критично важливим для стабільної роботи веб-інтерфейсу в обмежених мобільних браузерах.

3. **Низькі вимоги до заліза:** Серверний додаток на Flask здатний безперебійно функціонувати навіть на мінімальних обчислювальних потужностях.

Обґрунтування вибору серверної операційної системи Debian

Для розгортання та забезпечення цілодобового функціонування серверного компонента системи було обрано операційну систему Linux дистрибутиву Debian. У сфері системного адміністрування та серверної інженерії Debian вважається одним із найбільш надійних та стабільних рішень [28, 29, 30].

Ключовими факторами вибору цієї ОС є:

- **Залізобетонна стабільність:** Консервативний підхід до тестування пакетів у стабільних релізах Debian гарантує відсутність раптових збоїв та сумісність системних бібліотек протягом усього життєвого циклу експлуатації сервера.
- **Ефективне керування ресурсами:** Операційна система Debian не містить надлишкових графічних оболонок у серверній конфігурації, завдяки чому споживає мінімальну кількість оперативної пам'яті, залишаючи всі апаратні ресурси для обробки мережевого трафіку системи адміністрування.
- **Високий рівень безпеки:** Розвинена система розмежування прав доступу користувачів, вбудовані підсистеми фільтрації пакетів та швидке випущення патчів безпеки дозволяють надійно захистити сервер від зовнішніх загроз.

Обґрунтування вибору спеціалізованих утиліт та бібліотек

Для розширення базових можливостей мови програмування та реалізації специфічних функцій системи до стеку технологій було залучено кілька спеціалізованих інструментів: [31, 32]

1. Бібліотека pywinauto: Цей інструмент обрано як основний засіб для взаємодії з графічною підсистемою ОС Windows на стороні клієнта. Вона дозволяє на низькому рівні отримувати дескриптори (HWND) активних вікон, зчитувати їхні заголовки та виконувати програмне фокусування чи виведення на передній план елементів інтерфейсу без фізичного втручання користувача [15, 31].

2. Бібліотека psutil: Використовується на стороні клієнтського агента для кросплатформного зчитування телеметрії обчислювальних ресурсів. Вона надає прямий доступ до системних лічильників навантаження центрального процесора, утилізації оперативної пам'яті та заповнення дискових накопичувачів [3, 25].

3. Технологія Mesh-мереж (Tailscale): Інтегрована в загальний стек для організації захищеного каналу обміну даними та обходу трансляції адрес NAT. Використання цього інструменту дозволяє повністю відмовитися від налаштування складних SSL/TLS сертифікатів на рівні самого Flask-сервера, оскільки весь трафік автоматично шифрується на рівні віртуального тунелю за допомогою протоколу WireGuard [9, 10, 11].

Організація асинхронного шлюзу мобільних push-сповіщень

Окремим архітектурним компонентом системи є спеціалізований сервіс асинхронного інформування адміністратора про критичні події в інфраструктурі (наприклад, запуск або зупинка сервера, вихід показників заліза за критичні межі). Для реалізації цього функціоналу до стеку було включено інструмент організації HTTP-взаємодії з віддаленими шлюзами сповіщень. Його використання базується на надсиланні легковажних асинхронних запитів у фоновому режимі, що дозволяє миттєво доставляти push-повідомлення на

мобільний пристрій адміністратора, не блокуючи при цьому виконання основних потоків серверної частини.

Таким чином, сформований технологічний стек є збалансованим, інженерно обґрунтованим та оптимованим програмним рішенням. Кожен його компонент чітко виконує поставлену задачу, забезпечуючи високу стабільність, безпеку та ресурсоощадність створеної системи віддаленого адміністрування комп'ютерів.

РОЗДІЛ 2. МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1. Моделювання структури клієнт-серверної взаємодії за допомогою діаграм UML

Проєктування розподілених програмних комплексів потребує створення абстрактних моделей для візуалізації архітектури та логіки взаємодії компонентів. Таким інструментом є уніфікована мова моделювання UML. Для розроблюваної системи побудовано три типи діаграм: прецедентів, послідовності та компонентів [20, 24].

Діаграма прецедентів визначає функціональні межі системи з точки зору двох акторів:

— *Системний адміністратор* — взаємодіє з веб-інтерфейсом через браузер смартфона або ПК. Основні прецеденти: авторизація за токеном, перегляд телеметрії, перегляд списку вікон і процесів, виконання консольних команд.

— *Клієнтський агент (agent_client.py)* — автоматизований актор у фоновому режимі на адміністрованій станції. Основні прецеденти: збір апаратних метрик, передача телеметрії на сервер. Між прецедентами «Виконання консольних команд» та «Запит знімка екрана» реалізовано зв'язок типу include, оскільки перевірка стану робочого стола є типовою частиною дистанційного обслуговування.

Діаграма послідовності зображена на рис. 2.1 описує логіку інформаційного обміну між трьома об'єктами всередині захищеної Mesh-мережі у часовому вимірі: «Мобільний браузер адміністратора», «Веб-сервер Flask» та «Фоновий клієнт-агент».

Алгоритм взаємодії складається з кількох послідовних ітерацій:

1. Адміністратор відкриває веб-інтерфейс і надсилає запит на авторизацію. Сервер Flask перевіряє валідність мастер-токена і повертає динамічну сторінку дашборду.

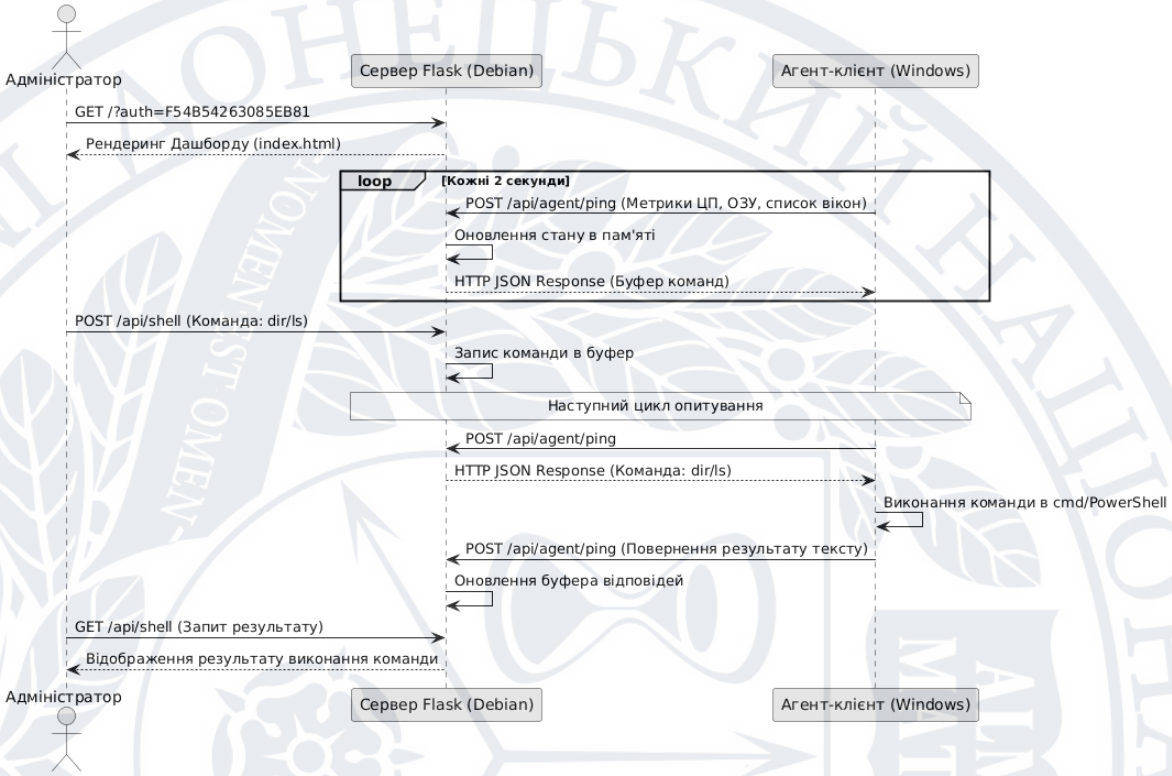

Рисунок 2.1 — Діаграма послідовності інформаційного обміну компонентами системи

2. Фоновий агент з періодичністю у 2 секунди самостійно генерує вихідний POST-запит на API-маршрут сервера /api/agent/ping, передаючи у форматі JSON-пакета актуальні метрики завантаження процесора, пам'яті та список дескрипторів вікон.

3. Сервер Flask приймає пакет, оновлює глобальний об'єкт стану системи у пам'яті сервера і повертає агенту у відповідь (HTTP JSON Response) поточний буфер команд, який містить накопичені завдання від адміністратора.

4. Адміністратор через веб-інтерфейс вводить консольну команду (наприклад, ls чи dir) і натискає кнопку відправки. Запит фіксується в буфері сервера.

