Plag

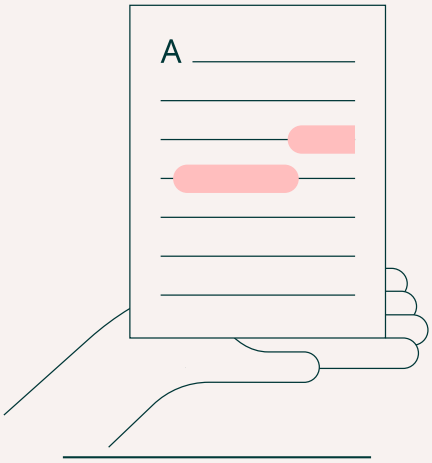
Звіт про оригінальність

● Оцінка схожості 4%

● Ризик плагіату НИЗЬКИЙ

Ігор Кагало 2025-06-14 10:25

Посилання на звіт: 10biF / Посилання користувача: qAHy



Ось вона –
Ваша звіт про оригінальність!

Ми раді повідомити, що перевірка вашого документа завершена, і результати вже готові! Наші алгоритми старанно працювали, щоб знайти збіги в наших базах даних.

На наступних сторінках ви знайдете результати перевірки:

Бали
Збіги
Посилання

Ваш документ було перевірено за такими джерелами:

- ✓ База даних інтернет-джерел
- ✓ База даних наукових статей
- ○ Глибока перевірка (наш вдосконалений алгоритм)

Бали

Збіги тексту	4%
Перефразування	0%
Цитований текст	0%
Неправильне цитування	0%
Збігів не знайдено	96%

Ризик плагіату НИЗЬКИЙ

Ризик плагіату вказує, як збіги тексту розподілені по документу. Вищий ризик виникає, коли збіги з'являються близько один до одного, наприклад, у тому самому абзаці або розділі.

Оцінка схожості 4%

Оцінка схожості показує, скільки слів або символів у вашому документі збігаються з текстами інших документів, включаючи перефразовані тексти або неправильні цитати.

Збіги

ВСТУП

Дана дипломна робота присвячена розробці системи моніторингу та адміністрування IT-інфраструктури для підприємств. У сучасному бізнес-середовищі ефективна робота інформаційних технологій є критично важливою для успіху. Зростаюча складність IT-інфраструктури вимагає наявності інструментів для відстеження її стану та оперативного реагування на проблеми. Розробка інтегрованої системи моніторингу та адміністрування сприятиме підвищенню надійності IT-сервісів та оптимізації витрат.

Основною метою роботи є створення системи, що забезпечить моніторинг ключових параметрів IT-інфраструктури (персональних комп'ютерів, мережі) та надасть базові інструменти адміністрування (перегляд данних про ПК та мережу). Для досягнення мети необхідно проаналізувати потреби бізнесу, 9 спроектувати архітектуру системи, розробити інтерфейс, запрограмувати модулі моніторингу та адміністрування, налаштувати оповіщення, 9 протестувати та оцінити ефективність розробленої системи.

9 Ефективний моніторинг та адміністрування IT-інфраструктури є ключовим фактором конкурентоздатності бізнесу в умовах зростаючої залежності від технологій. Системи моніторингу дозволяють своєчасно виявляти проблеми, а інструменти адміністрування спрощують роботу IT-персоналу.

Актуальність роботи обумовлена потребою підприємств у надійних інструментах для управління IT-інфраструктурою. Успішне впровадження розробленої системи підвищить доступність IT-сервісів, скоротить час простою, оптимізує роботу IT-відділу та покращить безпеку інформаційних систем, 20 сприяючи ефективнішому використанню ресурсів та зниженню витрат.

Очікується, що розроблена система матиме зручний інтерфейс та надаватиме адміністраторам всю необхідну інформацію для ефективного управління IT-середовищем. Результати даної роботи можуть бути використані підприємствами для оптимізації своїх IT-процесів.

Таким чином, об'єктом дослідження даної дипломної роботи є процес розробки

програмних засобів для моніторингу та адміністрування комп'ютерних систем. 5 В якості предмета дослідження виступають моделі, методи та інструментальні засоби для реалізації такого продукту у вигляді десктопного додатку на мові Python. 5 Для досягнення поставленої мети та розкриття предмету дослідження було використано комплекс наукових та практичних методів: аналіз науково-технічної літератури та існуючих програмних рішень, системний аналіз для визначення вимог, об'єктно-орієнтоване проектування для створення архітектури, а також програмна реалізація та емпіричне тестування розробленого продукту.

ДОСЛІДЖЕННЯ СФЕРИ РОЗРОБКИ СИСТЕМ МОНІТОРИНГУ ТА АДМІНІСТРУВАННЯ IT-ІНФРАСТРУКТУРИ

Поняття IT-інфраструктури та її компоненти

У сучасному, динамічному та взаємопов'язаному бізнес-ландшафті, інформаційні технології перетворилися з допоміжних інструментів на невід'ємний стратегічний актив, що формує основу для більшості операційних процесів та інновацій. Ключовим елементом, що забезпечує безперебійне та ефективне функціонування цих технологій, є IT-інфраструктура. Вона являє собою комплексне поєднання апаратних засобів, програмного забезпечення, мережевих компонентів, а також супутніх служб та процесів, які спільно забезпечують функціонування інформаційних систем підприємства. Надійність та стабільність її роботи є абсолютно критично важливою для підтримання бізнес-безперервності, досягнення високої продуктивності та забезпечення надійного захисту цінних даних. Це стосується як традиційних розгортань у фізичних центрах обробки даних, так і гнучких рішень у хмарних середовищах. Загальний успіх та конкурентоспроможність сучасного підприємства безпосередньо залежать від того, наскільки добре спроектована, керована та ефективно моніториться її базова IT-інфраструктура, оскільки будь-які збої або неефективності 13 можуть призвести до значних фінансових та репутаційних втрат.

13 Ефективне управління IT-інфраструктурою неможливе без глибокого розуміння її складових частин та їхньої взаємодії. Кожен компонент відіграє унікальну роль у забезпеченні загальної функціональності системи та вимагає специфічного підходу до моніторингу та адміністрування. Правильне визначення та класифікація цих компонентів дозволяє розробляти цільові рішення для їхнього відстеження та управління, забезпечуючи максимальну ефективність та мінімізуючи ризики.

Детальніше розглянемо основні компоненти сучасної IT-інфраструктури.

Апаратне забезпечення. Ця категорія охоплює фізичні складові, які формують основу для обчислювальних процесів та зберігання даних. Вона включає сервери, що надають обчислювальні потужності, 12 системи зберігання даних (SAN, NAS) 8 для збереження

інформації та мережеве обладнання (маршрутизатори, комутатори, брандмауери), що забезпечує зв'язок.

Програмне забезпечення. 8 Цей компонент включає всі програмні рішення, які керують апаратними ресурсами та забезпечують функціональність інфраструктури. 8 До нього належать операційні системи, програмне забезпечення для віртуалізації, проміжне ПЗ (middleware), 12 системи управління базами даних (СУБД) та прикладні бізнес-додатки.

Мережеві компоненти. Відповідають за безперебійний обмін інформацією та комунікацію між усіма елементами інфраструктури. Це охоплює ключові мережеві протоколи, які регулюють передачу даних, обрану мережеву топологію, а також різноманітні мережеві сервіси, такі як VPN та системи безпеки (IDS/IPS).

Сервіси. Це додаткові служби, що підтримують ефективне функціонування та управління IT-інфраструктурою протягом усього життєвого циклу. Серед них виділяють системи управління ідентифікацією та доступом (IAM), інструменти моніторингу та управління, системи резервного копіювання, а також комплексні рішення з кібербезпеки (антивіруси, SIEM-системи).

Ефективний моніторинг та адміністрування всіх цих компонентів IT-інфраструктури є не просто технічною необхідністю, а стратегічним імперативом для сучасного бізнесу. Комплексний підхід до їхнього управління забезпечує стабільність функціонування, високу продуктивність та надійну безпеку всієї IT-екосистеми підприємства, що, у свою чергу, безпосередньо впливає на його конкурентоспроможність та загальний успіх у динамічному ринковому середовищі, де технологічні переваги та стійкість до зовнішніх викликів стають дедалі важливішими факторами успіху.

1.2 Аналіз методів і засобів розробки систем моніторингу та адміністрування IT-інфраструктури з використанням Python.

Розробка ефективної та надійної системи моніторингу та адміністрування IT-інфраструктури вимагає ретельного аналізу та обґрунтованого вибору методів збору даних, а також відповідних засобів реалізації. У сучасному ландшафті програмної розробки Python виділяється як мова, що надає широкі можливості для створення подібних систем, 19 завдяки своїй гнучкості, багатій екосистемі бібліотек та відносно низькому порогу входу, що дозволяє швидко прототипувати та розгортати рішення.

Центральним аспектом будь-якої системи моніторингу є механізм збору даних. Існують декілька основних підходів до цього процесу, кожен з яких має свої переваги та недоліки, і вибір яких залежить від специфіки контрольованої інфраструктури та необхідної глибини моніторингу.

Агентний моніторинг. Цей метод передбачає встановлення спеціалізованих програмних модулів, або агентів, безпосередньо на кожному контрольованому сервері чи пристрої. Агенти відповідають за збір локальних системних метрик (таких як завантаження процесора, використання оперативної пам'яті, дискового простору, обсяг мережевого трафіку, стан запущених процесів та служб) та лог-файлів. Перевага агентного підходу, особливо при реалізації на Python з використанням крос-платформних бібліотек на кшталт psutil, полягає в його здатності надавати глибокий та детальний рівень інформації про стан системи, а також у можливості моніторити різноманітні операційні системи за допомогою єдиної, уніфікованої кодової бази. Для збору специфічних даних з Windows-систем додатково може бути використана бібліотека wmi, що забезпечує доступ до розширеного інструментарію управління Windows.

Безагентний моніторинг (Agentless). На відміну від агентного, цей метод не вимагає встановлення додаткового програмного забезпечення на цільових системах. Збір інформації здійснюється за допомогою стандартних мережевих протоколів, таких як SNMP (Simple Network Management Protocol), WMI (віддалено для Windows), SSH (Secure Shell) або ICMP (Internet Control Message Protocol). Реалізація без-агентного моніторингу на Python можлива за допомогою бібліотек типу paramiko для безпечних SSH-з'єднань, або pysnmp 4 для роботи з SNMP. Цей підхід є зручним для великих інфраструктур, де розгортання та управління агентами може бути складним або небажаним, проте він може мати певні обмеження щодо глибини та гранулярності зібраних даних.

Збір та аналіз логів. Аналіз системних та прикладних лог-файлів є надзвичайно важливим для виявлення подій, помилок, аномалій та потенційних проблем у роботі IT-інфраструктури. Python надає гнучкі та потужні засоби для парсингу, обробки та передачі логів до централізованих систем, використовуючи стандартні бібліотеки 4 для роботи з файловими системами та мережею. Це дозволяє здійснювати проактивний моніторинг та оперативно реагувати на критичні події.

Щодо засобів розробки, 4 мова програмування Python обрана як основна для даної системи завдяки її простоті синтаксису, широкій підтримці спільноти, а також багатій екосистемі спеціалізованих бібліотек, що значно прискорює процес розробки. Серед ключових бібліотек, які є основою для реалізації системи моніторингу та адміністрування, варто виділити наступні:

Бібліотеки для збору даних. Бібліотека psutil забезпечує крос-платформний інтерфейс для отримання різноманітної системної інформації, що є фундаментальним для роботи агентів. Для деталізованого моніторингу Windows-систем використовується wmi.

Бібліотеки для мережевої взаємодії. Модуль socket надає базові функції для

низькорівневої роботи з мережевими сокетами, що дозволяє створювати ефективні клієнт-серверні взаємодії. Бібліотека requests спрощує виконання HTTP/HTTPS-запитів, що може бути використано для взаємодії з веб-сервісами або API. Для забезпечення безпечного віддаленого адміністрування та виконання команд на вузлах використовується paramiko, що реалізує протокол SSHv2.

Бази даних. На початкових етапах розробки та для невеликих інфраструктур оптимальним вибором є sqlite3 – легка вбудована реляційна СУБД, яка не потребує окремого серверу. Для спрощення взаємодії з базою даних та забезпечення гнучкості до різних СУБД (PostgreSQL, MySQL у майбутньому) застосовується SQLAlchemy – потужний ORM (Object-Relational Mapper) для Python.

Інструменти для інтерфейсу користувача: Залежно від обраного підходу до реалізації інтерфейсу (десктопний або веб-базований), можуть бути використані Tkinter або PyQt для створення десктопного графічного інтерфейсу, що забезпечить високу швидкість відгуку та незалежність від браузера. Якщо буде обрано веб-інтерфейс, то Flask може слугувати як легкий та гнучкий веб-фреймворк для побудови RESTful API серверної частини, а для фронтенду застосовуватимуться стандартні веб-технології: HTML, CSS та JavaScript.

Окрім перелічених, важливими є також інструменти для управління розробкою, такі як Git та платформи на його основі (GitHub/GitLab), що є незамінними для контролю версій та спільної роботи над проектом. Формат JSON буде використовуватися як стандарт для обміну даними між різними компонентами системи завдяки його легкості та універсальності. Вбудований у Python модуль logging забезпечить ефективне журналювання подій системи для налагодження та аудиту.

Вибір конкретних методів та засобів реалізації, орієнтованих на Python, забезпечує значну гнучкість у адаптації системи до різноманітних вимог IT-інфраструктури, дозволяючи створювати масштабовані, надійні та ефективні рішення для моніторингу та адміністрування.

1.3 Дослідження специфічних вимог бізнесу до систем моніторингу та адміністрування IT-інфраструктури.

Етап вибору технологій та інструментів є одним з найважливіших у життєвому циклі розробки програмного забезпечення. 15 Саме на цьому етапі закладається фундамент майбутнього продукту, визначаються його ключові характеристики, такі як продуктивність, безпека, масштабованість, зручність використання та вартість розробки. Для проекту створення комплексної системи моніторингу та адміністрування IT-інфраструктури, яка має забезпечити ефективне управління інформаційними ресурсами бізнесу, ретельний аналіз доступних технологій та обґрунтований вибір є

абсолютно необхідними для досягнення поставлених цілей. Враховуючи специфічні вимоги до системи, наявні знання та досвід команди розробників, а також багату екосистему інструментів, доступних для розробки на Python, 2 саме ця мова програмування була обрана як основна для реалізації проекту.