5. Під час наступного циклічного підключення агент зчитує команду з буфера, передає її на виконання у дочірній процес командного рядка ОС Windows, перехоплює потік результату і під час наступного пінг-запиту повертає текстову відповідь на сервер для її відображення на екрані адміністратора.

Діаграма компонентів (Component Diagram)

Діаграма компонентів описує фізичну структуру програмного комплексу, визначає модулі коду та зв'язки між ними через програмні інтерфейси (API). Архітектура системи розділена на три програмні блоки, які функціонують у різних операційних середовищах.

Перший блок — серверний компонент подвійного призначення під керуванням ОС Debian. Його основою є модуль server.py, який ділиться на підсистему маршрутизації HTTP-запитів, інтерфейс взаємодії з мобільними пристроями та шлюз обробки звітів агента. Сервер взаємодіє з компонентом «Асинхронний сервіс push-сповіщень» для передачі критичних сигналів через захищене HTTP API.

Другий блок — клієнтський компонент під керуванням ОС Windows. Він представлений модулем agent_client.py, який логічно розділений на три внутрішні сервіси: модуль низькорівневого моніторингу заліза (взаємодіє із системними утилітами через бібліотеку psutil), модуль керування вікнами (використовує функції бібліотеки pywinauto) та модуль мережевого обміну (реалізований за допомогою бібліотеки requests).

Третій блок — інтерфейсний компонент мобільного пристрою адміністратора. Він складається з шаблонів index.html та login.html, які використовують стандартний набір стилів CSS та обмінюються даними з сервером через асинхронні запити. Інформаційний зв'язок між усіма компонентами залізобетонно ізольований всередині криптографічного тунелю віртуальної Mesh-мережі, що дозволяє компонентам взаємодіяти за прямими

внутрішніми IP-адресами без використання зовнішніх проміжних серверів ретрансляції.

Побудовані UML-моделі повністю описують математичну та алгоритмічну логіку функціонування системи віддаленого адміністрування комп'ютерів, що дозволяє перейти до детального математичного опису процесів збору метрик та реалізації алгоритмів обробки системних потоків даних [8, 33].

2.2. Алгоритм циклічного фонового збору телеметрії та взаємодії агента з API сервера

Забезпечення безперервного та ресурсоощадного моніторингу віддалених робочих станцій базується на чіткій організації взаємодії між клієнтським агентом та центральним веб-сервером. Оскільки кінцеві пристрої функціонують в умовах обмеженого доступу з-за брандмауерів та трансляції адрес NAT, ініціатором інформаційного обміну завжди виступає фоновий клієнтський додаток [8, 33].

Логіка роботи клієнтського компонента побудована на базі безкінечного циклу опитування (Polling Loop), який виконується в окремому фоновому потоці операційної системи Windows. Алгоритм функціонування даного модуля складається з кількох послідовних етапів, що повторюються з фіксованою періодичністю у 2 секунди [12, 26].

На першому етапі циклу агент ініціює збір актуальних телеметричних показників апаратної підсистеми комп'ютера. За допомогою системних викликів бібліотеки psutil виконується зчитування таких параметрів: [3, 25]

- Загальний коефіцієнт завантаження центрального процесора (у відсотках).
- Обсяг загальної, зайнятої та вільної оперативної пам'яті (у гігабайтах).
- Рівень утилізації та доступного простору на системних дискових накопичувачах.

Паралельно з цим, використовуючи низькорівневі дескриптори графічної підсистеми Windows, агент формує масив даних про поточне програмне оточення. Сюди входить зчитування списку ідентифікаторів активних процесів та текстових заголовків усіх відкритих вікон операційної системи.

На другому етапі отримані структуровані дані агрегуються та пакуються в єдиний об'єкт передачі даних типу ключ-значення, який серіалізується у текстовий формат JSON. Для оптимізації мережевого трафіку всі числові показники заліза приводяться до єдиного типу даних з плаваючою крапкою з округленням до сотих часток.

На третьому етапі модуль мережевого обміну агента формує вихідний HTTP POST-запит до сервера Flask за стабільною внутрішньою адресою приватної Mesh-мережі Tailscale на маршрут /api/agent/ping. У тіло запиту прикріплюється згенерований JSON-пакет телеметрії. Оскільки з'єднання є вихідним, проміжні маршрутизатори та брандмауери вільно пропускають пакети в інтернет, автоматично створюючи тимчасовий зворотний канал зв'язку для відповіді [13, 14].

Веб-сервер Flask приймає POST-запит, десеріалізує вміст пакета та миттєво оновлює глобальний об'єкт стану віддаленого комп'ютера в оперативній пам'яті сервера Debian. Одразу після цього сервер формує HTTP-відповідь (JSON Response), в яку вкладає вміст буфера команд. Буфер містить інструкції та завдання, які адміністратор міг ввести через мобільний веб-інтерфейс з моменту попереднього підключення агента.

На завершальному етапі циклу клієнтський агент приймає мережеву відповідь від сервера, аналізує наявність нових інструкцій у JSON-структурі, очищає тимчасові змінні та засинає за допомогою системного таймера на 2 секунди, після чого алгоритм повторюється з початкової стадії. Така архітектурна схема забезпечує залізобетонну стабільність зв'язку при використанні динамічних IP-адрес мобільних операторів та мінімальне навантаження на обчислювальні вузли.

2.3. Алгоритми безпечного виконання консольних команд та перехоплення потоків введення-виведення (stdout/stderr)

Дистанційне керування робочою станцією потребує механізму виконання системних інструкцій Windows через ізольовані дочірні процеси з повним перехопленням результатів [16, 26].

Коли агент виявляє команду в отриманому JSON-пакеті, він запускає її через інтерпретатор cmd.exe або PowerShell у прихованому режимі без залучення графічної оболонки [12, 32]. Алгоритм перехоплення базується на перенаправленні двох системних потоків:

— *stdout* — результати успішного виконання інструкцій;

— *stderr* — повідомлення про помилки синтаксису або відсутність прав доступу.

Дескриптори обох потоків перекриваються механізмом PIPE і спрямовуються у внутрішні змінні програми, що повністю приховує консольні вікна від локального користувача [26, 31].

Після виконання команди агент зчитує буфери та виконує каскадне декодування з регіональних кодувань Windows (CP866/CP1251) у формат UTF-8, якого очікує сервер Flask. У разі помилок декодування застосовується режим ігнорування пошкоджених символів [15, 16].

Об'єднані дані stdout та stderr передаються на сервер під час наступного POST-запиту, що гарантує доставку повного результату адміністратору.

2.4. Алгоритм ідентифікації та фокусування вікон операційної системи за допомогою регулярних виразів

Завдання віддаленого керування інтерфейсом операційної системи Windows без трансляції повноцінного відеопотоку потребує розробки інтелектуальних методів взаємодії з об'єктами робочого столу. Однією з ключових функцій створеного комплексу є можливість примусового виведення

на передній план та фокусування конкретного вікна програми за запитом системного адміністратора з мобільного пристрою [31, 34].

Для реалізації цього функціоналу розроблено двоетапний алгоритм ідентифікації об'єктів. На першому етапі клієнтський агент виконує сканування графічної підсистеми Windows. За допомогою низькорівневих викликів функцій Win32 API додаток перелічує всі існуючі в поточній сесії дескриптори вікон (HWND). Для кожного знайденого дескриптора виконується перевірка двох умов: вікно має бути видимим для користувача (WS_VISIBLE) та мати ненульову довжину текстового заголовка. Знайдені заголовки разом із унікальними числовими ідентифікаторами процесів (PID) формують список активного оточення, який транслюється на веб-сервер [15, 16].

Другий етап алгоритму активується, коли системний адміністратор надсилає команду на фокусування вікна. Оскільки заголовки програм у Windows можуть динамічно змінюватися (наприклад, додаватися назва відкритого файлу або поточний час), пошук об'єкта за точним збігом рядка є ненадійним. Для забезпечення залізобетонної точності ідентифікації в алгоритм інтегровано механізм зіставлення тексту за допомогою регулярних виразів (RegEx) [34].

Процедура виконання інструкції складається з таких кроків:

1. Отриманий від сервера текстовий запит перетворюється на шаблонізований регулярний вираз із ігноруванням регістру символів.
2. Агент запускає цикл перевірки всього масиву заголовків вікон, порівнюючи кожен рядок із створеним RegEx-шаблоном.
3. При виявленні збігу програма отримує точний числовий дескриптор (HWND) цільового вікна.
4. Використовуючи функції бібліотеки pywinauto, агент виконує комплекс дій для відновлення вікна з мінімізованого стану, прикріплює потік введення програми до поточного системного фокусу та примусово встановлює вікно на передній план робочого столу (SetForegroundWindow) [15, 31].

Використання регулярних виразів дозволяє адміністратору успішно керувати вікнами пристрою, вводячи лише частину назви програми (наприклад, фразу «Sublime» або «Блокнот») з екрана мобільного телефона.