Вибір Python як основної мови програмування для розробки системи моніторингу та адміністрування IT-інфраструктури зумовлений цілим рядом вагомих переваг, які роблять її особливо привабливою для вирішення завдань такого класу.

Однією з фундаментальних переваг Python є її елегантний та лаконічний синтаксис, який значно підвищує читабельність коду. Це не лише полегшує процес написання та розуміння коду для розробників, але й спрощує подальшу підтримку, налагодження та модифікацію системи. Читабельний код також сприяє ефективній співпраці в команді, оскільки різні розробники можуть швидше орієнтуватися в кодовій базі.

2 Python має одну з найбільших та найактивніших спільнот розробників у світі. Це означає доступ до величезної кількості документації, навчальних ресурсів, готових рішень на форумах та у блогах, а також активну підтримку у випадку виникнення проблем. Велика спільнота також сприяє швидкому розвитку мови та її екосистеми, а також оперативному виявленню та виправленню помилок.

Однією з ключових переваг Python є наявність величезної кількості високоякісних сторонніх бібліотек та фреймворків, які значно спрощують та прискорюють розробку складних систем. Для завдань моніторингу та адміністрування IT-інфраструктури існує безліч спеціалізованих інструментів, які надають готовий функціонал для взаємодії з операційною системою, мережею, різним обладнанням та програмним забезпеченням.

Python є крос-платформною мовою програмування, що дозволяє запускати написаний код на різних операційних системах (Windows, Linux, macOS) без необхідності значних змін. Це є критично важливим для системи моніторингу та адміністрування, яка повинна мати можливість працювати в різноманітних IT-середовищах, що можуть включати гетерогенний набір операційних систем.

Python підтримує різні парадигми програмування (об'єктно-орієнтоване, функціональне, імперативне), що надає розробникам гнучкість у виборі найбільш підходящого підходу для вирішення конкретних завдань. Крім того, Python має потужні механізми для інтеграції з кодом, написаним іншими мовами програмування (наприклад, C, C++), 6 що може бути корисним для оптимізації критичних до продуктивності частин системи або для взаємодії з існуючими низькорівневими бібліотеками.

Завдяки простому синтаксису, великій кількості готових бібліотек та високому рівню

абстракції, Python 6 дозволяє значно скоротити час, необхідний для розробки програмного забезпечення. Це дає змогу швидше отримувати працездатні прототипи та ітеративно розвивати систему, враховуючи потреби замовника та користувачів.

1.4 Огляд існуючих систем моніторингу та адміністрування.

На сучасному ринку представлено значну кількість комерційних та відкритих рішень 3 для моніторингу та адміністрування IT-інфраструктури, кожне з яких має свої переваги та недоліки, а також орієнтоване на певні розміри та типи організацій. Вибір відповідної системи є критично важливим, оскільки вона має інтегруватися в існуючі процеси, забезпечувати необхідний рівень функціональності та бути масштабованою відповідно до потреб зростаючої інфраструктури. Аналіз провідних рішень дозволяє виявити найкращі практики, стандарти функціональності та підходи до архітектури, які можуть бути адаптовані або слугувати орієнтиром для розробки власної системи.

Серед широкого спектру існуючих систем моніторингу та адміністрування, варто виділити декілька ключових гравців, які є загальновизнаними в галузі:

Zabbix. Це універсальна відкрита система моніторингу, яка здатна відстежувати велику кількість мережевих параметрів, серверів та мережевого обладнання. Вона пропонує широкі можливості для налаштування, включаючи гнучку систему шаблонів, виявлення аномалій, а також розвинені механізми оповіщення та візуалізації даних. 3 Zabbix підтримує як агентний, так і безагентний моніторинг, що робить його дуже гнучким для різноманітних IT-середовищ.

Nagios Core / Nagios XI. Nagios Core є відкритою, дуже гнучкою системою моніторингу, яка широко використовується для перевірки доступності хостів та сервісів, а також для моніторингу ресурсів. Nagios XI – це комерційна версія, що додає розширені можливості, такі як веб-інтерфейс, звіти та автоматизацію. Вона відома своєю стабільністю та здатністю до глибокого моніторингу.

Prometheus. Це сучасна система моніторингу та оповіщення з відкритим вихідним кодом, яка орієнтована на моніторинг динамічних хмарних та контейнеризованих середовищ. Її архітектура базується на моделі витягування метрик (pull model), має потужну мову запитів PromQL та інтегрується з візуалізацією за допомогою Grafana. Prometheus чудово підходить для мікросервісної архітектури.

Grafana. Хоча Grafana сама по собі не є повноцінною системою моніторингу, вона є потужним інструментом для візуалізації даних, отриманих від інших джерел моніторингу (наприклад, Prometheus, Zabbix, Elasticsearch, баз даних). Вона дозволяє створювати інтерактивні дашборди, графіки та алерти, роблячи дані зрозумілими та доступними для аналізу.

PRTG Network Monitor. Це комерційне рішення від Paessler AG, яке надає комплексний моніторинг мереж, серверів, додатків та віртуальних середовищ. PRTG є відносно простим у налаштуванні та експлуатації, пропонує широкий набір сенсорів та має інтуїтивно зрозумілий інтерфейс, що робить його привабливим для середніх та великих підприємств.

Порівняльний аналіз цих систем (Таблиця 1.1) дозволяє виявити їхні ключові відмінності у функціональності, моделях ліцензування, легкості розгортання, масштабованості та підтримці.

Таблиця 1.1 - Порівняльний аналіз існуючих систем моніторингу

Характеристика

Zabbix

Nagios Core / XI

Prometheus

PRTG Network Monitor

Тип ліцензії

Open Source

Core: OS; XI: Комерційна

Open Source

Комерційна

Тип моніторингу

Агентний, без агент-ний

Агентний, безагентний

Pull-модель, експортери

Безагентний (сенсори)

Масштабо-ваність

Висока

Висока

Дуже висока

Середня / Висока

Візуалізація

Вбудовані, Grafana

Базові, плагіни

PromQL, Grafana

Вбудовані дашборди

Автоматизація

Автовиявлення, віддалені команди

Обмежена (Core), розширена (XI)

Service Discovery

Розширена

Переваги

Гнучкість, комплекс-ність

Стабільність надійність

Для хмари, PromQL

Простота, все в одному

Недоліки

Високий поріг входу

Складність налаштуван-ня

Для динамічних IT, без вбудованих звітів

Комерційна, ціна

Цей аналіз є важливим для визначення того, які аспекти цих рішень можуть бути інтегровані або надихнути на розробку власної, спеціалізованої системи, що відповідатиме унікальним вимогам проекту. Він показує, що хоча на ринку існують потужні комплексні рішення, такі як Zabbix та Prometheus, вони часто мають високий поріг входження та є надлишковими для завдань швидкої локальної діагностики. Це відкриває нішу для більш простих, вузькоспеціалізованих утиліт, орієнтованих на зручність використання, що і є метою даної роботи.

Результати проведеного дослідження будуть враховані при розробці системи моніторингу та адміністрування IT-інфраструктури для бізнесу в межах даної дипломної роботи.

ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ

2.1 Визначення функціональних та нефункціональних вимог до системи

На етапі проектування будь-якої програмної системи, особливо такої, що має критичне значення для функціонування IT-інфраструктури, першочерговим завданням є чітке та вичерпне визначення функціональних та нефункціональних вимог. Цей процес є фундаментальним, оскільки він закладає основу для всіх подальших етапів розробки: архітектурного проектування, вибору технологій, реалізації та тестування. Недостатньо чітко сформульовані вимоги 11 можуть призвести до розробки системи, що не відповідає потребам користувачів, перевищує бюджет або має проблеми з продуктивністю та надійністю. Розуміння як "що" система має робити (функціональні вимоги), так і "як добре" вона має це робити (нефункціональні вимоги), є запорукою успішного проекту.

Функціональні вимоги описують конкретні дії та можливості, які система повинна надавати кінцевому користувачеві або іншим системам. Вони визначають основне призначення та функціонал рішення. Для системи моніторингу та адміністрування IT-інфраструктури ключовими функціональними вимогами є:

Збір даних моніторингу. 11 Система повинна забезпечувати можливість збору широкого спектру метрик з контрольованих вузлів, включаючи використання процесора, оперативної пам'яті, дискового простору, мережевого трафіку, а також дані про запущені процеси та сервіси. Збір має здійснюватися як з операційних систем Linux, так і Windows.

Зберігання та агрегація даних. Зібрані метрики та лог-дані повинні бути ефективно збережені у базі даних, з можливістю агрегації за різними періодами часу для подальшого аналізу та візуалізації.

Аналіз даних та виявлення аномалій. Система має виконувати аналіз отриманих

метрик, порівнюючи їх зі встановленими пороговими значеннями. У разі перевищення порогів або виявлення аномальної поведінки, система повинна генерувати події та оповіщення.

Система оповіщень. Повинна бути реалізована гнучка система сповіщень, яка інформує адміністраторів про виявлені проблеми через різні канали (наприклад, електронна пошта, Telegram, Slack). Має бути можливість налаштування рівнів критичності та ескалації сповіщень.Віддалене адміністрування. Система повинна надавати функціональність для безпечного віддаленого виконання команд на контрольованих серверах, що дозволить оперативно реагувати на інциденти або автоматизувати рутинні завдання (наприклад, перезапуск сервісів).

Візуалізація даних. Інформація про стан інфраструктури, метрики та події повинна бути представлена в інтуїтивно зрозумілому графічному вигляді (дашборди, графіки), що забезпечує швидкий огляд та детальний аналіз.

Управління конфігурацією. Система має дозволяти адміністраторам додавати/видаляти контрольовані вузли, налаштовувати параметри моніторингу, порогові значення та правила оповіщень через централізований інтерфейс.

Управління користувачами та правами доступу. Повинна бути реалізована система аутентифікації та авторизації, що дозволяє керувати обліковими записами користувачів та їхніми ролями, забезпечуючи доступ до відповідної функціональності системи.

Нефункціональні вимоги описують 7 якісні характеристики системи, такі як продуктивність, надійність, безпека, зручність використання та масштабованість. Вони не стосуються безпосередньо функціоналу, але є критично важливими для загального успіху та прийнятності системи. Для даної системи моніторингу вони включають такі фактори як надійність, продуктивність та зручність використання.

Надійність системи характеризується тим, що вона повинна працювати стабільно та без збоїв 24/7, забезпечуючи безперервний моніторинг інфраструктури. Можливість відновлення після збоїв та мінімізація втрати даних є обов'язковими.

Продуктивність системи передбачає забезпечення збору, обробки та зберігання великих обсягів даних моніторингу без значних затримок. Інтерфейс користувача має бути швидкодіючим та реагувати на дії користувача без зависань. Масштабованість. Архітектура системи повинна дозволяти легке розширення для моніторингу зростаючої кількості вузлів та збору більшого обсягу даних без значної реорганізації або падіння продуктивності. Безпека. Це включає захист даних під час передачі та зберігання, надійні механізми аутентифікації та авторизації користувачів, а також захист від несанкціонованого доступу та атак. Всі віддалені команди мають виконуватися через

захищені протоколи.

Зручність використання (Usability) характеризується тим, що інтерфейс системи повинен бути інтуїтивно зрозумілим, легким у навігації та налаштуванні для адміністраторів, що дозволить ефективно використовувати її функціонал. Адаптивність/Гнучкість. Система повинна бути гнучкою до змін у складі інфраструктури та нових вимог, дозволяючи легке додавання нових типів моніторингу або інтеграцію з іншими системами. Сумісність. Система повинна бути сумісною з основними операційними системами (Linux, Windows) та стандартними мережевими протоколами. Чітке визначення цих вимог є основою для подальшого проектування архітектури системи, вибору технологій та розробки, гарантуючи, що кінцевий продукт буде повністю відповідати потребам бізнесу та IT-персоналу. На основі аналізу предметної області та потенційних потреб користувачів було сформовано перелік ключових вимог до програмного продукту. Усі функціональні та нефункціональні вимоги були систематизовані та представлені у таблиці 2.1.

Таблиця 2.1 – Функціональні та нефункціональні вимоги до системи

Тип вимоги

Вимога

Опис

Функціональні

Збір даних

Збір метрик (ЦП, ОЗП, диск, мережа) з вузлів Windows/Linux

Візуалізація даних

Представлення даних у вигляді графіків та дашбордів у реальному часі

Адміністрування процесів

Перегляд та примусове завершення активних процесів

Діагностика системи

Інструменти для стрес-тестування та мережевої діагностики

Перегляд конфігурації

Відображення даних про апаратне забезпечення та встановлене ПЗ

Нефункціональні

Продуктивність

Швидкий відгук інтерфейсу; мінімальне навантаження на систему (<5% ЦП)

Надійність

Стабільна робота 24/7, коректна обробка системних помилок

Зручність використання

Інтуїтивно зрозумілий інтерфейс для технічного спеціаліста

Сумісність

Підтримка роботи на ОС Windows 10/11 та основних дистрибутивах Linux

Безпека

Підтвердження користувачем дій, що змінюють стан системи

Перелік вимог є фундаментальним етапом проектування, що визначає як функціональне наповнення системи, так і якісні характеристики її роботи. Ключовий акцент зроблено на нефункціональних аспектах, таких як висока продуктивність та надійність, оскільки для інструменту моніторингу критично важливо працювати стабільно та з мінімальним впливом на саму систему. Водночас, функціональні вимоги охоплюють широкий спектр завдань: від пасивного збору метрик до активних інструментів адміністрування, що перетворює продукт на комплексну діагностичну утиліту. Таким чином, ця сукупність вимог формує детальне технічне завдання, що стало основою для проектування архітектури, здатної гармонійно поєднати широкий функціонал з високими стандартами стабільності.

2.2Розробка архітектури системи.

Ефективна система моніторингу та адміністрування IT-інфраструктури повинна забезпечувати безперервний збір даних про стан усіх критично важливих компонентів, надавати зручні інструменти для візуалізації та аналізу цих даних, своєчасно інформувати адміністраторів про виявлені проблеми та, за потреби, надавати засоби для оперативного втручання та усунення несправностей. Реалізація цих основних функцій вимагає комплексного підходу до розробки кожного компонента системи.

Збір даних моніторингу є першочерговим завданням. На кожному вузлі, який підлягає моніторингу, встановлюються спеціалізовані агенти, розроблені на мові

програмування Python. Вибір Python зумовлений його крос-платформністю, що дозволяє використовувати єдину кодову базу для моніторингу різноманітних операційних систем. Агенти використовують потужні бібліотеки, такі як psutil, яка надає уніфікований доступ до інформації про процеси, використання системних ресурсів (ЦП, пам'ять, дисковий простір, мережеві інтерфейси) та іншої системної статистики на різних платформах. Для отримання більш глибоких та специфічних даних на системах Windows застосовується бібліотека wmi, яка взаємодіє з Windows Management Instrumentation.Процес збору даних відбувається періодично, згідно з налаштованим інтервалом, що дозволяє 10 отримувати актуальну інформацію про стан системи в динаміці. Зібрані метрики формуються у структуровані повідомлення (наприклад, у форматі JSON) та передаються на центральний сервер. Для забезпечення надійної передачі даних використовуються стандартні мережеві протоколи, такі як TCP/IP (за допомогою бібліотеки socket) або HTTP/HTTPS (з використанням бібліотеки requests). У випадку тимчасових мережевих проблем агенти можуть мати механізми буферизації даних та повторної спроби їхньої передачі після відновлення з'єднання.

Відображення даних моніторингу є ключовим 10 для забезпечення ефективного аналізу стану IT-інфраструктури. Центральний сервер отримує та обробляє дані від агентів, а потім надає їх для візуалізації через інтерфейс користувача. Інтерфейс може бути реалізований як графічний додаток на Python з використанням таких бібліотек, як Tkinter або PyQt, що дозволяють створювати інтерактивні дашборди та елементи керування.

У інтерфейсі користувача ключові метрики відображаються у вигляді наочних графіків, діаграм, таблиць та індикаторів. Наприклад, використання ЦП та оперативної пам'яті можуть бути представлені у вигляді графіків часових рядів, завантаження дискової підсистеми – у вигляді кругових діаграм, а мережевий трафік – у вигляді лінійних графіків. Кольорова індикація використовується для швидкого виявлення критичних значень або потенційних проблем (наприклад, червоний колір може сигналізувати про перевищення порогових значень). Адміністратори 18 мають можливість переглядати детальну інформацію про кожен контрольований вузол, аналізувати історичні дані за різні періоди часу, а також налаштовувати відображення найбільш важливих для них метрик.

Система сповіщень є невід'ємною частиною системи моніторингу, яка забезпечує своєчасне інформування адміністраторів про виявлені аномалії або проблеми. Адміністратори мають можливість визначати порогові значення для різних контрольованих метрик через інтерфейс користувача. Ці налаштування зберігаються на центральному сервері, можливо, у базі даних sqlite3. Центральний сервер постійно відстежує отримані від агентів дані та порівнює їх з встановленими порогами. У випадку виявлення порушення порогового значення генерується сповіщення. Кожне

сповіщення має свій рівень серйозності (наприклад, інформаційне, попередження, критичне), що допомагає адміністраторам пріоритезувати свою роботу. Сповіщення відображаються в спеціальному розділі інтерфейсу користувача та можуть надсилатися адміністраторам 14 різними каналами зв'язку, такими як електронна пошта (з використанням Python-бібліотек для роботи з поштою) або системні повідомлення.

Функціональність віддаленого виконання команд надає адміністраторам можливість оперативно реагувати на виявлені інциденти без необхідності безпосереднього доступу до проблемних вузлів. Для реалізації цієї функції використовується бібліотека paramiko, яка забезпечує встановлення захищених SSH-з'єднань з віддаленими вузлами. Доступ до цієї функціональності суворо контролюється за допомогою надійних механізмів аутентифікації та авторизації. Адміністраторам надається можливість вводити та виконувати системні команди на вибраних віддалених вузлах безпосередньо через інтерфейс системи моніторингу. Всі дії, пов'язані з віддаленим виконанням команд, ретельно логуються для забезпечення аудиту та безпеки. Для реалізації поставлених вимог було спроектовано модульну архітектуру додатку, що базується на об'єктно-орієнтованому підході. Основні програмні компоненти, їх функціональне призначення та технологічна основа описані в таблиці 2.2

Таблиця 2.2 – Архітектурні компоненти програмного продукту

Назва компонента (класу)

Основне призначення

Використані бібліотеки/модулі

SystemInfoDashboard

Головний клас програми, створення основного вікна та навігації.

CustomTkinter, psutil

ProcessManagerWindow

Відображення та керування системними процесами (перегляд, сортування, завершення).

psutil, tkinter.ttk.Treeview

InstalledProgramsWindow

Перегляд та керування встановленим ПЗ (для Windows).

winreg, subprocess, tkinter.ttk.Treeview

StressTestWindow

Проведення навантажувального тестування ЦП, ГП, ОЗП та дисків.

multiprocessing, matplotlib, threading

(Фонові потоки/процеси)

Асинхронний збір даних та виконання тривалих завдань без блокування GUI.

threading, queue

(Модулі інформації)

Створення дочірніх вікон з детальною інформацією про компоненти системи.

matplotlib, wmi, cpuinfo

На блок-схемі 2.1 представлена високорівнева архітектура розробленого програмного продукту. В основі системи лежить головне графічне вікно (GUI), яке надає доступ до трьох основних функціональних блоків: модуля моніторингу, модуля управління та модуля тестування. Кожен з цих блоків, у свою чергу, декомпозовано на конкретні підмодулі: моніторинг ресурсів (CPU/RAM/GPU, диски), управління системою (менеджер процесів, програмне забезпечення) та проведення діагностики (стрес-тести, мережеві тести). Схема також ілюструє зв'язок між функціональними компонентами та ключовими технологіями, що лежать в їх основі, такими як бібліотеки psutil для збору даних, matplotlib для візуалізації та multiprocessing для реалізації навантажувальних тестів. Така модульна структура забезпечує логічний поділ функціоналу та спрощує подальшу розробку й підтримку системи.

Рисунок 2.1 структури програми

2.3 Вибір технологій та інструментів для реалізації.

Вибір технологічного стеку є фундаментальним етапом проектування, що визначає архітектурні можливості, швидкість розробки та кінцеву функціональність програмного продукту. Для реалізації системи моніторингу було обрано набір інструментів, що найкраще відповідає поставленим вимогам: створення нативного десктопного додатку з високою продуктивністю, прямим доступом до системних ресурсів та сучасним, інформативним графічним інтерфейсом. Рішення приймалися на основі аналізу переваг та недоліків потенційних технологій у контексті специфічних завдань проєкту.

Основним критерієм при виборі мови програмування була швидкість розробки та наявність готових інструментів для системної взаємодії. Мову програмування Python було обрано як оптимальний варіант, оскільки вона поєднує в собі простий та виразний синтаксис з величезною екосистемою сторонніх бібліотек. На відміну від компільованих мов, таких як C++ або Java, які потребують більше коду для виконання аналогічних завдань та мають довший цикл компіляції, Python дозволяє значно швидше прототипувати та ітерувати функціонал. Для завдань системного моніторингу це є вирішальною перевагою, оскільки такі бібліотеки, як psutil та wmi, вже надають високорівневий доступ до низькорівневих функцій операційної системи. Це в інших мовах потребувало б написання значної кількості коду для взаємодії з системними API, що суттєво збільшило б складність та час розробки.