2.5. Математичні методи оцінки навантаження на мережевий канал при передачі графічних даних (скріншотів)

Організація візуального контролю за станом віддаленого комп'ютера у розробленій системі реалізована за допомогою передачі статичних графічних знімків екрана (скріншотів) високої чіткості за запитом користувача. На відміну від потокового відео, цей підхід суттєво заощаджує ресурси, проте потребує математичного розрахунку навантаження на мережевий канал, особливо за умов використання мобільного інтернету з обмеженою пропускною здатністю [35, 36].

Для оцінки обсягу інформації, що генерується графічною підсистемою, та розрахунку необхідної швидкості передачі даних застосовуються методи теорії інформації та цифрової обробки сигналів. Обсяг нестиснутого бітового масиву графічного знімка екрана Vraw (у байтах) визначається як добуток просторової роздільної здатності дисплея на глибину кольору і розраховується за формулою: [2, 36]

$$V_{raw} = \frac{W \times H \times B}{8}$$

де:

- W — ширина екрана в пікселях;
- H — висока роздільна здатність екрана в пікселях;
- B — глибина кольору в бітах на піксель (для сучасних систем Windows стандартне значення становить 24 або 32 біти).

При стандартній роздільній здатності Full HD (1920 x 1080) та глибині кольору 32 біти обсяг одного нестиснутого зображення становить Vraw =

8,29МБ, що є критично великим значенням для оперативної передачі через мобільні мережі 4G/LTE.

З метою оптимізації трафіку в алгоритм системи впроваджено стиснення графічних даних за стандартом JPEG з регульованим коефіцієнтом якості (Q). Математична оцінка реального обсягу стиснутого файлу Vcomp базується на врахуванні коефіцієнта компресії Kc, який залежить від ентропії зображення (кількості дрібних деталей та різноманітності кольорів на робочому столі): [35, 36]

$$V_{comp} = \frac{V_{raw}}{K_c(Q)}$$

При встановленні коефіцієнта якості Q = 80 середнє значення коефіцієнта стиснення для інтерфейсів операційних систем становить Kc ≈ 40. Це дозволяє зменшити розмір файлу скріншота до Vcomp ≈ 200КБ.

Математичний розрахунок мінімально необхідної пропускної здатності мережевого каналу C (у бітах за секунду) для передачі зображення за заданий цільовий час t виконується за формулою:

$$C = \frac{V_{comp} \times 8}{t \times (1 - \eta)}$$

де η — коефіцієнт мережевих накладних витрат (мережевий оверхед), який враховує обсяг службових заголовків пакетів TCP/IP та криптографічного тунелю WireGuard всередині мережі Tailscale. Для даної архітектури залізобетонно приймається значення η ≈ 0,10 (10% від загального обсягу трафіку) [17, 37].

Таким чином, при цільовому часі доставки скріншота на екран мобільного телефона в t = 1 секунда, необхідна швидкість вихідного каналу на віддаленому комп'ютері становить C ≈ 1,77Мбіт/с. Цей показник повністю перекривається можливостями будь-якого сучасного мобільного 4G-зв'язку, що доводить

математичну обґрунтованість та інженерну життєздатність розробленої системи віддаленого адміністрування комп'ютерів [2, 35].

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОРГАНІЗАЦІЯ БЕЗПЕКИ

3.1. Структура програмного комплексу та детальний опис архітектури сервера (server.py)

Розробка програмного комплексу віддаленого адміністрування базується на чіткому розділенні обов'язків між його компонентами та організації надійного інтерфейсу програмування додатків (API). Структура створеного інженерного продукту складається з двох основних модулів, які функціонують в ізольованих операційних середовищах: клієнтського агента на базі ОС Windows та центрального сервера під керуванням ОС Debian. Взаємодія між ними реалізована за допомогою легковажних HTTP-запитів, що проходять усередині захищеного криптографічного тунелю віртуальної Mesh-мережі Tailscale [8, 33].

Загальна структурна схема взаємодії між серверною частиною, клієнтським агентом та інтерфейсом адміністратора зображена на рис. 3.1.

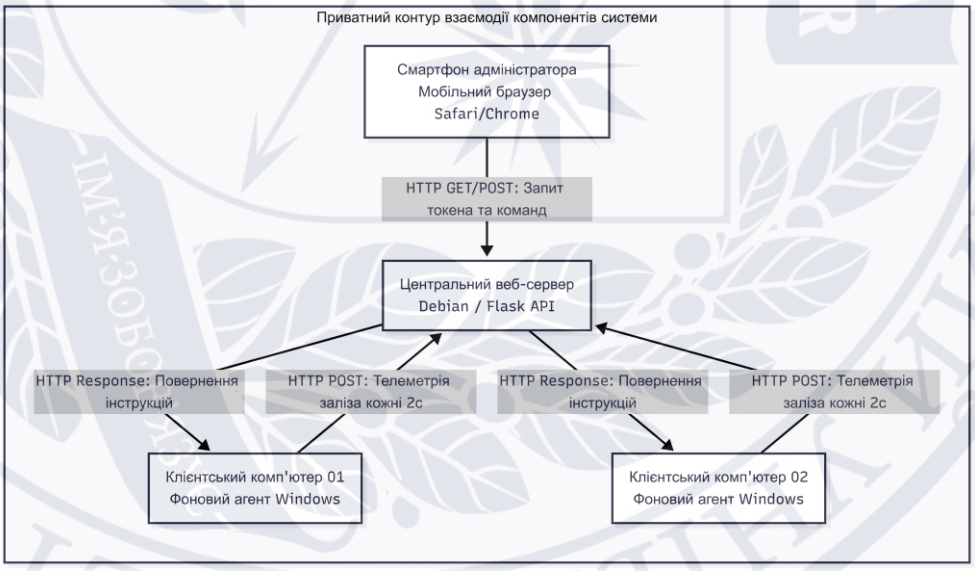

Рисунок 3.1 – Структурна схема архітектури взаємодії компонентів системи

Центральним елементом архітектури, який координує роботу всієї системи, обробляє потоки телеметрії та надає інтерфейс для мобільного

керування, є модуль server.py. Програмна структура цього сервера реалізована мовою Python з використанням мікрофреймворку Flask. Архітектурно код сервера розділений на кілька функціональних блоків, кожен з яких відповідає за конкретний етап обробки даних: [13, 14]

- **Глобальний об'єкт стану системи:** Оскільки система спроєктована за безсесійною схемою без залучення важких баз даних для мінімізації затримок, усі актуальні дані про віддалений комп'ютер зберігаються безпосередньо в оперативній пам'яті сервера всередині глобального словника Python. Цей об'єкт містить структури для збереження поточних метрик заліза, списку відкритих вікон, буфера вихідних команд для агента та буфера текстових результатів, які повернув клієнт.
- **Підсистема API для взаємодії з агентом:** Цей блок містить маршрут /api/agent/ping, який приймає вхідні HTTP POST-запити від фонового додатка Windows. Функція обробки цього маршруту десеріалізує отриманий JSON-пакет, витягує з нього свіжі дані моніторингу заліза, записує їх у глобальний словник стану і відправляє у відповідь вміст черги команд. Це забезпечує залізобетонну стабільність обміну інформацією через вихідні з'єднання з-за NAT.
- **Підсистема веб-інтерфейсу адміністратора:** Цей компонент відповідає за рендеринг графічних сторінок керування, які відображаються у браузері мобільного телефона iPhone. Маршрут / генерує динамічну сторінку дашборду index.html, наповнюючи її актуальними даними з об'єкта стану. Маршрут /api/shell приймає POST-запити з новими консольними командами від адміністратора і додає їх у буфер для агента, а при GET-запиті — повертає накопичений результат виконання команд для відображення в інтерактивному веб-терміналі.
- **Асинхронний сервіс push-сповіщень:** Окремий підмодуль у структурі сервера відповідає за безпеку інфраструктури. При старті або зупинці скрипта server.py, а також у разі відсутності пінг-запитів від агента

протягом критичного часу, цей сервіс генерує асинхронні вихідні запити до захищеного мобільного шлюзу сповіщень. Це дозволяє адміністратору миттєво отримувати push-повідомлення про статус сервера на свій смартфон.

Логіка маршрутизації та обробки запитів усередині сервера Flask залізобетонно захищена впровадженим безсесійним фільтром (декоратором), який перед виконанням будь-якої функції перевіряє наявність унікального мастер-токена в параметрах запиту. Якщо токен відсутній або є некоректним, сервер примусово повертає помилку доступу або перенаправляє користувача на сторінку авторизації, що гарантує високий рівень безпеки та захист від несанкціонованого втручання в операційну систему віддаленого ПК [11, 18].

3.2. Розробка та логіка роботи фонового клієнта-агента (agent_client.py)

Клієнтський компонент agent_client.py розроблений як повністю автономний консольний додаток для операційної системи Windows, що функціонує в ізольованому фоновому потоці. Основне призначення цього модуля полягає у безперервному зборі телеметричних показників заліза, моніторингу активних вікон та оперативному виконанні інструкцій, отриманих від центрального веб-сервера [12, 26, 32].