Графічний інтерфейс користувача реалізовано за допомогою Tkinter та CustomTkinter. При виборі фреймворку для графічного інтерфейсу розглядалися такі альтернативи, як PyQt6 та Kivy. PyQt є надзвичайно потужним та функціональним фреймворком, проте його використання ускладнюється подвійною ліцензією (GPL/Commercial), що накладає обмеження на розповсюдження продукту, а також вимагає встановлення значної кількості залежностей. Kivy, у свою чергу, орієнтований переважно на створення крос-платформних мобільних додатків, що не було пріоритетом для даного проєкту. У зв'язку з цим, було прийнято рішення використати Tkinter як базову бібліотеку. Її головна перевага полягає в тому, що вона входить до стандартної бібліотеки Python, а отже, не вимагає жодних додаткових інсталяцій, що гарантує легкість розповсюдження та запуску додатку на будь-якій системі з встановленим Python. Проте, візуальний стиль стандартних віджетів Tkinter є застарілим. Для вирішення цієї проблеми було інтегровано бібліотеку CustomTkinter. Вона є сучасною надбудовою над Tkinter, яка малює віджети з нуля, надаючи їм сучасний вигляд, підтримку тем оформлення (світла/темна) та широкі можливості для кастомізації. Такий гібридний підхід дозволив поєднати надійність, простоту і портативність Tkinter з сучасним та естетично привабливим дизайном, який забезпечує CustomTkinter.

Центральним компонентом системи є модуль збору даних. Бібліотека psutil (Python System and Process Utilities) стала безальтернативним вибором для цього завдання. Вона виступає потужним шаром абстракції, що приховує складність взаємодії з різними операційними системами. Без неї для отримання даних на Windows довелося б працювати з громіздким WinAPI через бібліотеку ctypes, а на Linux — парсити текстові файли у віртуальній файловій системі /proc. psutil уніфікує ці підходи, надаючи простий та єдиний API, де виклик psutil.cpu_percent() однаково працює на всіх платформах. Для отримання розширеної інформації, специфічної для ОС Windows, як-от дані про виробника материнської плати, версію BIOS чи унікальні атрибути накопичувачів, було додатково інтегровано бібліотеку wmi. Вона надає зручний об'єктно-орієнтований інтерфейс для роботи з Windows Management Instrumentation, дозволяючи

виконувати глибоку діагностику апаратного забезпечення.

Візуалізація даних та паралельні обчислення реалізовано за допомогою бібліотек matplotlib та multiprocessing. Для візуалізації даних у вигляді графіків було обрано бібліотеку matplotlib. Вона є стандартом де-факто для наукової візуалізації в Python і, що важливо, має спеціальний модуль backend_tkagg, який є "мостом" між логікою matplotlib та графічною системою Tkinter. Це дозволило вбудовувати складні графіки безпосередньо у вікна додатку та оновлювати їх у реальному часі, створюючи по-справжньому інтерактивний досвід для користувача.Для реалізації ресурсоємних завдань, таких як стрес-тести, було використано стандартний модуль multiprocessing. Вибір на користь мультипроцесингу, а не багатопотоковості (threading), був зумовлений наявністю в Python механізму Global Interpreter Lock (GIL). GIL — це м'ютекс, який захищає доступ до об'єктів Python, запобігаючи одночасному виконанню байт-коду Python кількома потоками в рамках одного процесу. Це робить threading неефективним для задач, що інтенсивно навантажують процесор. Модуль multiprocessing обходить це обмеження, створюючи окремі процеси, кожен з яких має власний інтерпретатор Python та пам'ять. Це єдиний надійний спосіб досягти справжнього паралелізму та повністю завантажити всі ядра процесора для проведення коректного стрес-тесту, не блокуючи при цьому основний потік графічного інтерфейсу.

2.4 Проектування інтерфейсу та користувацького досвіду

Проектування 16 інтерфейсу користувача (UI) та користувацького досвіду (UX) є критично важливим етапом у розробці системи моніторингу. Оскільки програма оперує великими обсягами технічних даних, головною метою дизайну було створення інтуїтивно зрозумілого, ефективного та неперевантаженого середовища для системного адміністратора. Ключове завдання — знизити когнітивне навантаження на користувача, дозволяючи йому швидко оцінювати стан IT-інфраструктури та оперативно приймати рішення.В основі проектування лежить принцип інформаційної ієрархії, що структурує дані за рівнями деталізації. Головний дашборд програми надає користувачеві загальний огляд ключових метрик, таких як миттєве завантаження центрального процесора та оперативної пам'яті. Це дозволяє одним поглядом оцінити загальний стан системи. Для більш глибокого аналізу передбачені спеціалізовані вікна, куди можна перейти з головного дашборду. Такий підхід дозволяє уникнути перевантаження основного екрану та надати деталі лише за запитом користувача.Для миттєвої ідентифікації стану системи було впроваджено кольорове кодування в інтерактивних елементах. Наприклад, віджети-вимірювачі змінюють свій колір із зеленого (нормальний стан) на жовтий (підвищене навантаження) та червоний (критичний стан), що дозволяє візуально сигналізувати про потенційні проблеми ще до того, як користувач побачить точні цифрові значення.Весь інтерфейс спроектовано за принципом модульності та консистентності. Програма розбита на логічні модулі, доступ

до яких здійснюється через сітку інтерактивних карток на головному екрані. Кожне інформаційне вікно, чи то аналіз процесора, чи моніторинг пам'яті, дотримується єдиного стилю та структури: у верхній частині розташовуються графіки для візуалізації даних у динаміці, а в нижній – таблиці зі статичними характеристиками. Така єдність дизайну робить поведінку програми передбачуваною та значно полегшує її освоєння.Для забезпечення ефективної взаємодії всі графіки та показники оновлюються в реальному часі, надаючи користувачеві актуальну інформацію без необхідності ручного оновлення. Елементи керування, такі як кнопки та перемикачі, мають візуальний відгук при наведенні та натисканні, що підтверджує виконання дії та покращує загальний користувацький досвід. Вибір сімейства шрифтів "Segoe UI" та мінімалістичної іконографіки спрямований на забезпечення високої читабельності та швидкого розпізнавання функцій.

Застосування цих фундаментальних принципів дизайну не було хаотичним, а мало системний характер, де кожне вікно та елемент керування проектувалися з урахуванням кінцевого користувача. Для наочної демонстрації того, як саме ці теоретичні підходи були втілені в практичні рішення в рамках розробленого додатку, ключові дизайнерські концепції та відповідні їм приклади реалізації зведено в таблицю 2.3.

Таблиця 2.3 – Застосування принципів UI/UX дизайну в системі

Принцип дизайну

Мета

Приклад реалізації в проєкті

Інформаційна ієрархія

Подання даних від загального до конкретного.

Дашборд — загальний стан; окремі вікна — деталі.

Модульність та консистентність

Єдиний та передбачуваний вигляд усіх вікон.

Всі модулі мають схожу структуру (графік + таблиця).

Кольорове кодування

Швидка візуальна оцінка стану системи.

Колір індикаторів (зелений/жовтий/червоний) сигналізує про навантаження.

Інтерактивний зворотний зв'язок

Створити чутливий та "живий" інтерфейс.

Анімація віджетів; візуальний відгук кнопок при наведенні.

Як видно з таблиці, кожне прийняте дизайнерське рішення мало на меті вирішення конкретного завдання користувацького досвіду — від зниження когнітивного навантаження до підвищення інтерактивності. Такий структурований підхід до проектування інтерфейсу є ключовим фактором у створенні не просто функціонального, а й по-справжньому зручного та ефективного програмного продукту, готового до використання технічними спеціалістами.

РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ТА АДМІНІСТРУВАННЯ ІТ-ІНФРАСТРУКТУРИ

3.1. Розробка архітектури та головного вікна програми

Практична реалізація системи моніторингу та адміністрування базується на проектних рішеннях, розроблених у другому розділі. Система створена у вигляді десктопного додатку з графічним інтерфейсом користувача (GUI) на 1 мові програмування Python. Такий 1 підхід забезпечує високу швидкість роботи, 1 прямий доступ до системних ресурсів та незалежність від веббраузера. Для побудови інтерфейсу було обрано бібліотеку CustomTkinter, яка є сучасним розширенням стандартної бібліотеки Tkinter і дозволяє створювати естетично привабливі та кастомізовані віджети.

В основі програми лежить головний клас SystemInfoDashboard, який відповідає за створення та керування головним вікном. При ініціалізації цього класу виконуються першочергові налаштування: встановлюються заголовок та початковий розмір вікна, а також створюється централізований словник шрифтів для забезпечення візуальної єдності всього додатку.

Архітектура програми є модульною – конструктор послідовно викликає низку внутрішніх методів, кожен з яких відповідає за створення окремого елемента інтерфейсу.Представлений на рисунку 3.1 інтерфейс слугує центральним хабом 1 для доступу до всіх функцій програми. Ключовим елементом навігації є сітка інтерактивних карток в основній робочій області, формування якої відбувається у методі _create_main_content_area. Для гнучкості було застосовано підхід, керований даними: вся інформація про картки (назва, іконка, функція-обробник) зберігається в єдиній структурі даних, по якій ітерує метод, що генерує сітку. Важливим аспектом є прив'язка обробників подій до кожної картки, що робить інтерфейс інтерактивним та чутливим до

дій користувача. Такий дизайн дозволяє швидко отримати доступ до будь-якого з одинадцяти основних модулів програми.

Рисунок 3.1 – Загальний вигляд головного вікна програм

3.2. Реалізація модулів збору та відображення системної інформації

Після створення основного каркасу програми було реалізовано ключовий функціонал — модулі для збору, обробки та візуалізації детальної інформації про основні компоненти системи. Кожен модуль представлений окремим вікном, яке відкривається при натисканні на відповідну картку на головному дашборді. Архітектура цих вікон є уніфікованою, проте кожен модуль має унікальні елементи візуалізації, що відповідають типу даних, які він представляє.Розглянемо реалізацію на прикладі модуля інформації про центральний процесор (ЦП). При його виклику створюється нове дочірнє вікно, де інформація логічно розділена на статичну (модель, архітектура, кількість ядер) та динамічну (поточне завантаження, частота). Для того, щоб постійний збір динамічних даних не блокував графічний інтерфейс, його було винесено в окремий фоновий потік за допомогою модуля threading. Цей потік з інтервалом в одну секунду опитує систему через psutil, отримує актуальні дані та безпечно передає їх до основного потоку GUI через об'єкт черги (queue.Queue). Основний потік, отримавши нові дані, оновлює відповідні текстові поля та плавно анімує віджети CTkProgressBar, що візуалізують завантаження кожного ядра.Аналогічний підхід до збору даних використовується і в інших модулях, однак для візуалізації історії використання ресурсів було застосовано більш потужний інструмент — бібліотеку matplotlib. Вона дозволяє створювати складні наукові графіки та інтегрувати їх безпосередньо в інтерфейс Tkinter за допомогою спеціального віджета-контейнера FigureCanvasTkAgg. Програма зберігає список з останніх 60 значень метрики, постійно оновлюючи його, і перемальовує графік, створюючи ефект "живого", рухомого моніторингу.Таким чином, представлений на рисунку 3.2 модуль демонструє успішну інтеграцію сторонньої бібліотеки для візуалізації matplotlib у графічний інтерфейс, створений на базі CustomTkinter. Це дозволило реалізувати складний та інформативний елемент — динамічний графік — що є недосяжним при використанні базових віджетів. Застосований підхід до збору та відображення даних є універсальним і використовується також для реалізації моніторингу інших компонентів системи.

Рисунок 3.2 – Вигляд вікна моніторингу оперативної пам'яті

Найбільш складною частиною є реалізація динамічного оновлення даних. Для того, щоб постійний збір інформації про поточне завантаження та частоту не блокував графічний інтерфейс, було застосовано асинхронний підхід з використанням окремого фонового потоку. Детальний алгоритм цієї взаємодії показано на (рис. 3.3)

Рисунок 3.3 – Алгоритм асинхронного збору системних метрик

Як ілюструє схема, процес розділено на два паралельні потоки. Перший, фоновий потік-виробник, з фіксованим інтервалом збирає дані за допомогою `psutil` і безпечно передає їх у чергу (`Queue`). Другий, головний потік GUI, виступає споживачем: за допомогою таймера він періодично перевіряє чергу і, якщо є нові дані, оновлює відповідні елементи інтерфейсу. Такий патерн "виробник-споживач" гарантує, що інтерфейс програми залишається повністю чутливим до дій користувача навіть під час безперервного збору системних метрик.

3.3 Реалізація інструментів адміністрування

Окрім функцій пасивного моніторингу, розроблена система надає користувачеві інструменти для активного адміністрування та керування системою. Ці інструменти дозволяють не лише спостерігати за станом ПК, але й безпосередньо втручатися в його роботу для вирішення типових завдань, таких як завершення завислих процесів або керування встановленим програмним забезпеченням.Першим з таких інструментів є Менеджер процесів (рис. 3.4), реалізований у класі ProcessManagerWindow. Він надає користувачеві детальний список усіх активних 1 процесів у системі, що є аналогом стандартного "Диспетчера завдань". Для відображення даних використовується віджет tkinter.ttk.Treeview, який дозволяє представити інформацію у вигляді зручної таблиці з можливістю сортування по будь-якому стовпчику. Для отримання списку процесів та їхніх атрибутів (PID, ім'я, використання ЦП та пам'яті, ім'я користувача) використовується ітератор psutil.process_iter(). Цей метод є високоефективним, оскільки дозволяє уникнути проблем, пов'язаних зі зміною списку 1 процесів під час його обробки. Список процесів в інтерфейсі динамічно оновлюється кожні дві секунди, що дозволяє відстежувати зміни в реальному часі. Ключовою функцією є можливість примусового завершення процесу. При натисканні правою кнопкою миші на обраному процесі з'являється контекстне меню, яке дозволяє виконати команду psutil.Process(pid).terminate(), що надсилає процесу сигнал на завершення роботи.

Рисунок 3.4 – Робоче вікно менеджера процесів

Чітка таблична структура та можливість сортування дозволяють швидко ідентифікувати процеси і програми, що 17 споживають найбільше ресурсів. Реалізація цієї функціональності показує здатність програми не лише зчитувати, але й активно взаємодіяти з операційною системою на рівні керування процесами.Іншим важливим інструментом адміністрування є менеджер 17 встановленого програмного забезпечення, реалізований у класі InstalledProgramsWindow. Ця функція є специфічною для операційної системи Windows і надає можливість перегляду списку всіх програм, інстальованих у системі, а також запуску їх стандартних деінсталяторів.

Для отримання цих даних програма звертається безпосередньо до системного реєстру Windows за допомогою вбудованої бібліотеки winreg. Вона послідовно сканує ключові гілки реєстру, такі як HKEY_LOCAL_MACHINE та HKEY_CURRENT_USER за шляхами SOFTWARE\Microsoft\Windows\ CurrentVersion\Uninstall, де зберігається інформація про встановлене ПЗ. Для кожної знайденої програми зчитуються її назва, версія, видавець та дата встановлення. Для зручності користувача у вікні реалізовано функцію пошуку, що дозволяє миттєво фільтрувати довгий список програм.