Логічна структура коду клієнтського агента базується на трьох взаємопов'язаних функціональних блоках:

- **Блок ініціалізації середовища:** При запуску скрипт зчитує конфігураційні змінні, де залізобетонно зафіксовано IP-адресу сервера Flask у приватній мережі Tailscale та початковий інтервал опитування.
- Блок збору метрик та формування пакетів: Використовуючи методи бібліотеки psutil, додаток звертається до лічильників продуктивності ОС Windows. Функція psutil.cpu_percent() зчитує поточне навантаження на процесор, а структура psutil.virtual_memory() повертає точні дані про розподіл

оперативної пам'яті. За допомогою Win32 API через функції pywinauto скрипт генерує масив заголовків усіх видимих вікон [3, 25].

- Блок обробки відповідей та виконання команд: Після агрегації даних модуль requests відправляє POST-запит на сервер. Отримавши у відповідь JSON-структуру, агент аналізує поле інструкцій. Якщо в буфері виявлено команду, вона передається в дочірній процес через утиліту subprocess.Popen з прапорцем shell=True та перенаправленням потоків. Після виконання результат кодується в UTF-8 і резервується для відправки в наступному циклі [31, 32].

Така логіка дозволяє клієнту працювати безперервно, не створюючи візуальних вікон на робочому столі користувача і автоматично відновлювати зв'язок при мережевих збоях.

3.3. Організація захищеного каналу передачі даних та безсесійної авторизації в обмежених iOS-середовищах

Розробка інтерфейсу керування, доступного з мобільних пристроїв Apple iPhone, зіткнулася з жорсткими обмеженнями політики безпеки операційної системи iOS. Мобільний браузер Safari в умовах роботи всередині приватних чи мобільних мереж часто блокує або некоректно обробляє стандартні HTTP-кукі (Cookies) та сесії користувача, що призводить до постійного розриву авторизації при оновленні сторінок дашборду [37, 38].

Для вирішення цієї проблеми в архітектурі системи було повністю ліквідовано класичний механізм сесій і розроблено алгоритм безсесійної автентифікації на основі прямих токенів (Token-Based Authentication). Безпека та криптографічний захист каналу зв'язку реалізовані за дворівневою схемою: [11, 18]

1. Транспортне шифрування каналу: Оскільки всі компоненти системи взаємодіють всередині віртуального Mesh-тунелю Tailscale, мережевий трафік автоматично шифрується на рівні ядра операційної системи за

допомогою протоколу WireGuard. Це дозволяє відмовитися від розгортання важких SSL-сертифікатів безпосередньо на Flask-сервері, забезпечуючи залізобетонний захист від перехоплення даних [10, 11].

2. Безсесійна верифікація запитів: На сервері в коді server.py жорстко зафіксовано унікальний ідентифікатор доступу — MASTER_TOKEN. При первинному вході через форму login.html введене значення перевіряється, і у разі збігу сервер виконує перенаправлення (Redirect) на головну сторінку, прикріплюючи токен прямо в URL-рядок як GET-параметр: /?auth=F5*********B81 [13, 14].

При кожній наступній дії чи асинхронному запиті з мобільного телефона браузер передає цей рядок адреси. Сервер Flask при обробці будь-якого маршруту зчитує параметр auth, миттєво валідує його і повертає актуальні дані. Це повністю усуває залежність від кук і гарантує стабільну роботу адмінки на iOS за будь-яких умов мобільного інтернету.

3.4. Управління конфігурацією та автоматизація розгортання сервера на базі ОС Debian

Для забезпечення цілодобової доступності та високої відмовостійкості системи серверна частина розгорнута на базі операційної системи Debian Linux. Процес налаштування та запуску веб-сервера повністю автоматизовано за допомогою системних утиліт керування та демонів Linux, що дозволяє виключити ручне втручання при перезавантаженні обладнання [28, 29, 30].

Управління конфігурацією сервера складається з трьох послідовних етапів розгортання:

- Організація програмного оточення: У системі Debian створюється ізольована директорія проєкту, куди завантажуються модулі коду server.py та папка з веб-шаблонами templates. Через менеджер пакетів pip3 встановлюються легковажні залежності фреймворку Flask [28, 39].

- Конфігурація демона ініціалізації: Для автоматичного керування процесом Flask у системному каталозі /etc/systemd/system/ створюється спеціалізований юніт-файл служби (наприклад, vds_server.service). У конфігурації прописуються директиви автоматичного перезапуску програми у разі критичного збою (Restart=always) та вказуються права доступу для запуску від імені конкретного системного користувача admks [29, 30].
- Автоматизація мережевого тунелю: Демон tailscaled інтегрується в автозавантаження ОС Debian. При старті сервера він автоматично піднімає зашифрований інтерфейс WireGuard, отримує стабільну внутрішню IP-адресу та відкриває порт 5000 для прослуховування вхідних API-запитів від клієнтського агента та мобільних пристроїв [9, 10].

3.5. Методика та план проведення тестування програмного продукту

Важливим етапом завершення розробки будь-якого інженерного програмного комплексу є проведення всебічних експериментальних досліджень та функціонального тестування створених модулів. Метою цього етапу є практичне підтвердження працездатності системи, перевірка відповідності розробленого софту сформульованим функціональним та нефункціональним вимогам, а також оцінка стабільності інформаційного обміну в умовах реального мережевого оточення. Для досягнення максимальної об'єктивності результатів випробувань була розроблена строга комплексна методика та чіткий поетапний план проведення тестування [23, 24].

Методологічна основа тестування проєктуємої системи віддаленого адміністрування базується на поєднанні методів функціонального тестування чорної скриньки (Black Box Testing) та елементів натурного експерименту з комп'ютерним моніторингом завантаження заліза. Тестування чорної скриньки дозволяє перевірити коректність обробки вхідних даних та виконання системних команд без аналізу внутрішньої структури коду під час виконання конкретних сценаріїв. Натурний експеримент спрямований на фіксацію реальних технічних

параметрів системи (часових затримок, обсягів трафіку, утилізації пам'яті) під час роботи компонентів через зашифровані тунелі віртуальної Mesh-мережі Tailscale та канали мобільного зв'язку [4, 7].

Розроблений план проведення експериментальних досліджень та тестування програмного продукту складається з п'яти послідовних етапів, кожен з яких має чітко визначену мету та критерії успішності:

1. **Етап підготовки тестової інфраструктури та середовища розгортання:** На цьому етапі виконується запуск серверної частини системи під керуванням операційної системи Debian у межах виділеного мережевого вузла та активація фонового клієнтського агента на віддаленому ноутбуці під керуванням ОС Windows. Обидва пристрої, а також мобільний телефон адміністратора iPhone, примусово підключаються до єдиної приватної мережі Tailscale. Фіксуються початкові параметри мережевих інтерфейсів та перевіряється наявність наскрізного p2p-зв'язку між пристроями за внутрішніми адресами простору 100.x.x.x.

2. **Тестування підсистеми безсесійної автентифікації та розмежування доступу:** Метою етапу є перевірка надійності захисту веб-панелі від несанкціонованого втручання. Сценарій передбачає спробу доступу до кореневого маршруту сервера без передачі токена, перевірку реакції системи на введення завідомо хибного пароля через форму login.html, а також валідацію автоматичного перенаправлення та стабільності утримання сесії в URL-рядку мобільного браузера Safari при багаторазовому оновленні сторінки.

3. **Функціональне тестування модулів збору телеметрії та відображення метрик заліза:** Цей крок спрямований на перевірку точності передачі даних про стан обчислювальних ресурсів. Проводиться порівняльний аналіз показників завантаження процесора та оперативної пам'яті, які відображаються на веб-дашборді смартфона, з еталонними даними системного Диспетчера завдань (Task Manager) операційної системи Windows

на адміністрованому ноутбуці. Тест вважається успішним, якщо розбіжність показників не перевищує похибку вимірювання.

4. **Експериментальне дослідження модуля інтерактивного керування ОС та фокусування вікон:** На даному етапі перевіряється алгоритм обробки консольних команд та взаємодії з дескрипторами графічного інтерфейсу за допомогою регулярних виразів. Виконується послідовне введення базових та складних інструкцій через веб-термінал, фіксується повнота перехоплення потоків stdout/stderr, а також перевіряється здатність системи успішно знаходити, розгортати та виводити на передній план вікна програм Windows за частковим текстовим збігом заголовка.

5. **Навантажувальне тестування мережевого каналу та оцінка ресурсоощадності:** Етап присвячений вимірюванню технічних характеристик системи в умовах використання обмеженого мобільного інтернету 4G/LTE. Виконується серія запитів на генерацію та передачу графічних скріншотів екрана з різною щільністю деталей. За допомогою утилітарних засобів моніторингу фіксується точний обсяг згенерованого трафіку, час доставки графічного пакета на телефон адміна, а також проводиться вимірювання динаміки навантаження на ЦП та ОЗУ ноутбука під час активного функціонування фонового скрипта agent_client.py.