Рисунок 3.5 – Вікно встанолених програм

Функція «Встановленні програми» надає користувачу повний та структурований перелік програмного забезпечення. Найважливішою функцією цього модуля є можливість деінсталяції. При виклику контекстного меню для обраної програми система зчитує з реєстру шлях до її деінсталятора (UninstallString) і запускає його за допомогою модуля subprocess. Таким чином, розроблений додаток надає зручний централізований інтерфейс для керування програмами, що є значно зручнішим, ніж стандартні засоби операційної системи.

3.4 Розробка комплексу для тестування системи

На додаток до моніторингу та адміністрування, в програмі реалізовано комплексний набір інструментів для поглибленої діагностики та перевірки стабільності системи під навантаженням. Цей функціонал дозволяє виявляти потенційні проблеми з апаратним забезпеченням та оцінювати продуктивність мережевих з'єднань. Він розділений на два основні модулі: "Центр стрес-тестування" та "Тест мережі".Центр стрес-тестування, реалізований у класі StressTestWindow, є найбільш технічно складною частиною програми. Він надає користувачеві можливість проводити інтенсивне навантажувальне тестування ключових компонентів: центрального процесора (ЦП), графічного процесора (ГП), оперативної пам'яті (ОЗП) та дискових накопичувачів. Для уникнення "зависання" графічного інтерфейсу під час виконання ресурсоємних завдань було застосовано паралельні обчислення. Тестування ЦП відбувається шляхом запуску окремого процесу для кожного логічного ядра за допомогою модуля multiprocessing. Кожен процес виконує нескінченний математичний цикл, що дозволяє досягти 100% завантаження всіх ядер і перевірити систему на перегрів та стабільність. Аналогічно, тест графічного процесора запускає в окремому процесі рендеринг 3D-сцени за допомогою бібліотеки OpenGL.Тестування диска реалізовано в окремому потоці (threading) і полягає у записі та подальшому зчитуванні великого тимчасового файлу, під час яких вимірюється швидкість виконання операцій. Протягом усього тестування користувач може в реальному часі спостерігати за ключовими показниками (температура, завантаження) на вбудованих графіках matplotlib, що дозволяє візуально контролювати процес. Продемонстрований на рисунку 3.6 інтерфейс дозволяє гнучко керувати процесом тестування та отримувати миттєвий візуальний фідбек. Реалізація

цього модуля підтверджує здатність програми виконувати складні, паралельні обчислення, одночасно підтримуючи стабільність та чутливість графічного інтерфейсу.Рисунок 3.6 – Центр Стрес-Тестування із графіками

Другим компонентом діагностичного комплексу є набір мережевих утиліт. Цей модуль надає користувачеві зручний графічний інтерфейс для виконання стандартних завдань з діагностики мережі. Функціонал розділено на вкладки: "Тест Швидкості", "Ping" та "Traceroute". Вкладка "Тест Швидкості" використовує популярну бібліотеку speedtest-cli для автоматичного пошуку оптимального сервера та вимірювання реальної швидкості завантаження, вивантаження та пінгу інтернет-з'єднання.Вкладки "Ping" та "Traceroute" надають можливість виконувати однойменні стандартні системні команди. Для цього програма використовує вбудований модуль subprocess, який запускає відповідну консольну утиліту у фоновому режимі. Важливо, що вивід цих команд перехоплюється в реальному часі та рядок за рядком відображається у текстовому полі в інтерфейсі програми. Це дозволяє користувачеві спостерігати за процесом виконання команди так само, як це відбувалося б у командному рядку, але в більш зручному графічному середовищі. Таким чином, розроблений набір утиліт, показаний на рисунку 3.7, перетворює програму на повноцінний діагностичний центр. Він демонструє здатність додатку інтегруватися як зі сторонніми Python-бібліотеками (speedtest-cli), так і зі стандартними інструментами операційної системи (ping, tracert), надаючи для них єдиний та зручний графічний інтерфейс.

Рисунок 3.7 – Зображено Інтерфейс модуля Тест Мережі

3.5 Тестування та оцінка ефективності системи

Фінальним етапом практичної реалізації програмного продукту є його комплексне тестування. Метою цього етапу була перевірка відповідності розробленої системи функціональним та нефункціональним вимогам, визначеним у розділі 2, а також виявлення та усунення можливих помилок. Тестування проводилося за кількома напрямками: функціональне тестування, тестування продуктивності та стабільності, а також оцінка зручності використання.Функціональне тестування полягало у послідовній перевірці працездатності всіх реалізованих модулів. Було перевірено відкриття кожного інформаційного вікна, коректність відображення статичних та динамічних даних. Особливу увагу було приділено адміністративним інструментам: успішно протестовано функцію завершення стороннього процесу через "Менеджер процесів", а також запуск деінсталятора програми через модуль "Встановлені програми". Усі діагностичні утиліти, включаючи ping, traceroute та тест швидкості, були перевірені на коректність виконання та відображення результатів.Тестування продуктивності та стабільності проводилося за допомогою вбудованого "Центру стрес-тестування". Було запущено комбінований 10-хвилинний стрес-тест для ЦП та ОЗП, протягом якого відстежувалась

стабільність роботи додатку та операційної системи. Програма продемонструвала стабільну роботу, графічний інтерфейс залишався чутливим до дій користувача завдяки винесенню навантаження в окремі процеси. Також було проведено моніторинг власного споживання ресурсів програми: в режимі спокою вона споживала менше 1% ресурсів ЦП та близько 80-100 МБ оперативної пам'яті, що є прийнятним показником.Модуль стрес-тестування має комплексний життєвий цикл, який залежить від вибору користувача. Процес переходу між станами, від очікування команди до вибору конкретного тесту та його завершення, наочно демонструє діяльнісна діаграма на рисунку 3.8

Рисунок 3.8 – Діаграма станів та переходів модуля «Центр стрес-тестування»

Посилання

Це джерела виділених збігів у вашому документі. Кожен збіг позначено темно-зеленим числом, яке відповідає вказаному тут джерелу. Джерела впорядковані за схожістю — чим вищий бал, тим сильніше збіг.

#	Джерело	%
1	metod.vntu.edu.ua	0.6%
2	ela.kpi.ua	0.3%
3	businessyield.com	0.2%
4	elartu.tntu.edu.ua	0.2%
5	essuir.sumdu.edu.ua	0.2%
6	lib.iitta.gov.ua	0.2%
7	plant.ranok.cx.ua	0.2%
8	188.190.43.194	0.2%
9	learn.ztu.edu.ua	0.2%
10	library.knuba.edu.ua	0.1%
11	ir.nmu.org.ua	0.1%
12	ecofin.at.ua	0.1%
13	sci-conf.com.ua	0.1%
14	elartu.tntu.edu.ua	0.1%
15	library.nuft.edu.ua	0.1%
16	eprints.library.odeku.edu.ua	0.1%
17	hostiko.com.ua	0.1%
18	dspace.znu.edu.ua	0.1%
19	sfa.org.ua	0.1%
20	it.nuft.edu.ua	0.1%

Plag

Дякуємо, що перевірили
свій документ за допомогою
Plag!