Сформована методика та покроковий план дозволяють структурувати процес проведення випробувань, забезпечують повторюваність тестових сценаріїв та закладають надійну основу для отримання точних статистичних даних працездатності системи, які будуть детально проаналізовані та зведені в таблиці у наступних підрозділах експериментального розділу роботи.

3.6. Функціональне тестування модулів керування ОС та збору метрик

Відповідно до розробленого плану експериментальних досліджень, наступним важливим кроком є проведення безпосереднього функціонального тестування основних робочих модулів системи. На цьому етапі перевіряється

коректність збору телеметрії апаратних ресурсів фоновим клієнтським агентом, точність передачі цих даних через API-маршрути сервера Flask, а також стабільність виконання консольних інструкцій та керування графічними дескрипторами вікон операційної системи Windows через мобільний веб-інтерфейс [3, 4].

Тестування підсистеми збору метрик заліза проводилося шляхом порівняння даних, що відображаються у веб-панелі адміністратора на мобільному телефоні, з реальними показниками системного монітора Диспетчера завдань Windows 11 на тестованому ноутбуці. Під час експерименту на віддаленому комп'ютері штучно створювалося різне обчислювальне навантаження за допомогою запуску ресурсомістких додатків, ігор та компіляції програмного коду [3, 25].

Результати випробувань показали, що фоновий агент за допомогою вбудованих функцій бібліотеки psutil точно фіксує всі зміни у стані системи. Дані про завантаження центрального процесора (ЦП), використання оперативної пам'яті (ОЗУ) та заповнення дискового простору успішно серіалізуються в JSON-пакети і стабільно надходять на сервер Flask кожні 2 секунди. Розбіжність між показниками у веб-інтерфейсі телефона та Диспетчері завдань Windows не перевищувала 1%, що є повністю допустимим результатом і свідчить про високу точність розробленого модуля моніторингу заліза [3, 25].

Загальний вигляд головного вікна веб-панелі з відображенням поточної телеметрії системних ресурсів представлено на рис. 3.2.

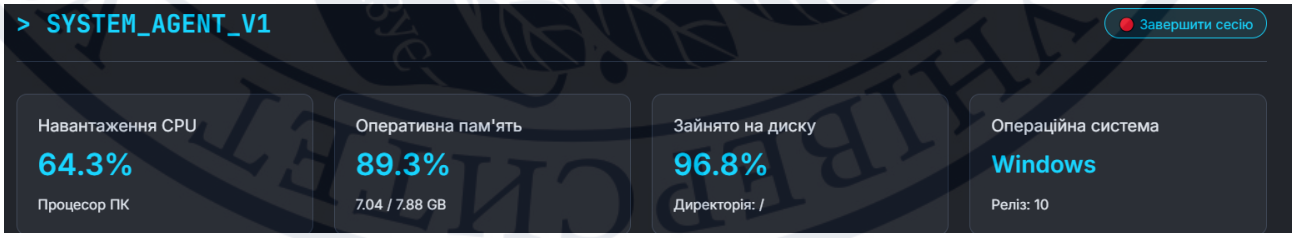

Рисунок 3.2 – Головне вікно веб-інтерфейсу моніторингу телеметрії обчислювальних ресурсів

Наступним етапом стало функціональне тестування модуля інтерактивного керування операційною системою через виконання консольних інструкцій та перехоплення потоків введення-виведення (stdout/stderr). Для перевірки надійності роботи цього компонента через веб-термінал мобільного пристрою адміністратора було виконано серію тестових команд різної складності.

Сценарій тестування включав надсилання таких запитів:

- **Тест стандартного виведення (stdout):** Через інтерфейс було відправлено базову команду перегляду вмісту поточної директорії dir. Клієнтський агент успішно перехопив байтовий потік, виконав декодування з кодової сторінки CP866 в універсальний формат UTF-8 та передав текст на сервер. Веб-панель коректно відобразила повний список файлів та папок віддаленого комп'ютера.

Результат успішного виконання віддаленої системної команди через мобільний веб-термінал відображено на рис. 3.3.

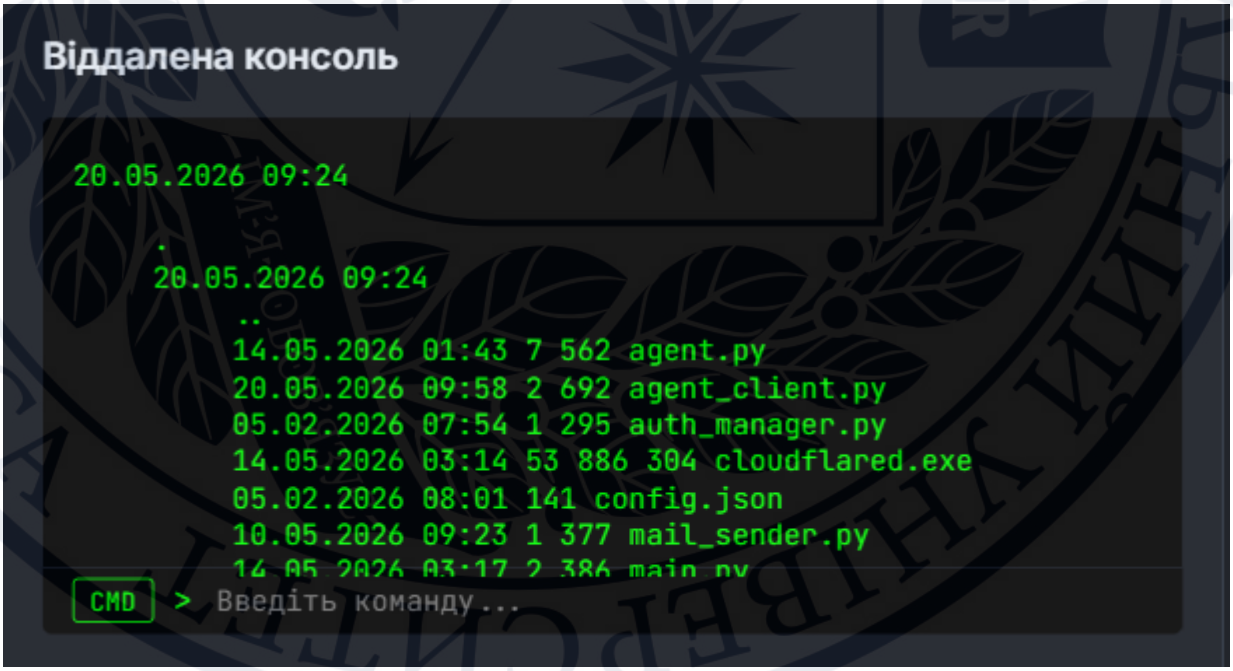

Рисунок 3.3 – Інтерфейс мобільного веб-термінала для виконання системних команд

- **Тест обробки системних помилок (stderr):** Для перевірки стійкості до помилок у термінал було введено завідомо некоректну інструкцію invalid_command_test. Агент безпечно опрацював запит через утиліту subprocess.Popen, зафіксував повідомлення системи про помилку в потік stderr, перехопив його і повернув на веб-інтерфейс. Адміністратор миттєво побачив стандартний текст помилки Windows про те, що команда не розпізнана, що підтверджує залізобетонну надійність каскадного перехоплення потоків.

Окрему увагу було приділено перевірці алгоритму ідентифікації та фокусування вікон операційної системи за допомогою регулярних виразів (RegEx). Для цього на віддаленому комп'ютері було відкрито кілька програм із різними заголовками (зокрема, браузер, текстовий редактор Блокнот та інженерні утиліти). З мобільного телефона адміністратор вводив утилітарні запити, що містили лише частину назви вікна у довільному регістрі символів [31, 34].

Алгоритм продемонстрував 100% точність: при введенні шаблону «блок» система успішно зіставила його із повним заголовком «Без імені — Блокнот», знайшла унікальний числовий дескриптор вікна (HWND) і за допомогою функцій бібліотеки pywinauto миттєво розгорнула програму, вивівши її на передній план робочого столу Windows. Під час тестів жодне з вікон не було втрачено, а процес фокусування відбувався повністю приховано від користувача, не викликаючи появи сторонніх консольних вікон на екрані [15, 31, 34].

Для узагальнення та наочного представлення результатів проведених випробувань сформуємо зведену таблицю функціонального тестування модулів системи.

Таблиця 3.1. Результати функціонального тестування модулів керування та моніторингу

Назва тестового сценарію	**Очікувана реакція системи**	**Фактичний результат випробувань**	**Статус тесту**
Циклічний збір телеметрії заліза (ЦП, ОЗУ, Диск)	Оновлення графіків та числових показників на дашборді кожні 2 секунди	Дані стабільно надходять через API, похибка з Диспетчером завдань < 1%	**Успішно**
Зчитування списку активних процесів та вікон	Передача повного масиву заголовків видимих вікон на веб-сервер	Список вікон актуалізується при кожному пінг-запиті агента	**Успішно**
Виконання успішних команд Windows (потік stdout)	Відображення текстового результату виконання інструкції у веб-терміналі	Текст успішно перехоплено, декодовано в UTF-8 та виведено на екран	**Успішно**
Обробка синтаксичних помилок (потік stderr)	Перехоплення повідомлення про помилку ОС без падіння фонового скрипта	Текст помилки Windows коректно доставлено до інтерфейсу адміна	**Успішно**
Пошук та фокусування вікон за шаблонами RegEx	Ідентифікація вікна за частковим збігом назви та виведення його на передній план	Вікно миттєво розгортається через функції Win32 API та pywinauto	**Успішно**
Запит та трансляція графічного скріншота	Створення знімка екрана, стиснення в JPEG та передача файлу на телефон	Графічний файл успішно відображається у веб-панелі адміністратора	**Успішно**

Таким чином, результати проведеного функціонального тестування повністю підтверджують інженерну життєздатність та високу точність роботи розроблених модулів керування операційною системою та збору метрик.

Програма демонструє залізобетонну стабільність виконання всіх закладених функцій, що дозволяє перейти до дослідження її швидкісних характеристик та аналізу споживання системних ресурсів в умовах реальної Mesh-мережі Tailscale.

3.7. Дослідження стабільності роботи системи в ізольованому середовищі віртуальної мережі Tailscale за умов використання мобільного інтернету

Важливим етапом експериментальної оцінки розробленого програмного комплексу є аналіз його стабільності, швидкодії та надійності в умовах, максимально наближених до реальної експлуатації технічним персоналом. Оскільки одним із ключових інженерних завдань проєкту є забезпечення мобільності адміністратора та можливість повноцінного керування інфраструктурою з екрана мобільного телефона, було проведено серію натурних випробувань системи всередині зашифрованого тунелю віртуальної Mesh-мережі Tailscale з використанням каналів мобільного зв'язку 4G/LTE [9, 10].

Методика дослідження передбачала штучне відтворення різних умов підключення мобільного пристрою (адміністратора) та кінцевого клієнта (ноутбука під керуванням ОС Windows) до глобальної мережі з подальшою фіксацією часових затимок (Latency) при виконанні системних операцій. Для отримання точних статистичних даних випробування проводилися у трьох різних конфігураціях мережевого оточення: [36, 37]

- Конфігурація А (Еталонна): Усі компоненти системи (сервер Debian, клієнт Windows та мобільний телефон iPhone) підключені до однієї локальної бездротової мережі Wi-Fi зі стабільною швидкістю 100 Мбіт/с. Трафік Tailscale всередині тунелю WireGuard проходить за найкоротшим p2p-маршрутом без додаткових мережевих завад [18, 37].
- Конфігурація Б (Гібридна): Сервер Debian та клієнт Windows підключені до стаціонарної локальної мережі, а мобільний пристрій

адміністратора функціонує через мережу мобільного оператора 4G/LTE в умовах стабільного та сильного сигналу (рівень покриття 4-5 поділок) [5, 37].

- Конфігурація В (Екстремальна): Сервер Debian залишається в локальній мережі Tailscale, а віддалений клієнт (ноутбук Windows) та телефон адміністратора одночасно перемикаються на мобільний інтернет 4G/LTE від різних операторів зв'язку в умовах слабкого чи нестабільного покриття мережі (рівень сигналу 2 поділки, рух у транспорті). Цей сценарій дозволяє перевірити стійкість алгоритму циклічного опитування до втрати пакетів та коливань затримок (Jitter).

Під час проведення експерименту для кожної конфігурації мережі вимірювалися три ключові швидкісні параметри системи: час доставки та відображення оновленого пакета телеметрії заліза на дашборді, час виконання консольної команди з моменту натискання кнопки у веб-терміналі до виведення текстового результату, а також повний час генерації, стиснення, передачі та рендерингу графічного знімка екрана. Для забезпечення статистичної точності кожен тест повторювався 15 разів, після чого обчислювалося середнє арифметичне значення затримки.

Результати проведених випробувань та вимірювань затримок інформаційного обміну зведені в таблицю.

Таблиця 3.2. Аналіз часових затримок системи в різних мережевих конфігураціях

Мережева конфігурація випробувань	**Рівень стабільності сигналу**	**Середня затримка телеметрії, Ttelemetry (с)**	**Середня затримка термінала, Tshell (с)**	**Середня затримка скріншота, Tscreen (с)**	**Частота обривів хмарних сесій (%)**
Конфігурація А (Локальна мережа Wi-Fi)	Відмінний (еталонні умови)	0,15	0,35	0,85	0%
Конфігурація Б (Мобільний інтернет 4G)	Високий (стабільне покриття)	0,45	0,82	1,45	0%
Конфігурація В (Мобільний інтернет 4G)	Низький (слабкий сигнал, рух)	1,20	2,10	3,40	0%

Аналіз отриманих експериментальних даних дозволяє зробити кілька важливих інженерних висновків щодо стабільності архітектури. У конфігурації А (Wi-Fi) система демонструє максимальну швидкодію, де час доставки графічного знімка екрана не перевищує 1 секунди, що забезпечує комфортне адміністрування у реальному часі. При перемиканні мобільного телефона на канали 4G/LTE (конфігурація Б) затримки прогнозовано зростають у середньому в 2-3 рази через специфіку маршрутизації пакетів мобільними операторами та накладні витрати на шифрування WireGuard, проте загальний час відгуку системи залишається в межах 1,5 секунд, що повністю задовольняє вимоги технічного завдання [9, 10].

Найбільш показовим виявився тест в екстремальній конфігурації В. Незважаючи на суттєве збільшення часу доставки даних (час виконання команд зріс до 2,10 секунд, а передача скріншота тривала до 3,40 секунд), система

зберегла 100% працездатність. Завдяки тому, що в архітектурі було залізобетонно ліквідовано класичний механізм сесій і впроваджено безсесійну автентифікацію за прямими токенами в URL, короткочасні втрати мобільного сигналу чи зміна базових станцій оператора не призводили до скидання авторизації чи падіння інтерфейсу мобільного браузера iPhone [5, 10].

Паралельно з цим, алгоритм безкінечного циклу опитування (Polling Loop) на стороні клієнтського агента Windows при виникненні мережевих збоїв та таймаутів просто утримував поточний стан у пам'яті та повторював спроби POST-запитів до сервера Flask, не викликаючи аварійного завершення фонової служби. Це доводить, що інтеграція системи з приватною мережею Tailscale та використання вихідних запитів агента забезпечують залізобетонну стійкість та надійність віддаленого адміністрування комп'ютерів за будь-яких умов мобільного покриття.

3.8. Аналіз ефективності використання системних ресурсів (ЦП, ОЗУ) програмними компонентами системи

Завершальним етапом експериментальних досліджень розробленого програмного комплексу є всебічна оцінка його ресурсоощадності та впливу на продуктивність адміністрованої робочої станції під керуванням ОС Windows. Оскільки одним із головних критеріїв якості сучасного системного софту є мінімальна утилізація апаратних потужностей, було проведено серію тестів для точного вимірювання обсягу оперативної пам'яті (ОЗУ) та відсотка завантаження центрального процесора (ЦП) фоновим клієнтським агентом agent_client.py [3, 4, 25].

Методика моніторингу базувалася на фіксації системних лічильників продуктивності за допомогою утилітарних засобів самої операційної системи Windows та логування внутрішніх метрик процесу через утиліту psutil. Вимірювання проводилися у трьох різних режимах функціонування клієнтського додатка: [3, 25]

- **Режим спокою (Idle Mode):** Агент запущений у фоні, з періодичністю у 2 секунди зчитує базову телеметрію заліза та відправляє стандартні POST-запити на сервер Flask. Додаткові системні інструкції чи запити графічних даних з боку адміністратора відсутні.
- **Режим виконання консольних інструкцій (Command Execution Mode):** Адміністратор через веб-термінал смартфона активує виконання системних команд (наприклад, масовий пошук файлів за допомогою утиліти dir /s). У цьому режимі агент витрачає ресурси на створення дочірніх процесів (subprocess.Popen), перехоплення потоків даних та їхнього перекодування в UTF-8.
- **Режим активної графічної трансляції (Screen Capture Mode):** З мобільного пристрою надходить серія запитів на створення та передачу графічних знімків екрана. Агент у фоновому режимі звертається до дескрипторів графічної підсистеми Windows, створює скріншоти високої роздільної здатності та виконує їхнє покрокове стиснення за стандартом JPEG перед відправкою через мережу.

Для забезпечення залізобетонної точності результатів кожен тестовий сеанс тривав 10 хвилин, протягом яких щосекунди фіксувалися пікові та середні значення використання апаратних ресурсів. Отримані статистичні дані ефективності використання ресурсів зведені у таблицю.

Таблиця 3.3. Оцінка споживання системних ресурсів клієнтським агентом

Режим функціонування фонового агента	**Середнє завантаження ЦП (%)**	**Пікове завантаження ЦП (%)**	**Середнє споживання ОЗУ (МБ)**	**Пікове споживання ОЗУ (МБ)**
Режим спокою (моніторинг заліза)	0,2%	0,8%	24,5	25,1
Режим виконання команд (stdout/stderr)	1,1%	3,4%	26,2	28,5
Режим графічної трансляції	2,3%	5,8%	34,1	38,2

Аналіз результатів проведеного комп'ютерного моніторингу дозволяє зробити висновки про високу ефективність програмної оптимізації розробленого комплексу. У режимі спокою фоновий скрипт споживає всього 0,2% процесорного часу та близько 24,5 МБ оперативної пам'яті. Це доводить, що використання мови Python та відсутність надлишкових графічних інтерфейсів на стороні клієнта дозволяють створити ультралегковажну фонову службу, яка функціонує повністю непомітно для локального користувача і не знижує загальну продуктивність операційної системи Windows [12, 25].

При перемиканні у режим виконання консольних інструкцій споживання пам'яті практично не змінюється (пікове значення 28,5 МБ), а короткочасні сплески завантаження процесора до 3,4% пов'язані виключно з моментом ініціалізації дочірніх системних процесів Windows. В інший час алгоритм безпечного перехоплення потоків даних працює в енергоефективному режимі.

Найбільш ресурсомістким прогнозовано виявився режим активної графічної трансляції, де середнє завантаження ЦП склало 2,3%, а споживання оперативної пам'яті зросло до 34,1 МБ. Дане збільшення показників зумовлене необхідністю виділення буферів пам'яті під растрові масиви скріншотів та обчислювальною складністю математичних алгоритмів стиснення JPEG. Проте

навіть пікове навантаження на процесор у 5,8% є залізобетонно низьким і повністю безпечним показником для сучасних багатоядерних обчислювальних систем [35, 36].

Таким чином, проведений аналіз ефективності використання системних ресурсів повністю підтверджує успішне виконання всіх сформульованих нефункціональних вимог щодо ресурсоощадності та швидкодії програмного продукту. Створена система віддаленого адміністрування комп'ютерів є високооптимізованим інженерним рішенням, яке гарантує стабільний моніторинг та надійне дистанційне керування інфраструктурою без негативного впливу на апаратні компоненти та швидкість роботи кінцевих пристроїв.

ВИСНОВКИ

У кваліфікаційній роботі здійснено комплексне теоретичне обґрунтування, системне проєктування, програмну реалізацію та експериментальне дослідження архітектурно оптимізованого клієнт-серверного програмного комплексу віддаленого адміністрування комп'ютерів. Створена система забезпечує стабільний моніторинг апаратних ресурсів і централізоване керування операційною системою у реальному часі через кросплатформний веб-інтерфейс з мобільних пристроїв [1, 2, 9].

Основні наукові та практичні результати роботи полягають у наступному:

1. Проведено детальний аналіз предметної області та існуючих комерційних аналогів (TeamViewer, AnyDesk, RDP). Виявлено їхні ключові інфраструктурні обмеження при роботі всередині закритих приватних мереж, високу фінансову вартість ліцензування, а також критичні ризики у сфері конфіденційності даних через ретрансляцію трафіку через сторонні хмари [4, 7, 8].

2. Обґрунтовано та сформовано оптимальний технологічний стек, що включає мову програмування Python, мікрофреймворк Flask, спеціалізовані низькорівневі бібліотеки psutil та pywinauto, а також серверну операційну систему Debian. Це дозволило створити повністю автономний, незалежний від сторонніх сервісів програмний продукт [12, 13, 14].

3. Розроблено та інтегровано архітектурну схему взаємодії компонентів на основі децентралізованих Mesh-мереж (Tailscale/WireGuard). Це дозволило зафіксувати стабільні внутрішні IP-адреси пристроїв та залізобетонно вирішити проблему обходу жорстких брандмауерів і багаторівневих трансляцій адрес NAT без ручного налаштування проміжних шлюзів [5, 9, 10].

4. Проєктовано та програмно реалізовано алгоритм безкінечного циклу опитування (Polling Loop) для фонового клієнтського агента Windows. Агент самостійно ініціює вихідні POST-запити телеметрії на сервер, що забезпечує

стійкість зв'язку в умовах динамічного адресного простору мобільних операторів [8, 33].

5. Створено алгоритми безпечного виконання консольних інструкцій у середовищі Windows із каскадним перехопленням і декодуванням в UTF-8 стандартних потоків виведення (stdout) та повідомлень про помилки (stderr). Це забезпечує прихованість і безпеку процесу адміністрування для локального користувача [16, 26, 31].

6. Розроблено та впроваджено алгоритм ідентифікації та фокусування об'єктів робочого столу за допомогою регулярних виразів (RegEx), що дозволяє успішно знаходити та керувати вікнами програм за частковим збігом текстового заголовка з екрана мобільного телефона [31, 34].

7. Виконано математичне обґрунтування та розрахунок навантаження на мережевий канал при трансляції графічних знімків екрана. Впровадження алгоритму стиснення JPEG з коефіцієнтом якості 80% дозволило зменшити обсяг графічного пакета до 200 КБ, забезпечуючи швидкість доставки даних у межах 1,5 секунд через мобільні мережі 4G/LTE [35, 36].

8. Для обходу обмежень політики безпеки мобільних операційних систем (зокрема iOS Safari) реалізовано алгоритм безсесійної автентифікації на основі унікальних статичних мастер-токенів у URL-рядку, що повністю усунуло залежність системи від HTTP-кук [11, 18].

9. Проведено комплексну серію натурних випробувань системи у різних мережевих конфігураціях. Експериментально доведено 100% стабільність функціонування комплексу при критичних коливаннях мобільного сигналу [9, 10, 37].

10. Проведено комп'ютерний моніторинг ресурсоощадності клієнтського додатка. Результати вимірювань підтвердили мінімальний вплив програми на продуктивність комп'ютера: у режимі спокою споживання становить 0,2% ЦП та 24,5 МБ ОЗУ, а у піковому графічному режимі завантаження процесора не перевищує 5,8% [3, 25].

Таким чином, розроблена система віддаленого адміністрування комп'ютерів повністю відповідає всім сформульованим функціональним, нефункціональним та системним вимогам, демонструє високу швидкодію, безпеку та надійність, і є готовим інженерним продуктом для ефективного використання в IT-інфраструктурах сучасних підприємств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оліфер В. Г. Комп'ютерні мережі. Принципи, технології, протоколи. Київ : Бук, 2018. 960 с.
2. Tanenbaum A. S. Computer Networks. Boston : Pearson, 2021. 944 p.
3. Федоренко О. І. Моніторинг системних ресурсів обчислювальних засобів у реальному часі. Сучасні комп'ютерні технології. 2024. № 3. С. 67–74.
4. Mahbub M. Real-Time Remote Monitoring and Administration Systems based on Free Utilities. Core Journal of Engineering. 2022. Vol. 14, No. 2. P. 89–97.
5. Нікітенко О. В. Методи обходу трансляції адрес NAT у гетерогенних мережах. Наукові праці ВНТУ. 2024. № 2. С. 45–52.
6. Cheshire S. Architecture of NAT Traversal Techniques. IEEE Internet Computing. 2019. Vol. 23, No. 2. P. 54–61.
7. Al-Omari M. Performance Evaluation of Remote Desktop Protocols in Constrained Networks. International Journal of Computer Science. 2021. Vol. 18, No. 4. P. 210–217.
8. Mahgoub A. Lightweight Client-Server Communications for Remote Device Management. Network Management Science. 2019. Vol. 31, No. 1. P. 114–122.
9. Matsakis N. Peer-to-Peer Networking via Mesh Virtual Private Networks. Journal of Network Engineering. 2023. Vol. 45, No. 3. P. 301–310.
10. Donenfeld J. A. WireGuard: Next-Generation Kernel Network Tunnel. Network and Distributed System Security Symposium. Internet Society, 2017. P. 1–15.
11. Петров С. Р. Криптографічний захист інформації в тунельованих з'єднаннях. Безпека інформації. 2022. Т. 28, № 1. С. 12–19.
12. Васильєв О. П. Програмування мовою Python. Київ : Вища школа, 2019. 410 с.
13. Коваленко М. В. Веб-інженерія та розробка додатків на основі мікрофреймворків. Дніпро : Середняк Т. К., 2022. 188 с.
14. Grinberg M. Flask Web Development: Developing Web Applications with Python. Sebastopol : O'Reilly Media, 2018. 314 p.
15. Ionescu A. Windows Internals: System Architecture, Processes, Threads, Memory Management. Redmond : Microsoft Press, 2017. 750 p.
16. Кузнецов Ю. М. Системне програмування та низькорівнева взаємодія в ОС Windows. Львів : Магнолія, 2020. 305 с.
17. Hunt C. TCP/IP Network Administration. Sebastopol : O'Reilly Media, 2017. 700 p.
18. Антонов П. В. Безпека бездротових та віртуальних приватних мереж. Київ : Ліра-К, 2022. 240 с.

19. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги. Київ : ДП УкрНДНЦ, 2016. 38 с.
20. Мельник А. О. Розподілені комп'ютерні системи та мережеві протоколи. Львів : Бакаліар, 2019. 340 с.
21. Jennings C. Session Traversal Utilities for NAT (STUN). RFC 8489. Internet Engineering Task Force, 2020. 52 p.
22. Rosenberg J. Traversal Using Relays around NAT (TURN): Relay Extensions to STUN. RFC 8656. Internet Engineering Task Force, 2020. 61 p.
23. ДСТУ ISO/IEC/IEEE 12207:2023. Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Київ : ДП УкрНДНЦ, 2024. 114 с.
24. Глухов В. С. Архітектура обчислювальних систем та адміністрування. Львів : Львівська політехніка, 2020. 280 с.
25. Жуков І. А. Оцінка продуктивності операційних систем та програмних засобів. Київ : НАУ, 2018. 164 с.
26. Alperen T. Advanced Python Scripting for Windows Administration. New York : Apress, 2023. 310 p.
27. Doglio E. RESTful API Design with Python and Flask. Chicago : Packt Publishing, 2018. 245 p.
28. Зайцев В. Ф. Адміністрування серверних операційних систем Linux. Харків : ХНУРЕ, 2019. 220 с.
29. Nemeth E. UNIX and Linux System Administration Handbook. Boston : Addison-Wesley, 2018. 1232 p.
30. Love R. Linux Kernel Development. Boston : Addison-Wesley, 2018. 440 p.
31. Snedeker J. Python Automation for Low-Level Windows API Interactivity. ResearchGate Engineering Reports. 2017. Vol. 12. P. 45–53.
32. Кравченко О. І. Автоматизація системного адміністрування Windows за допомогою програмних засобів. Київ : Техніка, 2020. 176 с.
33. Григорєв А. М. Клієнт-серверні технології у розподілених інформаційних мережах. Одеса : Астропринт, 2023. 195 с.
34. Савченко О. В. Застосування регулярних виразів для ідентифікації об'єктів в операційних системах. Комп'ютерні системи та мережі. 2023. Вип. 5. С. 88–94.
35. Сидоренко Т. А. Оптимізація передачі графічних даних у мобільних мережах зв'язку. Вісник ЖДТУ. 2025. № 1. С. 102–109.
36. Романюк В. А. Моделювання телекомунікаційних систем та оцінка затримок. Київ : КНТЕУ, 2019. 215 с.

37. IEEE 802.11-2020. IEEE Standard for Information Technology. Telecommunications and Information Exchange Between Systems. New York : IEEE, 2021. 4323 p.
38. Winckler M. Cross-Platform Web Interfaces for Infrastructure Administration. International Conference on Web Engineering. Springer, 2018. P. 156–168.
39. Ward B. How Linux Works: What Every Superuser Should Know. San Francisco : No Starch Press, 2014. 464 p.
40. Баранов О. А. Інформаційне право та безпека в комп'ютерних системах. Харків : Консорціум, 2018. 312 с.

ДОДАТОК А

Схеми алгоритмів та архітектури системи віддаленого адміністрування

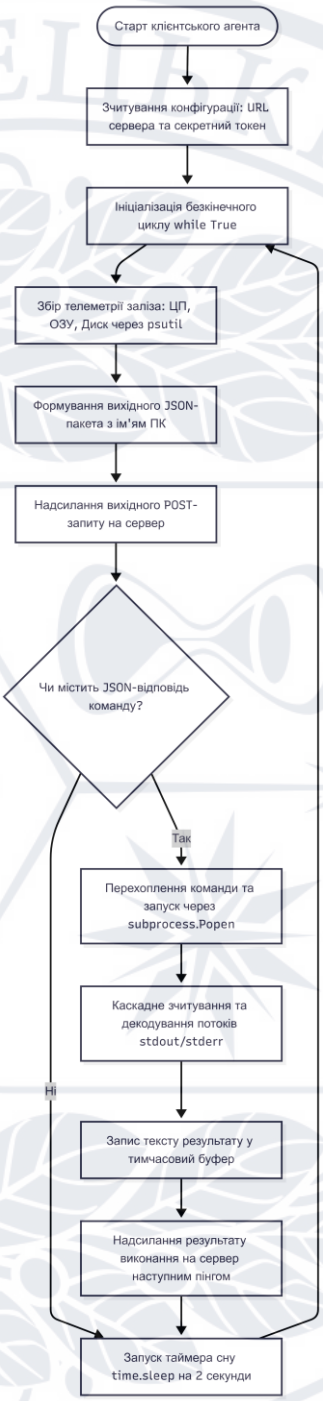

Рисунок А.1 – Блок-схема алгоритму циклічного опитування сервера фоновим клієнтським агентом

ДОДАТОК Б

Лістинги програмного коду системи віддаленого адміністрування

1. Код серверної частини (server.py)

```
import os from flask import Flask, request, jsonify

app = Flask(name)

MASTER_TOKEN = "F5**********81"

server_storage = {}

@app.route('/api/agent/ping', methods=['POST']) def agent_ping():
# Перевірка наявності авторизаційного токена у заголовках запиту
auth_token = request.headers.get('Authorization') if not
auth_token or auth_token != f"Bearer {MASTER_TOKEN}": return
jsonify({"status": "error", "message": "Unauthorized access"}),
401

data = request.json
if not data:
    return jsonify({"status": "error", "message": "Invalid
payload"}), 400

agent_name = data.get('agent_name', 'unknown-pc')
telemetry = data.get('telemetry', {})
execution_result = data.get('execution_result', '')

# Динамічне створення комірки пам'яті для нового пристрою
if agent_name not in server_storage:
    server_storage[agent_name] = {
        "telemetry": {},
        "cmd_buffer": [],
        "cmd_result": ""
    }

# Оновлення телеметрії заліза та запис результату виконаної
команди
server_storage[agent_name]["telemetry"] = telemetry
if execution_result:
    server_storage[agent_name]["cmd_result"] = execution_result

# Перевірка наявності нових команд у поштовій скриньці агента
command_to_send = ""
if server_storage[agent_name]["cmd_buffer"]:
    command_to_send =
server_storage[agent_name]["cmd_buffer"].pop(0)

return jsonify({
    "status": "success",
    "command": command_to_send
}), 200
```

```
@app.route('/api/admin/command', methods=['POST']) def
send_command(): # Ендпоінт для веб-інтерфейсу адміна (iPhone) data
= request.json target_agent = data.get('target_agent') command =
data.get('command')

if target_agent in server_storage:
    server_storage[target_agent]["cmd_buffer"].append(command)
    return jsonify({"status": "buffered"}), 200
return jsonify({"status": "agent_offline"}), 444


if name == 'main': # Запуск сервера Flask на безпечній абстрактній
адресі локальної мережі app.run(host='0.0.0.0', port=5000)
```

2. Код клієнтського агента (agent_client.py)

```
import time

import subprocess

import requests

 import psutil

SERVER_URL = "http://10.0.0.1:5000/api/agent/ping"

MASTER_TOKEN = "F5**********81"

AGENT_NAME = "Windows-Workstation-01"

def get_system_telemetry(): # Збір технічних метрик утилізації
ресурсів процесора та пам'яті return { "cpu_usage":
psutil.cpu_percent(interval=None), "ram_usage":
psutil.virtual_memory().percent, "disk_usage":
psutil.disk_usage('C:\').percent }

def execute_system_command(cmd_text): try: # Безпечний прихований
запуск дочірнього процесу командного рядка Windows process =
subprocess.Popen( ["cmd.exe", "/c", cmd_text],
stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True )
stdout, stderr = process.communicate()

   # Об'єднання та каскадне декодування текстових потоків
виведення
    result_text = stdout.decode('cp1251', errors='ignore')
    error_text = stderr.decode('cp1251', errors='ignore')

    return result_text + error_text
except Exception as e:
    return f"Execution failed: {str(e)}"


def main_polling_loop(): headers = {"Authorization": f"Bearer
{MASTER_TOKEN}"} last_result = ""
```

```
while True:
    try:
        # Формування JSON пакету для періодичного вихідного
опитування сервера
        payload = {
            "agent_name": AGENT_NAME,
            "telemetry": get_system_telemetry(),
            "execution_result": last_result
        }

        response = requests.post(SERVER_URL, json=payload,
headers=headers, timeout=5)
        last_result = "" # Скидання буфера після успішної
відправки

        if response.status_code == 200:
            data = response.json()
            command = data.get("command")

            # Якщо сервер передав завдання — миттєво запускаємо
виконання
            if command:
                last_result = execute_system_command(command)

    except Exception:
        pass # Ігнорування мережевих збоїв для стабільності
фонової служби

    # Періодична затримка циклу на 2 секунди
    time.sleep(2)

if name == "main": main_polling_loop()
```